Edición Excel 365 / 2019

De 0 a 100 con macros y VBA

Sergio Propergol

ayudaexcel
Formación y recursos para Excelers

Índice

Introducción

¿Alguna vez has querido crear una hoja en un libro de Excel sin utilizar los comandos habituales, o crear un formulario totalmente automatizado para recoger datos y almacenarlos en una base de datos?

Si has respondido Sí a cualquiera de estas dos preguntas, estás leyendo el libro correcto.

Con este manual aprenderás todo lo que se puede hacer más allá de la interfaz de usuario estándar de Excel. Su objetivo es enseñarte a automatizar tareas repetitivas utilizando VBA, el lenguaje de programación que utilizan Excel y las demás aplicaciones de Office. Verás cómo ejecutando comandos e instrucciones especiales, vas a poder trabajar de forma más eficiente de lo que nunca llegaste a imaginar.

Mi nombre es Sergio Propergol. Comencé a utilizar Excel de forma habitual en 1999. Desde 2006 estoy al frente de Ayuda Excel ayudando a particulares y empresas en el desarrollo de plantillas y aplicaciones VBA para Excel. Lidero un grupo de expertos usuarios que, desde el foro de la web, damos ideas y soluciones a las dudas que nos plantean otros usuarios. Además, imparto clases *In company* a empresas, totalmente personalizadas y adaptadas al nivel y al campo de aprendizaje. Han pasado muchos años desde que comencé a utilizar Excel y he de decir que me sigue gustando como el primer día (o más).

Te doy la enhorabuena por unirte al grupo de EXCELERS que creen en la magia de Excel.

De 0 a 100 con macros y VBA está dividido en 24 capítulos que te introducirán progresivamente en la programación de Excel 365/2019, así como en el control de otras aplicaciones con Excel.

He diseñado el manual para los usuarios de Excel que quieren ampliar sus conocimientos y aprender utilizar la hoja de cálculo a un nivel superior, más allá de la interfaz de usuario.

Considera este manual como una formación individual y personalizada a la que puedes asistir en la comodidad de tu casa u oficina.

Algunas formaciones tienen requisitos previos y esta no es una excepción. ***De 0 a 100 con macros y VBA***, no explica, por ejemplo cómo seleccionar opciones de la cinta o a usar atajos de teclado. El manual asume que conoces la ubicación, en la cinta de opciones, de las herramientas para hacer cualquier tarea (tablas dinámicas, formatos condicionales, formato de celdas, etc.). Con estos simples requisitos, este manual te llevará al siguiente nivel de aprendizaje con Excel.

Deja que tus hojas de cálculo realicen cosas mágicas por ti…

Capítulo 1
VBA y la grabadora de macros

VBA,o *Visual Basic For Applications,* es el lenguaje de programación que incorporan tanto Excel como el resto de aplicaciones de Office.

Aprendiendo algunos comandos básicos de VBA puedes comenzar a automatizar muchas de las tareas rutinarias y repetitivas que realizas en Excel. En este capítulo aprenderás a utilizar la grabadora de macros y el editor de VBA para examinar y editar el código que hay detrás de una macro grabada.

Una macro es una serie de instrucciones que, al ejecutarse en el mismo orden en que se grabaron, pueden reducir el número de pasos necesarios para finalizar una tarea, disminuyendo significativamente el tiempo que pasas introduciendo datos, dándoles formato y analizándolos.

Puedes grabar una macro utilizando la grabadora de macros o puedes escribirla desde cero utilizando el editor de Visual Basic.

También puedes combinar macros grabadas con tu propio código de programación para crear aplicaciones VBA únicas que se ajusten a tus necesidades diarias. Tanto si escribes o grabas estarás utilizando el poderoso lenguaje de programación VBA.

Excel cuenta con cientos de características y herramientas que permiten trabajar de forma eficiente. Antes de ponerte a automatizar tareas, ya sea grabando una macro o escribiéndola desde cero, asegúrate de que no exista ya una de estas herramientas que puedas usar para realizar la tarea. Considera la posibilidad de grabar o escribir una macro cuando te encuentres realizando tareas repetitivas o Excel no cuente con una herramienta para realizar ese trabajo.

Únicamente aprendiendo a manejar la grabadora de macros de Excel y a modificar esa grabación con un nivel básico de VBA, serás capaz de automatizar cualquier parte de tu hoja de trabajo. Por ejemplo, puedes automatizar la introducción de datos grabando una macro que sustituya los títulos de las columnas por los encabezados de los datos. Aplicando un poco de lógica condicional podrás comprobar automáticamente si existe información duplicada en un rango de celdas concreto. Con una macro también podrías aplicar formato rápidamente a varias hojas del

libro (fuentes, colores, bordes, sombreado…) Las macros te ahorrarán el tener que pulsar el teclado cuando se trate de establecer áreas de impresión, márgenes, encabezados, etc.

1.1 Extensiones de archivos para macros

Cuando un libro de Excel contiene código de programación, debe ser guardado en uno de los siguientes formatos:

- Libro de Excel habilitado para macros (.xlsm).
- Libro de Excel binario (.xlsb).
- Plantilla de Excel habilitada para macros (.xltm).

Si intentas guardar el libro en un formato incompatible con macros, Excel te advertirá con el siguiente mensaje:

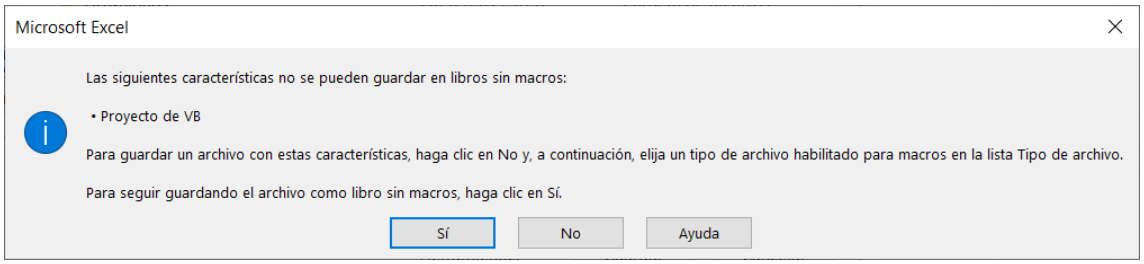


Imagen 2.1 Cuando un libro contiene código de programación, debes guardarlo en un tipo de archivo habilitado para macros en lugar de un archivo normal de libro .XLSX.

1.2 Configuración de seguridad de las macros

Como las macros pueden contener código malicioso para introducir virus en tu equipo, es importante entender las diferentes configuraciones de seguridad disponibles. También es fundamental que se ejecute un software antivirus actualizado en tu ordenador. Este software escaneará el libro de Excel que intentas abrir, si éste contiene macros. El valor predeterminado de la configuración de seguridad es tener deshabilitadas todas las macros con notificación, tal y como se muestra en la Imagen 2.2.

Para utilizar los componentes deshabilitados debes hacer clic en el botón **Habilitar contenido**. Esto agregará el libro a la lista de documentos de confianza. La próxima vez que abras este archivo no volverá a aparecer el mensaje, habilitándose automáticamente las macros. En caso de que necesites más información antes de habilitar el contenido, puedes hacer clic en el texto del mensaje para activar el menú **Archivo**, donde encontrarás una explicación sobre el contenido que ha sido desactivado.

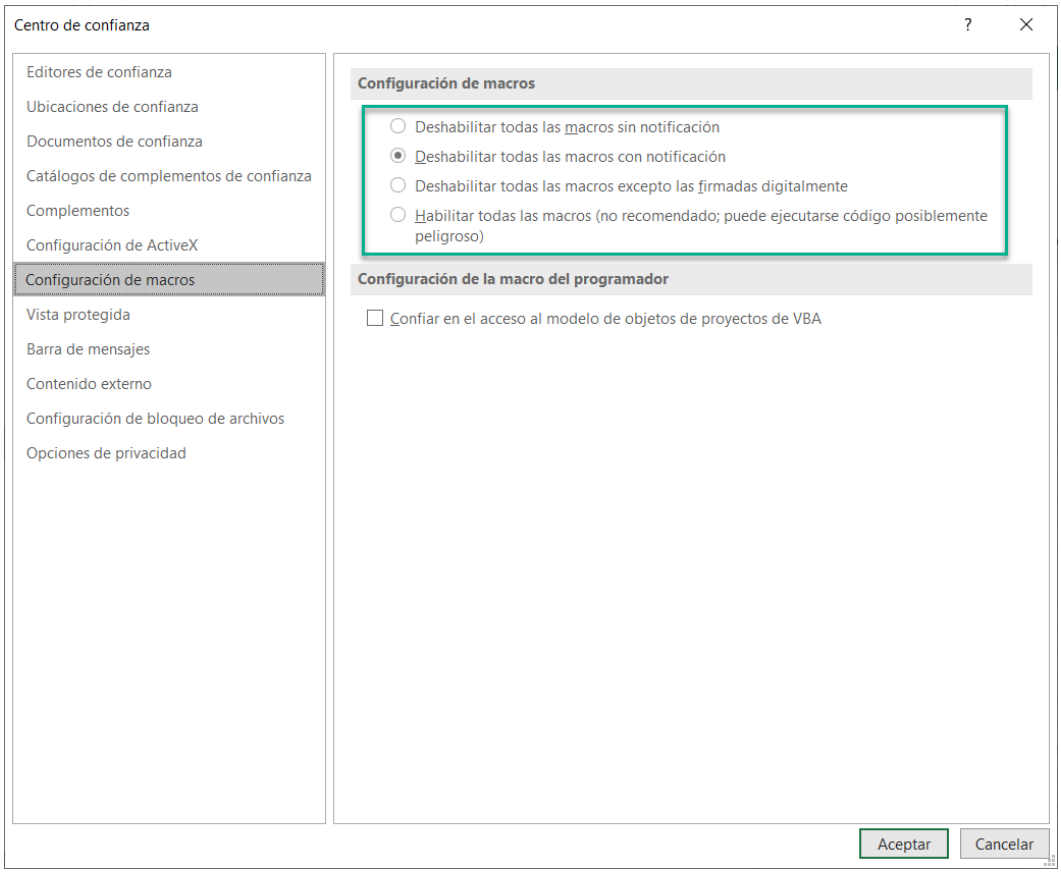


Imagen 2.2 Las opciones de Configuración de macros en el Centro de Confianza te permiten controlar cómo debe tratar Excel las macros cuando están presentes en un libro de trabajo abierto. Para abrir las Configuraciones de macros del Centro de confianza, selecciona

De este modo, si existen macros en un libro que estés intentando abrir, recibirás un mensaje de advertencia de seguridad inmediatamente debajo de la cinta de opciones:

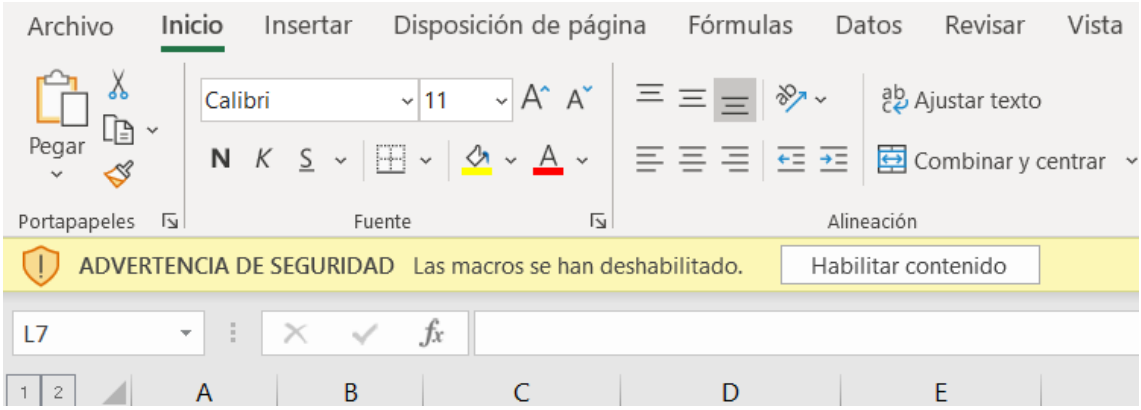


Imagen 2.3 Al abrir un libro de trabajo con macros, Excel muestra un mensaje de advertencia de seguridad.

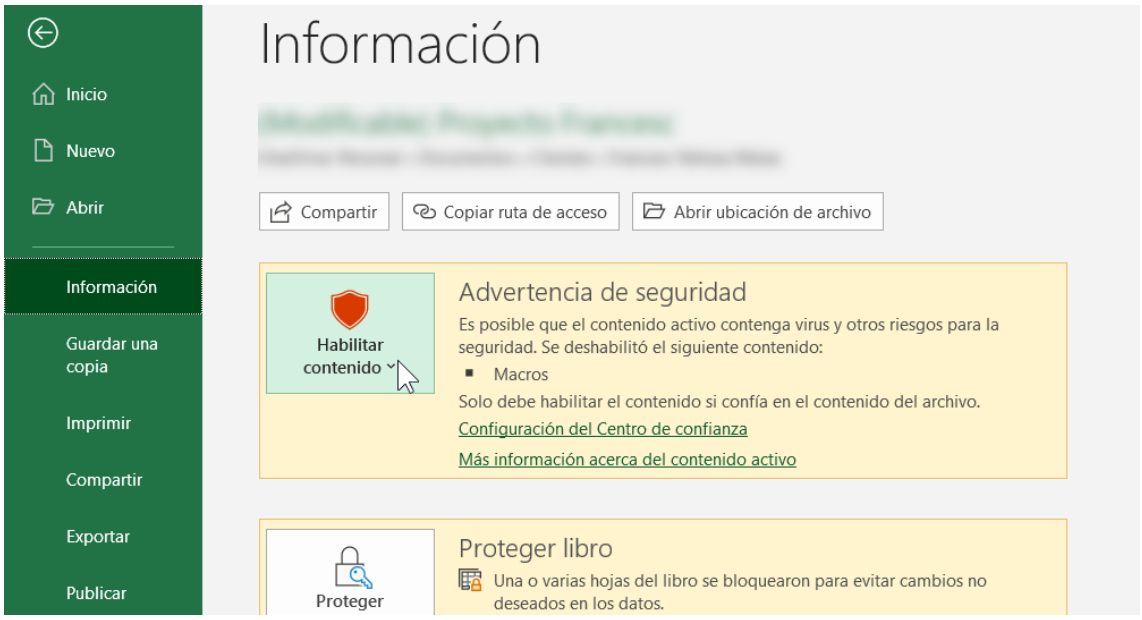


Imagen 2.4 En el menú Archivo

Al hacer clic en el botón **Habilitar contenido**, aparecerán dos opciones:

- Habilitar todo el contenido: Tiene el mismo efecto que hacer clic en el botón Habilitar contenido de la barra de mensajes, es decir, habilitará las macros y convertirá el archivo en un documento de confianza.
- Opciones avanzadas: Muestra el cuadro de diálogo **Opciones de seguridad de Microsoft Office**, que contiene opciones para habilitar el contenido para "esta sesión".

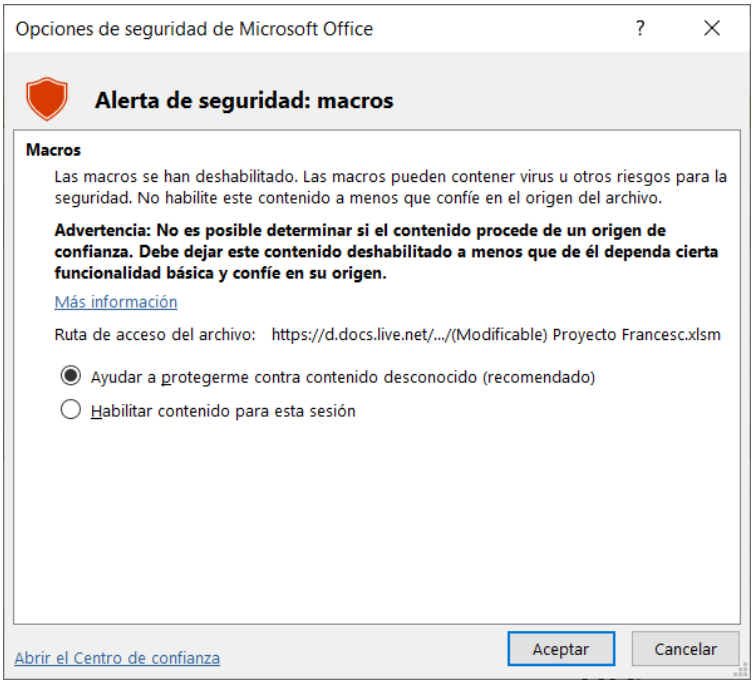


Imagen 2.5 Las macros deshabilitadas pueden activarse para la sesión actual en el cuadro Opciones de seguridad de Microsoft Office.

2 Habilitar la ficha Programador

Para facilitar el trabajo con los archivos de ejercicios de este manual, debes preparar el entorno, haciendo visible la ficha **Programador** (en otras versiones llamada Desarrollador) y cambiando la configuración de seguridad de las macros. Sigue estos pasos:

1. Crea una carpeta en la unidad C:\ de tu equipo (para que sea más accesible) y llámala "Archivos Manual VBA".
2. Abre Excel en un libro nuevo.
3. Selecciona **Archivo > Opciones**.
4. En el cuadro de diálogo **Opciones de Excel**, haz clic en el menú **Personalizar** cinta de opciones. En la sección derecha activa la casilla **Programador** y haz clic en **Aceptar**. Esto hará que se muestre la siguiente ficha en la cinta de opciones:

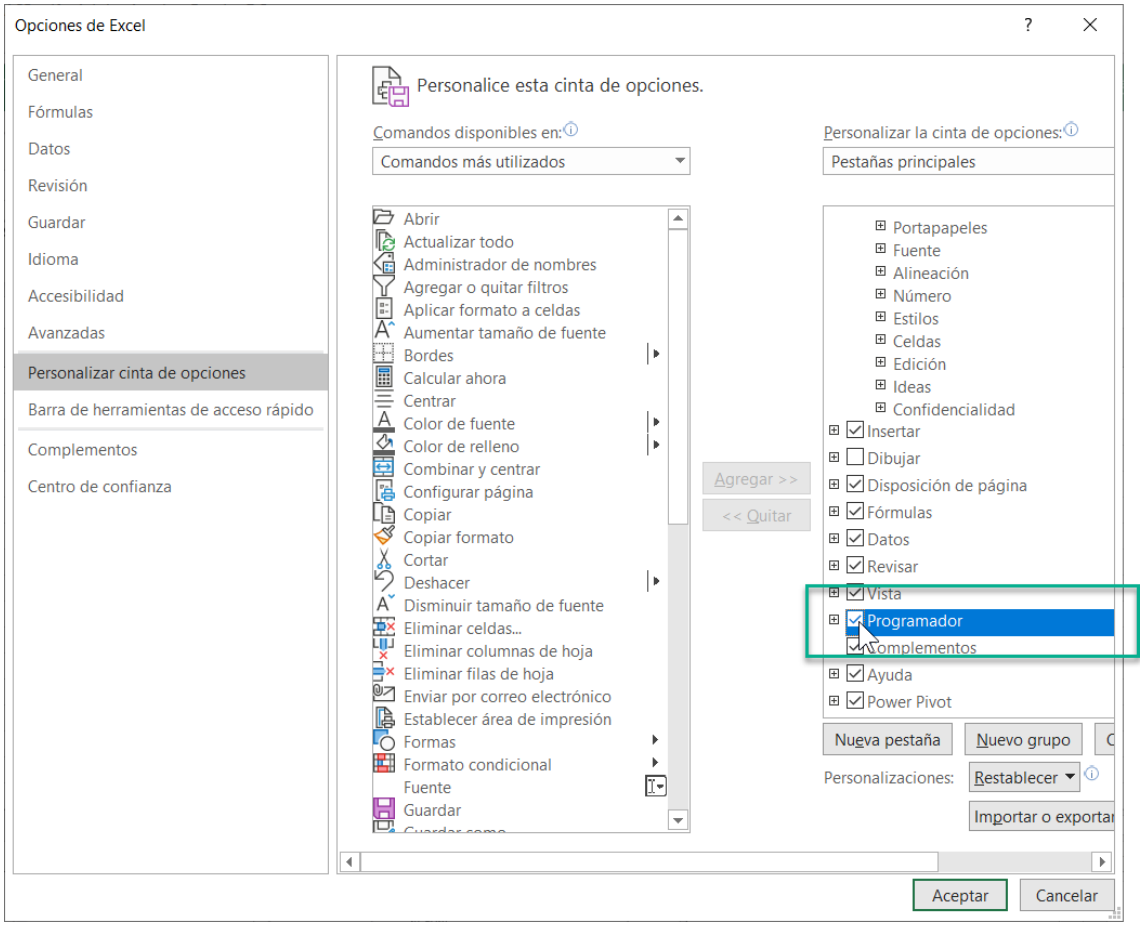


Imagen 2.1 Para habilitar la pestaña Programador en la Cinta, utiliza el cuadro de diálogo Opciones de Excel y selecciona Personalizar la Cinta.

5. En el grupo **Código** de la ficha **Programador**, haz clic en el botón **Seguridad** de las macros.

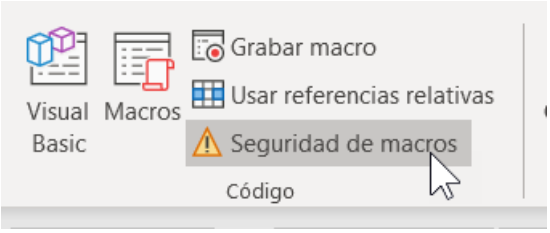


Imagen 2.2 Utiliza el botón Seguridad de macros en el grupo Código de la pestaña Programador para personalizar la configuración de seguridad de las macros.

Aparece el cuadro **Centro de confianza**.

6. En el panel izquierdo del cuadro haz clic en **Ubicaciones de confianza** que, como puedes comprobar, ya contiene algunas carpetas predefinidas que se crearon al instalar Excel.
7. Haz clic en **Agregar nueva ubicación**.
8. Haz clic en el botón **Examinar** para buscar la carpeta que has creado en el paso 1.

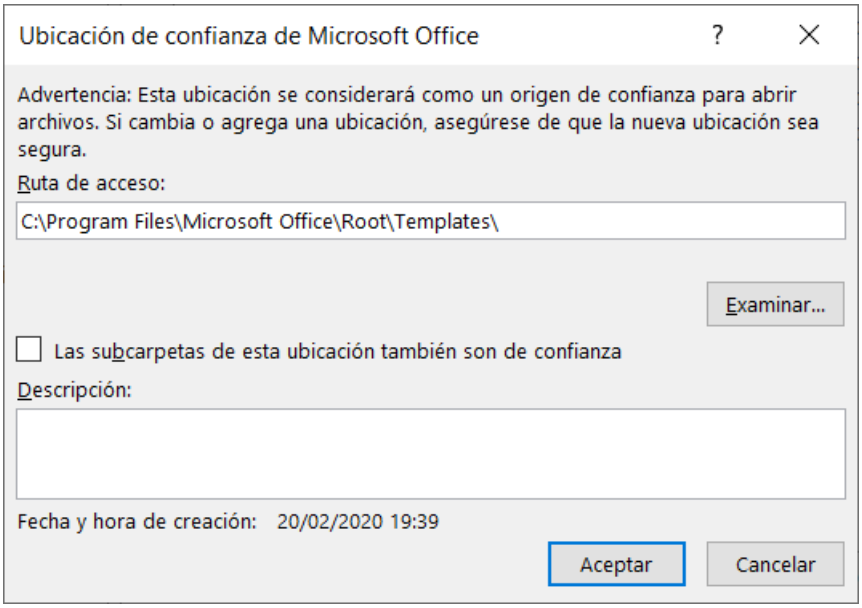


Imagen 2.3 Designar una carpeta de Ubicación de Confianza para los ejemplos de programación de este libro.

9. Haz clic en **Aceptar** para cerrar el cuadro.
10. Observa que el listado de ubicaciones ahora contiene la ruta que acabas de introducir: C:\Archivos Manual VBA\. Los archivos colocados en una ubicación de confianza pueden abrirse sin ser revisados por la función de seguridad del Centro de Confianza. Haz clic en **Aceptar** para cerrar el cuadro de diálogo del **Centro de confianza**.

Por último, introduce en la carpeta Archivos Manual VBA, todos los archivos del manual.

3 Cómo utilizar la grabadora de macros

En esta sección grabaremos algunas macros cortas que realizan tareas de introducción de datos y formateo de celdas de una hoja de Excel. Aprenderás a:

- Planificar tus macros.
- Grabar pulsaciones de teclas y acciones.
- Editar y mejorar el código grabado.
- Varias formas de ejecutar macros.
- Solucionar problemas básicos en caso de que aparezca algún error.
- Guardar tus macros, renombrarlas, combinarlas e imprimirlas.

3.1 Planificar una macro

Antes de ponerte manos a la obra, debes pararte un momento a pensar qué quieres que haga la macro.

La forma más fácil de planificar una macro es realizar manualmente las acciones que debe repetir la macro. A medida que vayas realizando una acción nueva, escríbela en un papel de la misma forma en que la haces. Puedes apuntar también qué resultados obtienes con cada acción.

Como una grabadora de voz, la grabadora de macros graba cada acción que realizas. Si no planificas la macro antes de grabarla, puedes terminar con acciones innecesarias que no solo ralentizarán la macro, sino que también requerirá más tiempo de edición posterior para que funcione como debería. Aunque es más fácil editar la macro que eliminar las palabras de una grabación de voz, si creas solo las acciones necesarias, ahorrarás tiempo y problemas de edición más adelante.

	A	B	C	D	E	F
1	Nombre del empleado	Nombre	Apellido	Importe / hora	Horas trabajadas	Total a pagar
2	Javier Jimeno	Javier	Jimeno	15	7	105,00 €
3	Sergio Segura	Sergio	Segura	13,4	6	80,40 €
4	Eugenio Esteban	Eugenio	Esteban	21,42	10	214,20 €
5	Mario Martínez	Mario	Martínez	16,5	11	181,50 €
6	Yolanda Yebra	Yolanda	Yebra	35	21	735,00 €
7	Diego Díaz	Diego	Díaz	28,33	14	396,62 €
8						
9						

Imagen 3.1 Esta es la hoja de muestra que se creará y formateará con la ayuda de la grabadora de macros.

Supón que te piden crear con VBA la hoja que aparece en la Imagen 3.1. No te preocupes, utilizar la grabadora de macros es muy sencillo.

Comencemos por identificar las tareas necesarias:

1. Insertar una hoja en un libro y renombrarla como "*Salarios de empleados*".
2. Introducir los títulos de las columnas en la primera fila y aplicar el formato requerido (tamaño de las columnas y estilos de fuente).
3. Introducir los datos de los empleados (Nombre completo, importe/hora y horas trabajadas).
4. Introducir las fórmulas para extraer el nombre del empleado.
5. Introducir las fórmulas para extraer el apellido del empleado.
6. Introducir las fórmulas para calcular el importe total a pagar a cada empleado.
7. Aplicar formato de tabla a los datos.

En lugar de crear una macro que realice todas las tareas, crearemos una macro diferente para cada tarea. Esto te dará la oportunidad de aprender el código de varias macros simples y a crear una "*macro principal*".

Crea un libro nuevo y llámalo "*Capítulo 1 – Inicio.xlsm*". Guárdalo en la carpeta "Archivos Manual VBA" que creaste en C:\. Observa que debes guardar el archivo con la extensión .xlsm, que es la adecuada para almacenar macros. Mantén el archivo abierto, pues lo usaremos para todas las macros del capítulo.

3.2 Grabar una macro

Antes de grabar la macro debes decidir si quieres que se grabe la posición de la celda activa. Si quieres que la macro siempre empiece en un lugar específico de la hoja, enciende primero la grabadora de macros y a continuación selecciona la celda donde quieres que comience. Si la macro puede comenzar en cualquier celda, selecciona una celda y a continuación enciende la grabadora de macros.

Nombres de macros

SI TE OLVIDAS DE DARLE UN NOMBRE A LA MACRO, EXCEL LE ASIGNA UN NOMBRE POR DEFECTO (MACRO1, MACRO2…) LOS NOMBRES DE LAS MACROS PUEDEN CONTENER LETRAS, NÚMEROS Y EL CARÁCTER DE SUBRAYADO, PERO EL PRIMER CARÁCTER DEBE SER UNA LETRA. NO SE PUEDEN USAR ESPACIOS

3.2.1 Insertar una hoja y renombrarla (Tarea 1)

1. Haz clic en la ficha **Programador** (Desarrollador en algunas versiones de Excel) y a continuación en **Grabar macro**.
2. En el cuadro de diálogo **Grabar macro** introduce el nombre `Insertar_nueva_hoja`, como se muestra en la Imagen 3.2. No cierres el cuadro hasta que no hayas finalizado de introducir los datos.
3. Selecciona **Este libro** en el cuadro **Guardar macro en**:

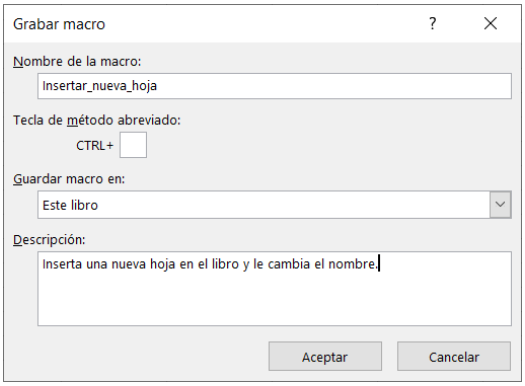


Imagen 3.2 Cuando grabas una nueva macro, debes ponerle un nombre. En el cuadro de diálogo Grabar macro, también puedes indicar una tecla de acceso directo, el lugar de almacenamiento y una descripción para tu macro.

Dónde almacenar la macro

EXCEL PERMITE ALMACENAR LAS MACROS EN TRES LUGARES:

- LIBRO DE MACROS PERSONAL: PODRÁS USAR LAS MACROS CUANDO ABRAS EXCEL. PUEDES ENCONTRAR ESTE LIBRO EN LA CARPETA XLSTART. SI ESTE LIBRO NO EXISTE, EXCEL LO CREARÁ LA PRIMERA VEZ QUE SELECCIONES ESTA OPCIÓN.
- LIBRO NUEVO: SE CREARÁ UN NUEVO LIBRO Y SE UBICARÁ EN ÉL LA MACRO.
- ESTE LIBRO: LA MACRO SE ALMACENA EN EL LIBRO QUE ESTÁS USANDO EN ESTE MOMENTO.

4. En el apartado **Descripción** introduce el texto "*Inserta una nueva hoja en el libro y le cambia el nombre*".
5. Haz clic en **Aceptar** para cerrar el cuadro de diálogo **Grabar macro**. En la barra de estado aparece el botón **Detener grabación** que se muestra en la Imagen 3.3. No hagas clic en el botón hasta que se indique. Cuando este botón aparece, significa que el libro está en modo de grabación.

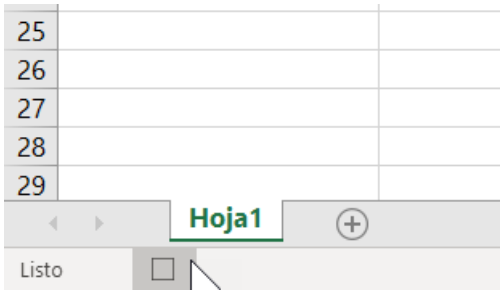


Imagen 3.3 El botón Detener grabación de la barra de estado indica que el modo de grabación de macro está activo.

El botón **Detener grabación** permanece en la barra de estado mientras se graba la macro. Solo se registran las acciones que se finalizan presionando la tecla **Intro** o al hacer clic en el botón **Aceptar**.

Si haces clic en el botón **Cancelar** o presionas la tecla **Esc**, la grabadora no graba esa acción.

6. Añade una nueva hoja al libro. Puedes hacerlo haciendo clic con el botón derecho del ratón en la pestaña de la Hoja1 y seleccionando **Insertar > Hoja de cálculo**, o haciendo clic en el botón redondo **+** que aparece a la derecha de Hoja1.
7. Cambia el nombre a la hoja creada. Llámala "Salarios de empleados".
8. Haz clic en **Detener grabación** en la barra de estado, como se muestra en la Imagen 3.3, o selecciona la ficha **Programador** y a continuación haz clic en el botón **Detener grabación**.

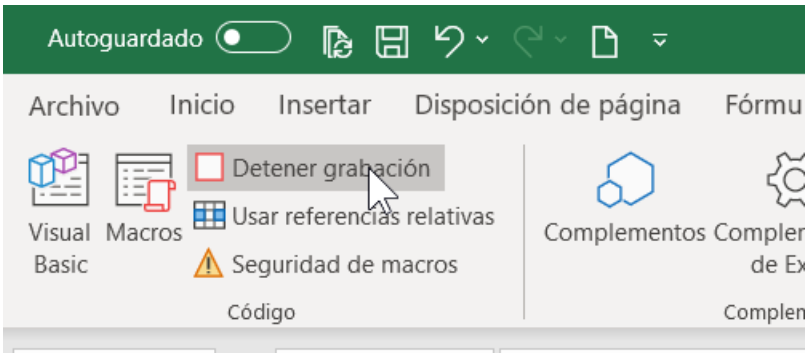


Imagen 3.4 También se puede detener la grabación desde la cinta de opciones.

Acabas de grabar tu primera macro. Excel ha escrito todas las instrucciones necesarias para ejecutar las acciones que has realizado. Ahora vamos a continuar grabando el resto de las acciones planificadas anteriormente. Después tendrás la oportunidad de revisar el código de las macros y probarlas.

3.2.2 Insertar encabezados de columnas y aplicar formato (Tarea 2)

1. Selecciona la ficha **Vista**. A continuación, haz clic en el desplegable del botón **Macros** y en **Grabar macro**. También puedes hacer clic en el botón **Grabar macro** de la barra de estado.
2. Introduce "*Insertar_encabezados*" como nombre para la macro.
3. Asegúrate de que esté seleccionada la ubicación **Este libro**.
4. Haz clic en **Aceptar**.
5. En este momento Excel enciende la grabadora. A partir de ahora todas las acciones se grabarán.
6. Selecciona la celda A1 e introduce el primer encabezado (Nombre completo).
7. Desplázate a la celda B1 e introduce "Nombre".
8. Introduce el resto de los encabezados en las celdas C1:F1 (Apellido, Importe / hora, Horas trabajadas y Total a pagar).
9. Selecciona el rango A1:F1 y aplica el formato Negrita presionando la combinación de teclas Ctrl + N (Ctrl + B para la configuración inglesa).
10. Cambia el tamaño de la fuente a 12 pt con el selector de tamaños (no con los botones para aumentar o disminuir).
11. Con el rango A1:F1 todavía seleccionado haz clic en **Inicio > Celdas > Formato > Autoajustar ancho de columna**.
12. Detén la grabación de la macro desde la barra de estado.

¿Referencias relativas o absolutas?

La grabadora de macros puede registrar tus acciones utilizando referencias de celda absolutas o relativas (ver la imagen 3.5).

- Para ejecutar la macro siempre en las mismas celdas sin importar cuál sea la celda activa en el momento de la ejecución, debes utilizar referencias absolutas (el modo por defecto). En este caso las referencias de las celdas tendrán el formato $A$1:$C$5. Antes de comenzar a grabar la macro asegúrate de que el botón **Usar referencias relativas** está desactivado, como se muestra en la Imagen 3.5.
- Si quieres que la macro realice las acciones en cualquier celda, asegúrate de seleccionar previamente el botón **Usar referencias relativas** antes de grabarla. Las referencias relativas tienen la forma A1:C5. La grabadora continuará usando referencias relativas hasta que salgas de Excel o desactives el botón de las referencias.

Durante el proceso de grabación puedes utilizar ambos métodos. Por ejemplo, puedes seleccionar una celda específica (por ejemplo, $A$4), realizar una acción y luego elegir otra celda relativa a la seleccionada (por ejemplo, C9, que se encuentra cinco filas hacia abajo y dos columnas hacia la derecha de la celda A4). Las referencias relativas se ajustan automáticamente y las absolutas no.

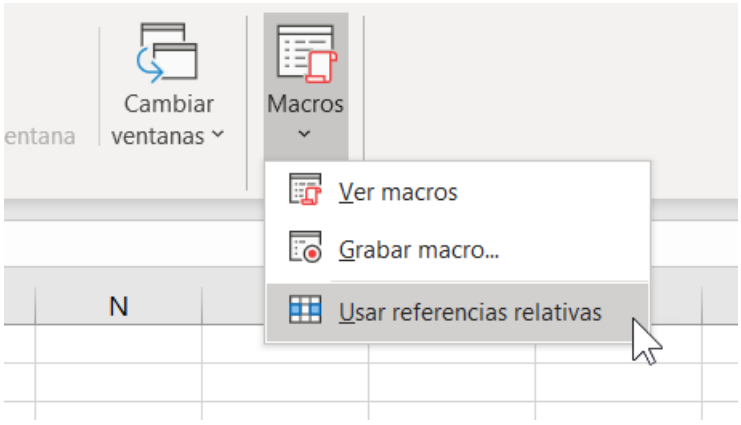


Imagen 3.5 La grabadora de macros puede registrar tus acciones utilizando referencias de celdas absolutas o relativas. Para elegir cualquiera de las dos opciones haz clic en Vista > Macros > Usar referencias relativas.

3.2.3 Introducir datos de los empleados (Tarea 3)

1. Selecciona **Vista > Macros > Ver macros** (también puedes hacer clic en el botón Grabar macro de la barra de estado).
2. Introduce "*Datos_Empleados*" como nombre de la macro.
3. Asegúrate de que el cuadro **Guardar macro en** tiene seleccionada la opción "Este libro".
4. Haz clic en **Aceptar**.

5. La grabadora de macros se encenderá y todas las acciones a partir de ahora se grabarán.
6. Introduce los datos de los empleados en las columnas A, D y E como se muestra en la Imagen 3.1.
7. Deja en blanco las columnas Nombre, Apellido y Total a pagar, pues se rellenarán más tarde.
8. Haz clic en el botón **Detener la grabación** de la barra de estado o haz clic en la ficha **Vista > Macros > Detener la grabación**.
9. Acabas de grabar la tercera macro. Has completado la introducción de datos. Ahora grabarás las macros que utilizarán fórmulas para rellenar las demás columnas.

3.2.4 Introducir fórmula para extraer el nombre (Tarea 4)

1. Selecciona **Vista > Macros > Ver macros** o pulsa el botón **Grabar macro** en la barra de estado.
2. Introduce "*Obtener_nombre*" como nombre de la macro.
3. Asegúrate de que el cuadro **Grabar macro** en tiene seleccionada la opción "Este libro". Si no es así, despliega el cuadro y haz clic en ella.
4. Haz clic en **Aceptar**.
5. Como en la tarea anterior, la grabadora de macros se enciende.
6. Introduce la siguiente fórmula en la celda B2:

   =IZQUIERDA(A2;ENCONTRAR(" ";A2)-1)

7. Copia las fórmulas en las celdas B3:B7 arrastrando el controlador de relleno (el pequeño cuadrado negro de la esquina inferior derecha de la celda) hacia abajo.
8. Haz clic en el botón **Detener grabación** en la barra de estado.

3.2.5 Introducir fórmula para extraer el apellido (Tarea 5)

1. Selecciona **Vista > Macros > Ver macros** o pulsa el botón **Grabar macro** en la barra de estado.
2. Introduce "*Obtener_apellido*" como nombre de la macro.
3. Asegúrate de que el cuadro **Grabar macro** en tiene seleccionada la opción "Este libro". Si no es así, despliega el cuadro y haz clic en ella.
4. Haz clic en **Aceptar**.
5. Introduce la siguiente fórmula en la celda C2:

   =DERECHA(A2;LARGO(A2)-ENCONTRAR(" ";A2))

6. Arrastra la fórmula a las celdas C3:C7 mediante el controlador de relleno. Verás que el resto de apellidos aparecen en las celdas.
7. Detén la grabación.

3.2.6 Introducir fórmula para calcular el importe a pagar a cada empleado (Tarea 6)

1. Selecciona **Vista > Macros > Ver macros** o pulsa el botón **Grabar macro** en la barra de estado.
2. Introduce "*Calcular_importes*" como nombre de la macro.

3. Asegúrate de que el cuadro **Grabar macro** en tiene seleccionada la opción “Este libro”. Si no es así, despliega el cuadro y haz clic en ella.
4. Haz clic en **Aceptar**.
5. Introduce la siguiente fórmula en la celda F2:

   =D2*E2

6. Arrastra la fórmula para obtener el cálculo en todas las celdas.
7. Aplica el formato Moneda a todas las celdas seleccionadas.
8. Detén la grabación.

3.2.7 Aplicar formato de tabla (Tarea 7)

1. Selecciona **Vista > Macros > Ver macros** o pulsa el botón **Grabar macro** en la barra de estado.
2. Introduce “*Formato_tabla*” como nombre de la macro.
3. Asegúrate de que el cuadro **Grabar macro** en tiene seleccionada la opción “Este libro”. Si no es así, despliega el cuadro y haz clic en ella.
4. Haz clic en **Aceptar**.
5. Selecciona todos los datos de la hoja y haz clic en **Inicio > Estilos > Dar formato como tabla**. Selecciona cualquiera de los estilos de tabla predefinidos.
6. Selecciona la celda A1.
7. Detén la grabación.

Al crear todas estas macros acabas de generar bastantes líneas de código VBA. A continuación, buscaremos dónde se encuentra ese código y lo examinaremos para optimizarlo.

3.3 Editar el código grabado

Para poder editar tus macros, debes conocer la ubicación donde se han almacenado.

Como recordarás, al grabar todas las macros, estaba selecciona la opción “Este libro” en el desplegable **Guardar macro en**.

Para encontrar los códigos sigue estos pasos:

1. Haz clic en el botón **Macros** de la ficha **Vista**. Aparecerá un cuadro de diálogo con las macros que acabas de grabar. (ver Imagen 3.6).
2. Selecciona la macro `Insertar_nueva_hoja` y haz clic en el botón **Editar/Modificar**.

   Excel abrirá una nueva ventana. Se trata del editor de VBA (también conocido como VBE), como se muestra en la Imagen 3.7. Esta ventana es el entorno de programación VBA, al cual también puedes acceder pulsando **Alt+F11**.

3. Para volver a la ventana de Excel, simplemente cierra la ventana del editor.

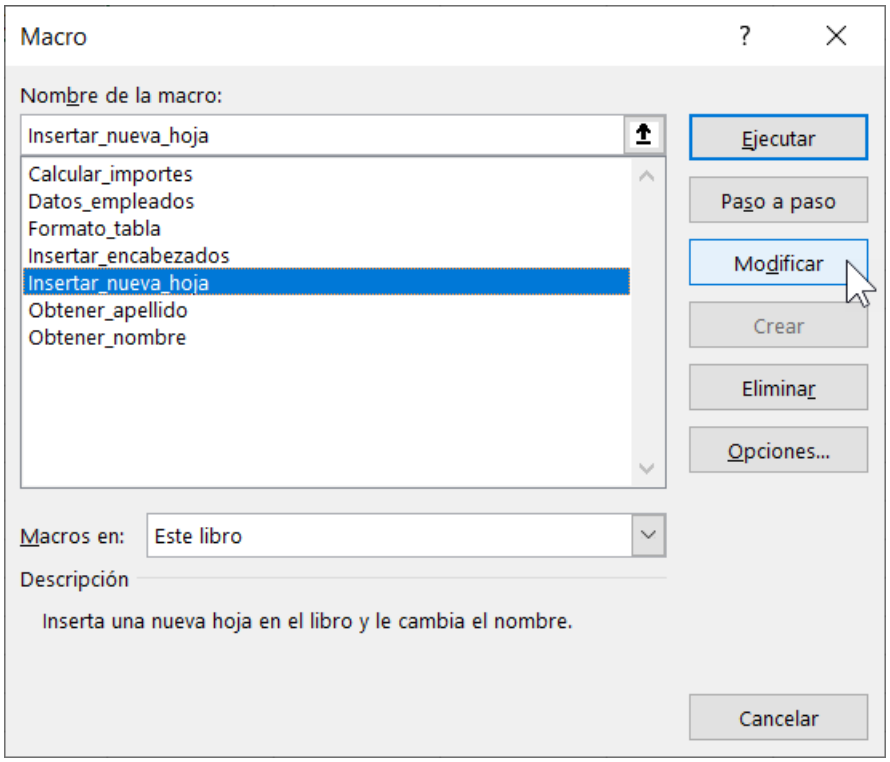


Imagen 3.6 En el cuadro de diálogo Macro puedes seleccionar una macro para ejecutarla, depurarla (paso a paso), editarla o borrarla. También puedes configurar las opciones de macro.

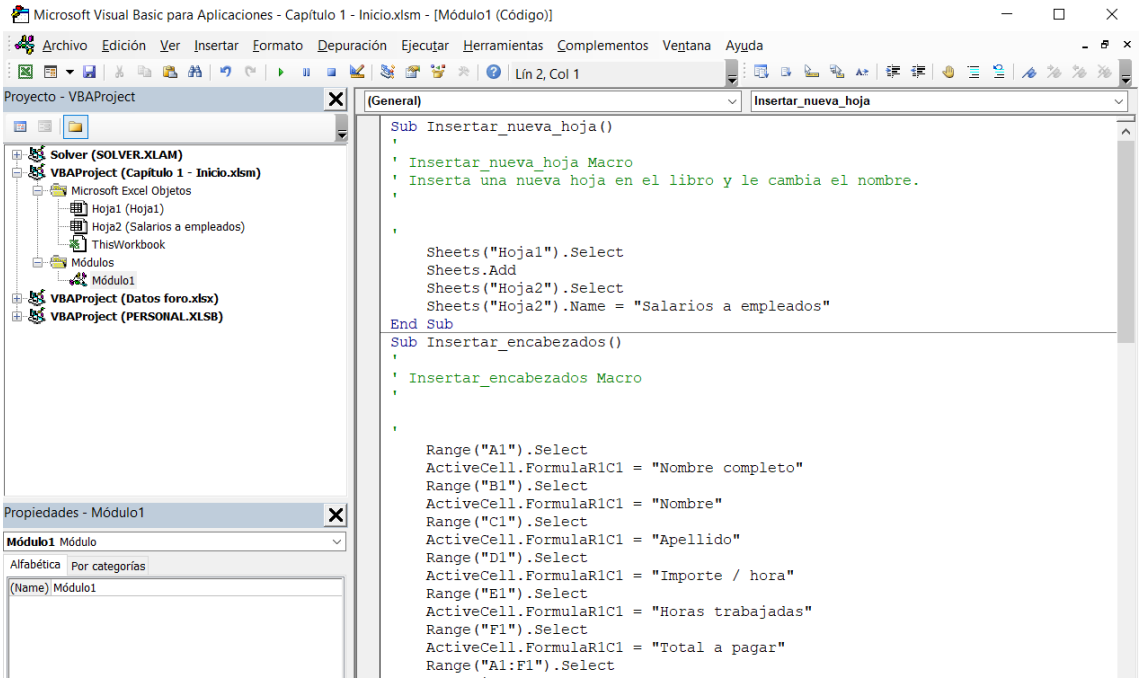


Imagen 3.7 La ventana del editor de VBA se usa para editar macros y también para escribir nuevos procedimientos.

No te preocupes si no entiendes los elementos de la ventana del editor. A medida que vayas trabajando con las macros grabadas y empieces a escribir tus propios procedimientos, te familiarizarás con todos ellos.

4. En la ventana de Excel selecciona la ficha **Programador > Visual Basic** para cambiar al entorno de programación.

   Fíjate en el menú y en la barra de herramientas del editor de VBA. Como puedes ver son totalmente diferentes. El editor utiliza la antigua barra de menús que se usaba en Excel 2003. En estos dos elementos se encuentran todas las herramientas necesarias para programar y probar las macros y procedimientos VBA. A medida que vayas profundizando en el libro, te sentirás más cómodo utilizando estas herramientas.

La parte principal de la ventana del editor de VBA es una superficie donde se acoplan varias ventanas. Éstas te serán muy útiles mientras creas y pruebas las macros.

La Imagen 3.7 muestra tres ventanas: el Explorador de proyectos, la ventana **Propiedades** y la ventana **Código**. La ventana del explorador de proyectos muestra una carpeta de módulos abierta.

Excel graba tus acciones en "almacenes" llamados Módulo1, Módulo 2, etc. Más adelante también utilizarás módulos para almacenar tu código personalizado.

Un módulo se parece a un documento en blanco de Word.

La ventana **Propiedades** muestra las propiedades del objeto que se encuentra seleccionado en ese momento en la ventana del **Explorador de proyectos**.

En la Imagen 3.7 puedes ver que en el **Explorador de proyectos** se encuentra seleccionado el Módulo1 y por lo tanto, la ventana **Propiedades** muestra las propiedades del Módulo 1.

Observa que la única propiedad del módulo es la propiedad `Name`. Puedes usar esta propiedad para cambiarle el nombre al módulo por otro más significativo.

¿Macro o procedimiento?

Una macro es una serie de comandos grabados con la ayuda de la grabadora de macros o introducida manualmente en un módulo de VBA. El término "macro" se sustituye muy a menudo por el término más amplio "procedimiento". Aunque los dos se pueden utilizar indistintamente, muchos programadores se inclinan más por "procedimiento".

La ventana **Código** del Módulo1 muestra el código de todas las macros que grabaste.

Por favor, ten en cuenta que el siguiente código puede no coincidir con el que se muestra en tu ventana. Excel registra todas las acciones mientras la grabadora está encendida, así que puedes ver más o menos acciones grabadas:

```
Sub Insertar_nueva_hoja()
'
```

```
' Insertar_nueva_hoja Macro
' Inserta una nueva hoja en el libro y le cambia el nombre.
'

'
    Sheets("Hoja1").Select
    Sheets.Add
    Sheets("Hoja2").Select
    Sheets("Hoja2").Name = "Salarios a empleados"
End Sub
Sub Insertar_encabezados()
'
' Insertar_encabezados Macro
'

'
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Nombre completo"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Nombre"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "Apellido"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "Importe / hora"
    Range("E1").Select
    ActiveCell.FormulaR1C1 = "Horas trabajadas"
    Range("F1").Select
    ActiveCell.FormulaR1C1 = "Total a pagar"
    Range("A1:F1").Select
    Selection.Font.Bold = True
    With Selection.Font
        .Name = "Calibri"
        .Size = 14
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ThemeColor = xlThemeColorLight1
```

```
        .TintAndShade = 0
        .ThemeFont = xlThemeFontMinor
    End With
        Selection.Columns.AutoFit
End Sub
Sub Datos_empleados()
'
' Datos_empleados Macro
'

'
    Range("A2").Select
    ActiveSheet.Next.Select
    ActiveSheet.Previous.Select
    ActiveCell.FormulaR1C1 = "Javier Jimeno"
    Range("D2").Select
    ActiveCell.FormulaR1C1 = "17"
    Range("E2").Select
    ActiveCell.FormulaR1C1 = "7"
    ActiveSheet.Next.Select
    ActiveSheet.Previous.Select
    Range("A3").Select
    ActiveCell.FormulaR1C1 = "Sergio Segura"
    Range("D3").Select
    ActiveCell.FormulaR1C1 = "13.4"
    Range("E3").Select
    ActiveCell.FormulaR1C1 = "6"
    ActiveSheet.Next.Select
    ActiveSheet.Previous.Select
    Range("A4").Select
    ActiveCell.FormulaR1C1 = "Eugenio Esteban"
    Range("D4").Select
    ActiveCell.FormulaR1C1 = "21.42"
    Range("E4").Select
    ActiveCell.FormulaR1C1 = "10"
    ActiveSheet.Next.Select
    ActiveSheet.Previous.Select
    Range("A5").Select
    ActiveCell.FormulaR1C1 = "Mario Martinez"
    Range("D5").Select
```

```
    ActiveCell.FormulaR1C1 = "16.5"
    Range("E5").Select
    ActiveCell.FormulaR1C1 = "11"
    ActiveSheet.Next.Select
    ActiveSheet.Previous.Select
    Range("A6").Select
    ActiveCell.FormulaR1C1 = "Yolanda Yebra"
    Range("D6").Select
    ActiveCell.FormulaR1C1 = "35"
    Range("E6").Select
    ActiveCell.FormulaR1C1 = "21"
    ActiveSheet.Next.Select
    ActiveSheet.Previous.Select
    Range("A7").Select
    ActiveCell.FormulaR1C1 = "Diego Díaz"
    Range("D7").Select
    ActiveCell.FormulaR1C1 = "28.33"
    Range("E7").Select
    ActiveCell.FormulaR1C1 = "14"
    Range("E8").Select
End Sub
Sub Obtener_nombre()
'
' Obtener_nombre Macro
'

'
    Range("B2").Select
    ActiveCell.FormulaR1C1 = "=LEFT(RC[-1],FIND("" "",RC[-1])-
1)"
    Range("B2").Select
    Selection.AutoFill Destination:=Range("B2:B7"),
Type:=xlFillDefault
    Range("B2:B7").Select
End Sub
Sub Obtener_apellido()
'
' Obtener_apellido Macro
'

'
```

```
    Range("C2").Select
    ActiveCell.FormulaR1C1 = "=RIGHT(RC[-2],LEN(RC[-2])-FIND(""
"",RC[-2]))"
    Range("C2").Select
    Selection.AutoFill Destination:=Range("C2:C7"),
Type:=xlFillDefault
    Range("C2:C7").Select
End Sub
Sub Calcular_importes()
'
' Calcular_importes Macro
'

'
    Range("F2").Select
    Application.CutCopyMode = False
    ActiveCell.FormulaR1C1 = "=RC[-2]*RC[-1]"
    Range("F2").Select
    Selection.AutoFill Destination:=Range("F2:F7"),
Type:=xlFillDefault
    Range("F2:F7").Select
    Selection.NumberFormat = "#,##0.00 $"
End Sub
Sub Formato_tabla()
'
' Formato_tabla Macro
'

'
    Range("A1:F7").Select
    Application.CutCopyMode = False
    ActiveSheet.ListObjects.Add(xlSrcRange, Range("$A$1:$F$7"),
, xlYes).Name = _
        "Tabla2"
    Range("Tabla2[#All]").Select
    ActiveSheet.ListObjects("Tabla2").TableStyle =
"TableStyleMedium7"
    Range("A1").Select
End Sub
```

Por ahora vamos a centrarnos en encontrar respuesta a dos preguntas:

- ¿Cómo se lee el código de las macros?
- ¿Cómo se puede editar el código de las macros?

Fijémonos en que cada macro diferente que has grabado se encuentra entre las líneas `Sub` y `End Sub`.

El código de estas macros se lee de arriba a abajo y la edición de las macros se reduce a eliminar o modificar algunas de las instrucciones en la ventana de Código.

3.4 Comentarios de las macros

Observemos el código de las macros. Las líneas que comienzan por un apóstrofe son comentarios. Por defecto los comentarios aparecen en color verde. Cuando el código de la macro se ejecuta, VBA ignora las líneas con comentarios.

Los comentarios suelen colocarse dentro de las macros para documentar el significado de ciertas líneas y aclarar conceptos que podrían dar lugar a confusión.

También se utilizan para desactivar temporalmente ciertas líneas o bloques de código que no queremos ejecutar. Esta práctica es muy común mientras se realizan pruebas o para resolver algunos problemas.

Vamos a añadir algunos comentarios a la macro `Calcular_importes`.

1. Haz que se muestre el editor de VBA, ya sea presionando **Alt+F11**, desde la ficha **Vista** o desde la ficha **Programador**.
2. Haz clic detrás de la instrucción `Range("F2").Select` y pulsa **Intro**.
3. En la nueva línea escribe lo siguiente:

   ```
   ' Multiplica el precio por hora por las horas trabajadas
   ```

4. Graba la macro haciendo clic en el diskette azul de la barra de herramientas.
5. Cierra la ventana del editor de VBA, haz clic en **Guardar como** (puedes presionar la tecla F12) y guarda el libro con el nombre "*Capítulo 1 – Introducción - modificado por mí.xlsm*".

Todos los procedimientos comienzan con la palabra clave `Sub` y terminan con `End Sub`. La palabra `Sub` va seguida del nombre de la macro y de un par de paréntesis. Entre `Sub` y `End Sub` se encuentran los procedimientos que VBA ejecuta cada vez que tú ejecutas la macro. Como he comentado antes, las declaraciones se leen de arriba a abajo ignorando las líneas con comentarios. Cuando la ejecución de las declaraciones llega a la línea `End Sub`, la macro se detiene.

3.5 Limpiar el código de las macros

A medida que vayas utilizando la grabadora de macros te darás cuenta de que se graba más código del necesario. Por ejemplo, en la macro `Insertar_encabezados`, además de grabar el estilo de fuente en negrita y cambiar su tamaño, también figuran las propiedades de tachado, superíndice, subíndice, subrayado… Cuando se utilizan cuadros de diálogo, Excel

graba todos los ajustes en la macro. En este caso el cuadro utilizado ha sido el desplegable para cambiar el tamaño de la fuente. Estas líneas innecesarias hacen que el código sea más largo, lento y difícil de entender.

Por lo tanto, al terminar de grabar tus macros, es una buena idea repasar las declaraciones que se han grabado y eliminar las líneas de código innecesarias.

Vamos a hacer un poco de limpieza de código:

1. En la ventana **Código** busca la macro `Insertar_encabezados` y localiza el siguiente bloque de código. A continuación, elimina las líneas que están tachadas.

```
With Selection.Font
        .Name = "Calibri"
        .Size = 14
        .FontStyle = "Bold"
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ThemeColor = xlThemeColorLight1
        .TintAndShade = 0
        .ThemeFont = xlThemeFontMinor
    End With
```

Después de la limpieza solo deberían quedar tres declaraciones entre las palabras `Sub` y `End Sub`. Estas declaraciones son los ajustes que modificaste al grabar la macro:

```
With Selection.Font
        .Name = "Calibri"
        .Size = 14
        .FontStyle = "Bold"
    End With
```

2. Reemplaza las dos primeras líneas de la macro `Insertar_encabezados` por la siguiente:

```
Range("A1").FormulaR1C1 = "Nombre completo"
```

3. Repite el paso 2 con cada uno de los encabezados de la tabla.

```
Range("A1").FormulaR1C1 = "Nombre completo"
Range("B1").FormulaR1C1 = "Nombre"
Range("C1").FormulaR1C1 = "Apellido"
Range("D1").FormulaR1C1 = "Importe / hora"
Range("E1").FormulaR1C1 = "Horas trabajadas"
```

```
Range("F1").FormulaR1C1 = "Total a pagar"
```

4. Realiza modificaciones similares en la macro `Datos_empleados`.
5. Presiona **Ctrl+S** en el editor de VBA para guardar los cambios.

3.6 Ejecutar una macro

Una macro se puede ejecutar desde la ventana de Excel o desde el editor de VBA.

Después de crear la macro es una buena idea probarla como mínimo una vez para ver si funciona correctamente. Más adelante en este capítulo te mostraré otras formas de ejecutar una macro, pero de momento vamos a utilizar el cuadro de diálogo **Macro**.

1. Abre el libro Capítulo 1 – Inicio.xlsm.
2. Elimina la hoja Salarios a empleados.
3. Haz clic en la ficha **Vista > Macros > Ver macros**.
4. En el cuadro de diálogo Macro, haz clic en `Insertar_nueva_hoja`.
5. Haz clic en **Ejecutar**.

   La macro `Insertar_nueva_hoja` inserta una hoja y le cambia el nombre por Salarios a empleados

Ahora vamos a ejecutar las macros restantes:

6. Haz clic en **Vista > Macros > Ver macros**.
7. Selecciona `Insertar_encabezados`.
8. Haz clic en ejecutar.
9. Ejecuta el resto de macros en el mismo orden en el que las grabaste.

Después de ejecutar todas las macros, la hoja debería tener un aspecto exactamente igual al de la hoja que eliminaste en el paso 2.

En muchas ocasiones observarás que la macro no muestra los resultados tal y como querías. O quizá después de grabar la macro has decidido cambiar de opinión sobre el tamaño de la fuente o su color.

No te preocupes, Excel hace posible la modificación de la macro sin obligarte a pasar otra vez por el engorroso proceso de grabar la macro de nuevo.

3.7 Probar y depurar una macro

Cuando se modifica una macro grabada es posible que se produzcan algunos errores. Por ejemplo, podrías eliminar una línea de código importante. Para asegurarte que, tras las modificaciones, una macro sigue funcionando igual que antes es necesario probarla de nuevo.

1. Crea un nuevo libro de Excel en blanco, manteniendo el libro original abierto.
2. Haz clic en la ficha **Programador > Visual Basic**.
3. En la ventana de código del Módulo1, pega el código de la macro de `Insertar_nueva_hoja`.
4. Haz clic en el menú **Ejecutar** – **Ejecutar Sub/UserForm**.

5. Si el nuevo libro no cuenta con una hoja llamada Hoja1 o la Hoja1 no se encuentra seleccionada en ese momento, obtendremos el error "*Subíndice fuera del intervalo*". Excel no puede encontrar la Hoja1, a la que hace referencia la macro en la primera línea. Antes de ejecutar una macro debes asegurarte de que se puede ejecutar en la hoja que está actualmente seleccionada. Cierra el cuadro de diálogo del error y a continuación selecciona Hoja1 o créala en caso de que no exista.
6. Para ver el resultado de la macro, cambia la ventana a Excel presionando **Alt+F11**.

Si has modificado la macro `Insertar_encabezados` y has eliminado por error el punto de separación de `Selection.Font` también se obtendrá un error. En este caso es "*424 – Se requiere un objeto*". Cuando veas el error presiona el botón **Depurar** y, automáticamente se mostrará la ventana de **Código**. Se activará el modo de interrupción y se resaltará en color amarillo la línea que no ha podido ejecutarse (no la que contiene el error). Al solucionar el error colocando el punto donde estaba, aparecerá otro mensaje "*Esta acción restablecerá su proyecto ¿desea continuar?*" Acepta el mensaje para cerrarlo.

3.8 Grabar y renombrar una macro

Todas las macros grabadas en este capítulo se encuentran en un libro de trabajo. Es por esto que cuando guardas el libro también se guardan las macros.

También es posible ejecutar macros ubicadas en otros libros, siempre que el libro que contiene la macro se encuentre abierto.

Desde el cuadro de diálogo **Macro** podemos seleccionar las macros que queremos ver mediante el desplegable **Macros en** de la parte inferior. Solo debes seleccionar **Todos los libros abiertos** para que aparezcan todos los nombres de las macros. Observa en la Imagen 3.8 que, si la macro se encuentra en otro libro, el nombre de la macro aparece precedido del nombre del libro donde se ubica.

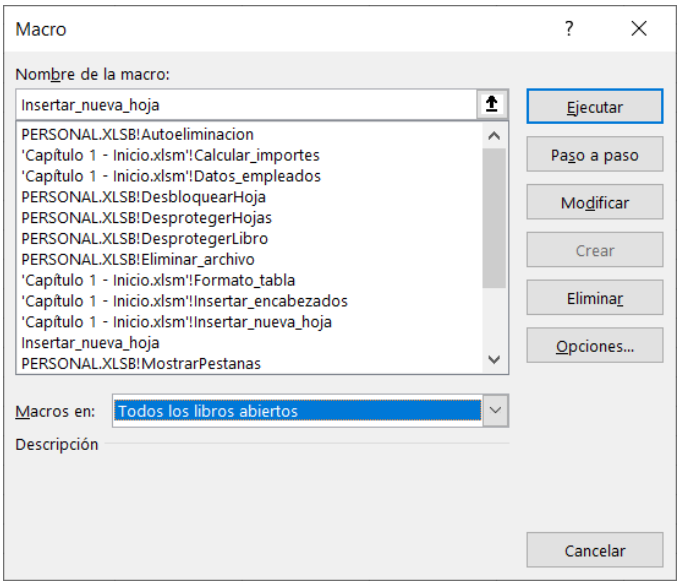


Imagen 3.8 Las macros ubicadas en otros libros se pueden ejecutar desde el libro activo.

4 Cómo mejorar las macros grabadas

Después de grabar una macro puedes darte cuenta de que necesitabas algún paso más. Agregar nuevas instrucciones a una macro no es complicado si estás familiarizado con el lenguaje VBA. Pero en muchas ocasiones puedes hacerlo de forma más rápida si utilizas la grabadora para crear ese nuevo paso y luego pegas las líneas generadas en la macro original.

Como comenté antes, la grabadora genera bastante código innecesario, pero merece la pena utilizarla porque nunca se equivoca.

Si quieres utilizar la grabadora de macros para agregar nuevas instrucciones a una macro, úsala para crear esas pequeñas acciones y, a continuación, pégalas en el lugar adecuado de la macro original.

A veces, puede ser necesario modificar el código de la macro eliminando algunas declaraciones. Antes de comenzar a hacerlo, piensa en cómo puedes utilizar la función de comentarios que hemos visto anteriormente. Puedes comentar las líneas no deseadas y ejecutar la macro con el código comentado. Si VBA no genera errores, entonces puedes eliminar las líneas con comentarios.

Una gran ventaja de utilizar la grabadora de macros es que mientras lo haces puedes aprender cuál es el código equivalente a la acción o acciones que realizas. A partir de ese comienzo puedes buscar el significado y uso de los comandos.

No te preocupes si no entiendes el código VBA. En el Capítulo 3 aprenderás todo sobre los objetos, propiedades y métodos del lenguaje VBA.

Para mejorar las macros que has creado, vamos a hacer que se muestre un mensaje cuando VBA haya terminado de ejecutar la última línea de código. Este tipo de acción no se puede grabar con la grabadora, pues Excel no cuenta con un comando que lo haga. Sin embargo, utilizando el lenguaje VBA es posible agregar nuevas instrucciones a mano. Vamos a ver cómo hacerlo:

1. En la ventana **Código** busca la macro `Formato_tabla` e inserta una línea justo antes de la instrucción `End Sub`.
2. En la nueva línea introduce la siguiente instrucción:

```
MsgBox "La tarea ha finalizado"
```

La próxima vez que ejecutes esta macro verás un cuadro de mensaje con el texto introducido. Para cerrar este mensaje debes hacer clic en el botón **Aceptar**.

`MsgBox` es una de las funciones integradas más utilizadas de VBA. La veremos más en detalle en el Capítulo 4.

5 Cómo crear una macro maestra

En este capítulo has grabado varias macros que, como has podido comprobar, requerían que las ejecutaras en el orden en que fueron grabadas. En lugar de ejecutar las macros una por

una, es más conveniente tener una macro maestra que realice todas las macros requeridas en el orden correcto. Veamos cómo hacerlo:

1. Abre el libro Capítulo 1 - Inicio.xlsm.
2. Dirígete al editor de VBA en la ventana del Explorador de proyectos.
3. Selecciona **Insertar módulo** para insertar un nuevo módulo al proyecto VBA seleccionado.
4. En la ventana **Propiedades** selecciona **Módulo2** y cambia el nombre de la propiedad `Name` por "*Procedimiento_Maestro*".
5. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub Crear_hoja_empleado()
    Insertar_nueva_hoja
    Insertar_encabezados
    Datos_empleados
    Obtener_nombre
    Obtener_apellido
    Calcular_importes
    Formato_tabla
End Sub
```

6. Presiona **Ctrl+S** para grabar los cambios.
7. Cierra la ventana del editor de VBA.
8. En la ventana de Excel, crea un libro nuevo en blanco.
9. Haz clic en **Vista > Macros > Ver macros** para mostrar el cuadro de diálogo **Macro**.
10. Selecciona la macro `Crear_hoja_empleado` y haz clic en **Ejecutar**.
    Excel ejecutará el código y al finalizar mostrará el cuadro de mensaje para indicar que ha finalizado.
11. Haz clic en **Aceptar** para cerrar el mensaje.
12. Puedes cerrar el libro sin guardarlo.

Como acabas de comprobar es muy fácil combinar pequeñas macros independientes en otras "maestras". Lo único que necesitas hacer es listar los nombres de las macros entre las líneas `Sub` y `End Sub`.

También podrías copiar todo seguido el código de las macros. Sin embargo, esto haría que el código fuese más difícil de entender. Es más conveniente trabajar con macros más pequeñas. En el capítulo 9 de este manual aprenderás varias técnicas que te permitirán probar tus macros con las herramientas que incorpora Excel.

6 Varias formas de ejecutar macros

Hasta ahora en este capítulo has aprendido un par de métodos para ejecutar macros.

Ya sabes cómo ejecutar macros desde la ventana del editor de VBA y desde el cuadro de diálogo **Macro** en la ficha **Vista**.

Mientras trabajas en la ventana del editor de VBA, es posible ejecutar una macro de las siguientes formas:

- Presionando F5 en el teclado.
- Desde el menú **Ejecutar** – **Ejecutar Sub/UserForm**.
- Haciendo clic en el botón **Ejecutar Sub/UserForm** en la barra de herramientas Estándar, como se muestra en la Imagen 6.1.

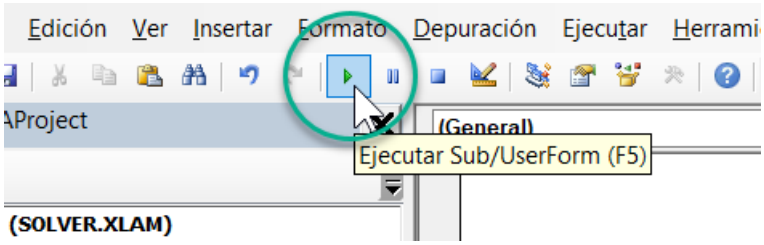


Imagen 6.1 El código VBA también se puede ejecutar desde el botón de la barra de herramientas Estándar.

En esta sección aprenderás tres métodos fabulosos para ejecutar macros que te permitirán hacerlo utilizando un atajo del teclado, un botón de la barra de herramientas o un botón en la hoja de cálculo.

6.1 Ejecutar una macro mediante un atajo de teclado

El método más conocido para ejecutar una macro es mediante un atajo de teclado.

Es mucho más rápido presionar **Ctrl+Mayús+I** que activar la macro desde el cuadro **Macro** de la ficha **Vista**.

Antes de poder utilizar un atajo de teclado debes asignárselo a la macro. Veamos cómo:

1. En la ventana de Excel presiona **Alt+F8** para mostrar el cuadro **Macro**.
2. En la lista de macros selecciona `Insertar_nueva_hoja` y luego haz clic en el botón **Opciones**.
3. Cuando aparezca el cuadro **Opciones** de la macro verás que el cursor se encuentra sobre el espacio para asignar el atajo de teclado.
4. Introduce la letra I (en mayúsculas).
5. El resultado se muestra en la Imagen 6.2.

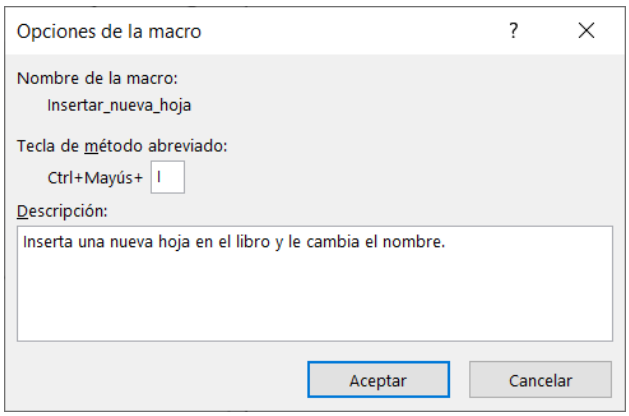


Imagen 6.2 Usando el cuadro Opciones de macro puedes asignar un atajo de teclado a una macro para ejecutarla.

6. Haz clic en **Aceptar** para cerrar el cuadro de diálogo **Opciones de macro**.
7. Haz clic en **Cancelar** para cerrar el cuadro **Macro**.

6.2 Ejecutar una macro desde la barra de herramientas de acceso rápido

Como ya sabrás es posible modificar tanto la cinta de opciones como la barra de herramientas de acceso rápido. Veamos la modificación de la barra para ejecutar una macro desde allí.

1. En la ventana de Excel haz clic en el botón **Personalizar barra de herramientas de acceso rápido** (la flecha que apunta hacia abajo en la barra de título) y elige la opción **Más comandos**, como se muestra en la Imagen 6.3.

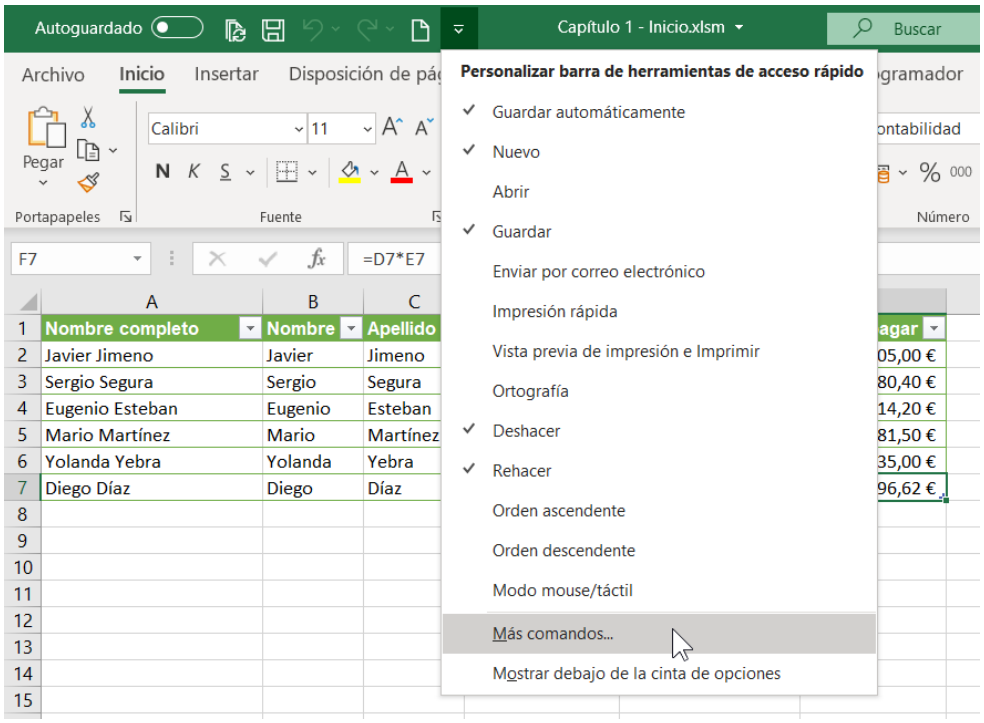


Imagen 6.3 Agregando un nuevo botón a la barra de herramientas de acceso rápido (paso 1).

Evita conflictos entre atajos de teclado

Si asignas un atajo de teclado a una macro que entra en conflicto con un atajo que ya existe en Excel (por ejemplo, Ctrl+C), Excel ejecutará la macro en vez de realizar la acción del atajo incorporado. En el caso de Ctrl+C, si lo asignamos a una macro, al pulsarlo ejecutará la macro, invalidando la utilidad de copiar.

Aparecerá el cuadro **Opciones de Excel** mostrando la **pestaña Barra de herramientas de acceso rápido**.

2. En el desplegable de la izquierda (**Comandos disponibles en**) selecciona **Macros**.
3. Haz clic en la macro `Crear_hoja_empleado`.
4. Haz clic en el botón **Agregar** para mover la macro a la lista del lado derecho, tal y como se muestra en la Imagen 6.4.

5. Para cambiar la imagen del botón para la macro, haz clic en **Modificar**.
6. Desde la galería de imágenes que se muestra selecciona cualquier botón que te guste y haz clic en **Aceptar**.

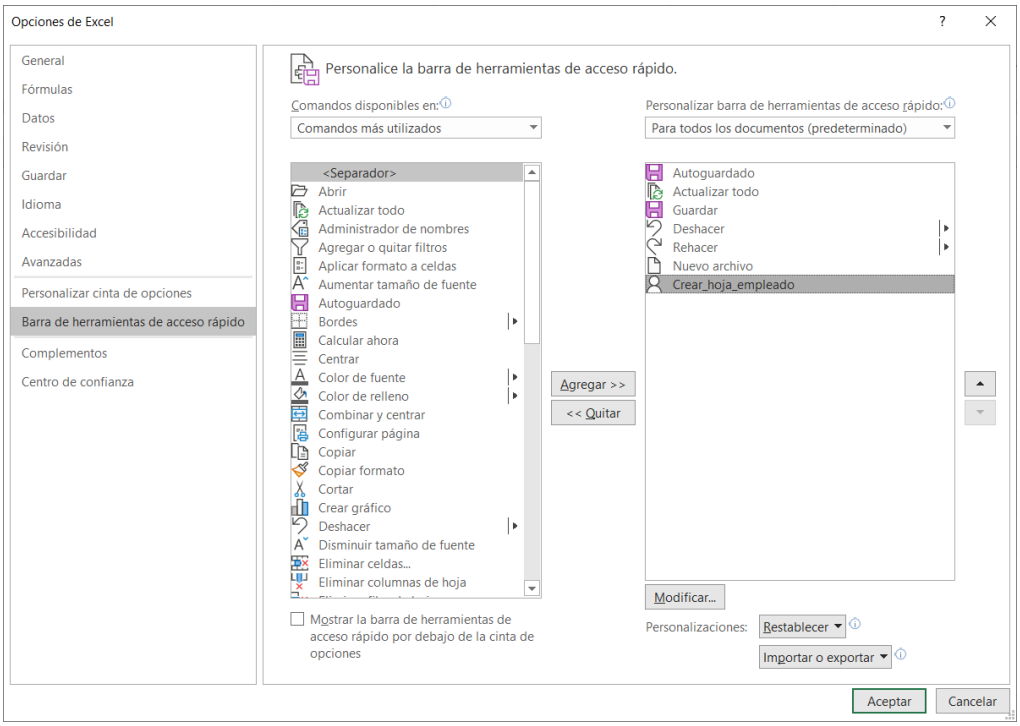


Imagen 6.4 Agregando un nuevo botón a la barra de herramientas de acceso rápido.

7. Después de cerrar la ventana de la galería de imágenes, asegúrate de que la imagen seleccionada ahora se muestra junto al nombre de la macro. Haz clic en **Aceptar** en el cuadro **Opciones de Excel** para cerrarlo. En este momento debería aparecer un nuevo botón en la barra de herramientas de acceso rápido como se muestra en la Imagen 6.5. Este botón estará disponible en cualquier libro que tengas abierto.

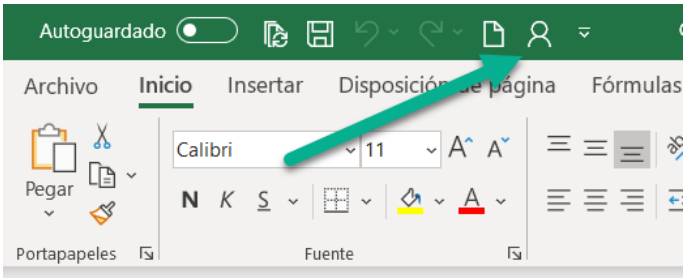


Imagen 6.5 Un botón personalizado colocado en la barra de herramientas de acceso rápido, ejecutará la macro especificada (paso 3).

8. Haz clic en el botón que acabas de agregar para ejecutar la macro asignada.

   De nuevo la macro funciona, pero tiene un problema cuando el libro desde que la ejecutas cuenta con una hoja que se llama igual que el nombre que quiere insertar la macro. Recuerda que antes de ejecutarla creaste un nuevo libro en blanco. Para ejecutar esta macro desde cualquier libro de Excel necesitas modificarla (Imagen 6.6).

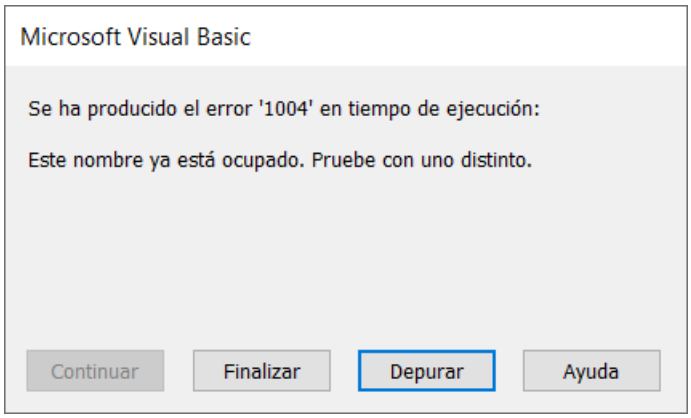


Imagen 6.6 Mensaje de error que indica que ya existe una hoja con el mismo nombre que intenta crear la macro.

9. Haz clic en el botón **Finalizar** del cuadro de diálogo del error.
10. Abre el editor de VBA y modifica la macro `Insertar_nueva_hoja` e introduce el siguiente código:

```
Sub Insertar_nueva_hoja()
'
' Insertar_nueva_hoja Macro
' Inserta una nueva hoja en el libro y le cambia el nombre.
'
Sheets.Add after:=ActiveSheet
ActiveSheet.Name = Application.InputBox("Introduce un nombre
para la hoja", "Renombrar la hoja")
End Sub
```

Para permitir al usuario darle nombre a la hoja durante la ejecución de la macro puedes utilizar el método `InputBox` que se explicará con detalle en el capítulo 4.

11. Guarda el libro y vuelve a la ventana de Excel.
12. Haz clic de nuevo en el botón de la macro de la barra de herramientas de acceso rápido.
13. Introduce el nombre que desees en el cuadro de diálogo y haz clic en **Aceptar**.

Atención: Si haces clic en **Cancelar** en lugar de escribir el nombre de la nueva hoja, VBA se encontrará con un problema y mostrará un "*error en tiempo de ejecución*" (error 1004). Haz clic en **Finalizar** para cerrar el mensaje de error y volverás a la ventana de Excel. De momento, elimina de forma manual la hoja y procura introducir un nombre cuando lo pida la macro. En el capítulo 9 te mostraré cómo manejar la mayoría de los errores para evitar que se produzcan.

Tras introducir el nombre de la hoja, VBA continúa ejecutando las macros restantes del procedimiento maestro. La ejecución vuelve a fallar cuando el programa llega al procedimiento `Formato_tabla`.

¿Qué ocurre con esta macro? Funcionaba correctamente cuando la grabaste…

A menudo surgen problemas con los nombres de los rangos cuando se utiliza la grabadora de macros. La primera línea del procedimiento le asigna el nombre Tabla2 al rango de la tabla. Como estás utilizando el procedimiento maestro en un libro que ya contiene una Tabla2, VBA muestra un error. Los nombres de las tablas en cada libro deben ser únicos. Para que este procedimiento funcione debemos revisarlo.

14. Haz clic en el botón **Depurar** y Excel resaltará la línea de código que no puede ejecutar.
15. Sal del modo de pausa seleccionando **Ejecutar** – **Restablecer**.
16. Modifica el procedimiento `Formato_tabla` de la siguiente manera:

```
Sub Formato_tabla()
'
' Formato_tabla revisada
'

Dim strNombreTabla As String
    strNombreTabla = InputBox("Introduce un nombre para _
      la tabla", "Nuevo nombre de tabla")
ActiveSheet.ListObjects.Add _
        (xlSrcRange, Range("$A$1:$F$7"), , xlYes).Name =
strNombreTabla
    ActiveSheet.ListObjects(strNombreTabla).TableStyle =
"TableStyleMedium7"
    Range("A1").Select
    MsgBox "La tarea ha finalizado"
End Sub
```

La primera línea del código declara la variable `strNombreTabla` para guardar el nombre de la tabla que se introduce mediante la función `InputBox`. En el Capítulo 3 aprenderemos todo sobre las variables y sus tipos, cómo declararlas y cómo asignarlas.

La siguiente línea crea un nuevo objeto y le asigna el nombre almacenado en la variable `strNombreTabla`. Cada vez que se ejecuta el procedimiento se solicita un nombre para la tabla. Debes introducir siempre un nombre único. En caso contrario se mostrará un error.

Fíjate en el carácter de subrayado que hay detrás de la función `ListObjects.Add`. Es la forma que tenemos de decirle a VBA que la línea continua debajo. También aprenderemos esto en el Capítulo 3.

Después de agregar la tabla y asignarle un nombre, la macro hace referencia de nuevo a la variable `strNombreTabla` para asignarle un estilo de formato predefinido.

A continuación, el procedimiento selecciona la celda A1 y para finalizar se muestra el mensaje al usuario.

17. Después de hacer los cambios en el procedimiento, guarda el código y vuelve a la ventana de Excel.
18. Vuelve a ejecutar el procedimiento haciendo clic en el botón que creaste en la barra de herramientas de acceso rápido.
19. El procedimiento `Crear_hoja_empleado` debería ejecutarse correctamente.

6.3 Ejecutar una macro desde un botón de la hoja

A veces tiene más sentido colocar el botón que ejecuta la macro en la propia hoja. De esta forma no pasará desapercibido (como puede ocurrir con el método anterior).

Veamos cómo hacerlo:

1. Abre el archivo Capítulo 1 – Materiales.xlsm.
2. En caso de que aparezca un mensaje para activar el contenido, actívalo.
3. Haz clic en **Programador > Insertar**. Aparecerá la barra **Controles de formulario** como se muestra en la Imagen 6.7.

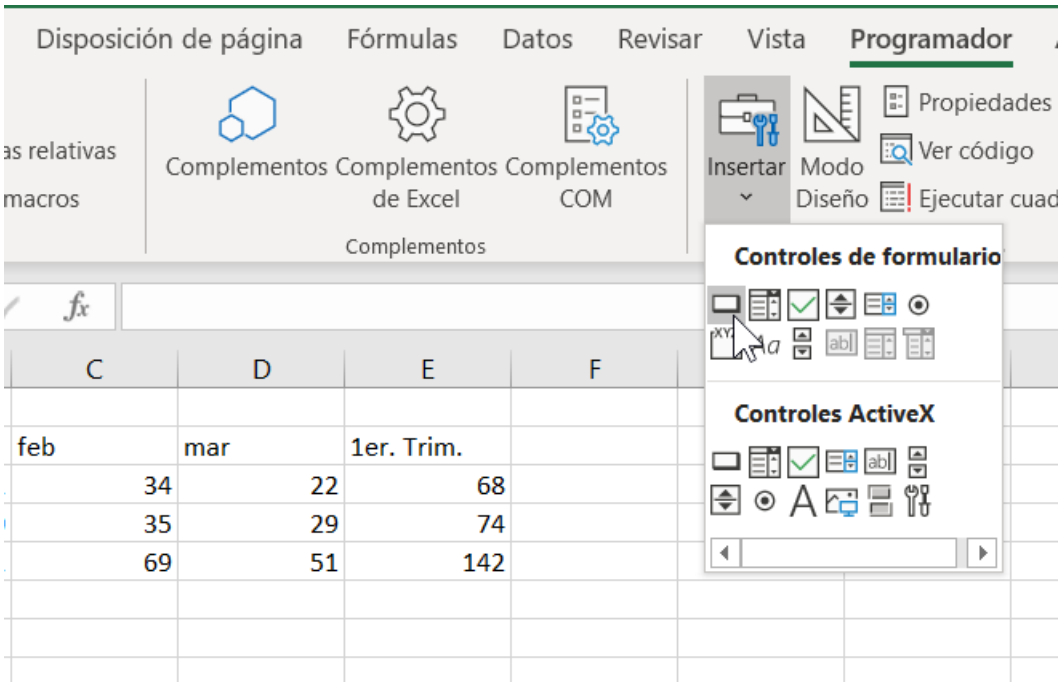


Imagen 6.7 Agregando un botón a la hoja.

4. Haz clic en el primer botón como se muestra en la Imagen 6.7.
5. Haz clic en cualquier parte vacía de la hoja. Cuando se muestra el cuadro **Asignar macro**, selecciona `Macro_material` y haz clic en **Aceptar**.
6. Excel creará un botón con la etiqueta Botón 1. Para cambiar la etiqueta haz clic dentro del botón y escribe el texto “Formato de celdas”. Si el texto no encaja, no te preocupes. Puedes cambiar el tamaño con los selectores de los extremos.
7. Ahora el botón debería parecerse al de la Imagen 6.8.

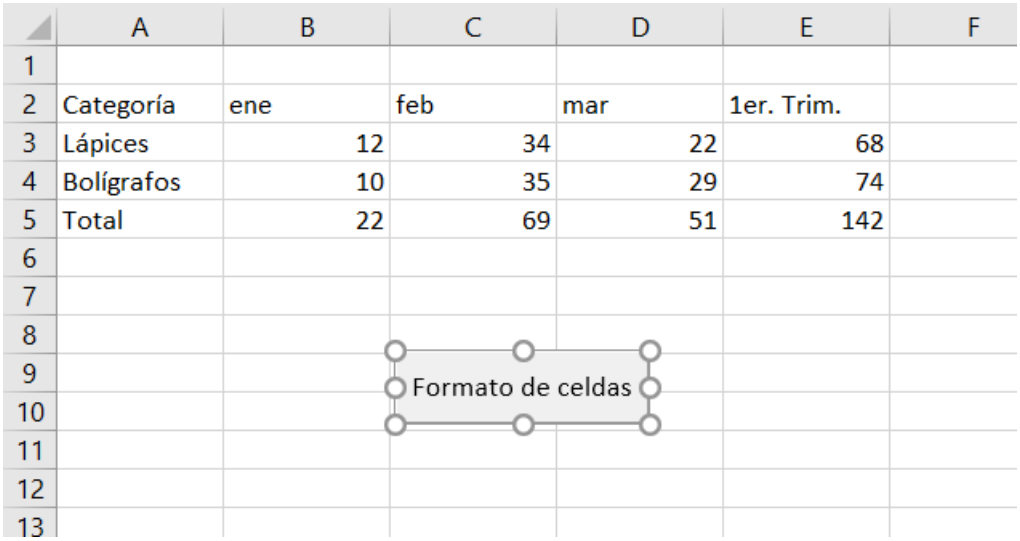


Imagen 6.8 Botón con macro asociada.

8. Cuando termines de cambiar el nombre al botón haz clic en otra parte de la hoja para salir del modo edición.

   Atención: Si haces clic en el botón izquierdo sin querer, ya no podrás hacer nada para evitar que la macro se ejecute. Puedes cambiar el tamaño del botón después de que se haya ejecutado.

9. Para ejecutar la macro haz clic en el botón. En un instante la hoja tiene el formato deseado (ver Imagen 6.9).

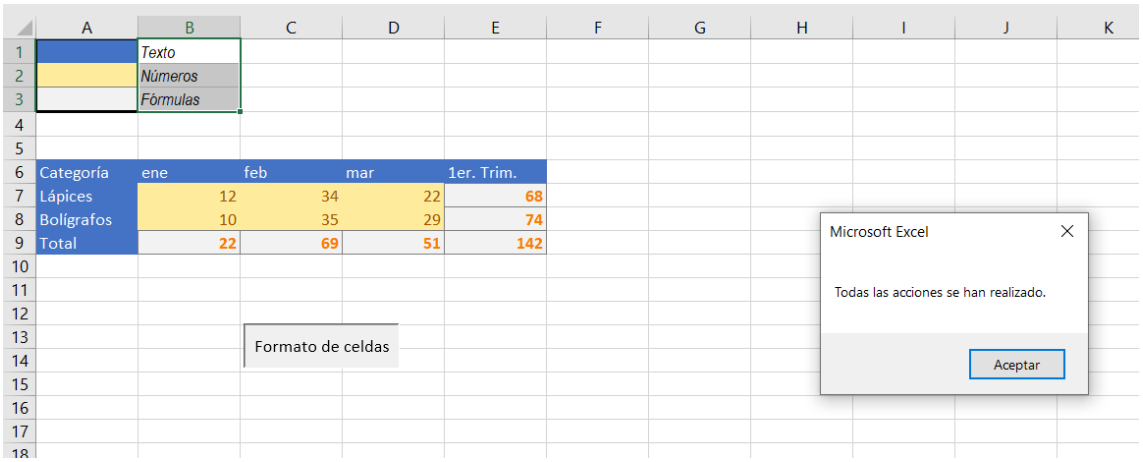


Imagen 6.9 Se ha cambiado el formato de la hoja desde un botón.

Ahora vamos a eliminar este formato ejecutando el procedimiento `Eliminar_formatos`.

10. Presiona **Alt+F8** para abrir el cuadro de diálogo **Macro**. Selecciona la macro `Eliminar_formatos` y haz clic en **Ejecutar**.
11. Siguiendo los pasos anteriores, crea un botón nuevo. Llámalo “Eliminar formatos” y le asignas la macro.
12. Guarda el libro con un nombre diferente para que puedas utilizarlo de nuevo cuando quieras volver a crear un botón.

7 Resumen

En este primer capítulo has aprendido a crear macros con la grabadora de macros, registrando tus acciones por la hoja. También has aprendido a ver, leer y modificar las macros grabadas desde la ventana del editor de VBA. Además, has tenido oportunidad de probar varios métodos de ejecución de macros.

También se han explicado los problemas de seguridad que debes tener en cuenta a la hora de abrir un libro que contenga macros.

En el siguiente capítulo nos centraremos en el uso de la ventana del editor de VBA.

Capítulo 2
El editor de VBA

Ahora que ya sabemos cómo grabar, ejecutar y editar macros, vamos a pasar a la ventana del editor de VBA (también conocido como VBE) para familiarizarnos con sus características. Como podrás intuir, desde el editor de VBA podrás:

- Escribir tus propios procedimientos de VBA.
- Crear formularios personalizados.
- Ver y modificar las propiedades de los objetos.
- Probar procedimientos VBA y localizar errores.

Podemos acceder al editor de VBA de las siguientes formas:

- Desde la ficha **Programador > Código > Visual Basic**.
- Desde la ficha **Programador > Controles > Ver código**.
- Presionado la combinación **Alt+F11**.

1 El Explorador de proyectos

La ventana del **Explorador de proyectos** muestra una lista jerárquica de los proyectos actualmente abiertos y sus elementos. Un proyecto VBA puede contener los siguientes elementos:

- Hojas.
- Gráficos.
- El propio libro.
- Módulos (los verdaderos almacenes de código).
- Clases (módulos especiales que permiten crear objetos).
- Formularios.
- Referencias a otros proyectos.

Desde el **Explorador de proyectos** podemos gestionar proyectos y movernos fácilmente entre los proyectos que se carguen en la memoria. Podemos activarlo de una de estas tres formas:

- Desde el menú **Ver**, seleccionando **Explorador de proyectos**.
- Desde el teclado, presionando **Ctrl+R**.
- Desde la barra de herramientas **Estándar**, presionando el botón del **Explorador de proyectos**, como se muestra en la **Imagen 1.1**.

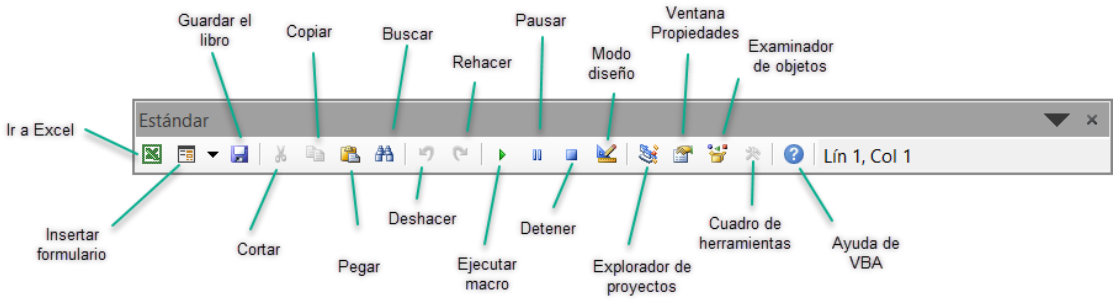


Imagen 1.1 Los botones de la barra de herramientas Estándar proporcionan una forma rápida de acceder a muchas de las características del editor de VBA.

La ventana **Explorador de proyectos** contiene tres botones, como se muestra en la **Imagen 1.2**.

El botón de la izquierda (**Ver código**) muestra la ventana de código del módulo seleccionado. El botón del centro (**Ver objeto**) muestra la hoja seleccionada en la carpeta de objetos o un formulario ubicado en la carpeta de formularios. El botón de la derecha (**Alternar carpetas**) activa o desactiva la visualización de las carpetas en el **Explorador de proyectos**.

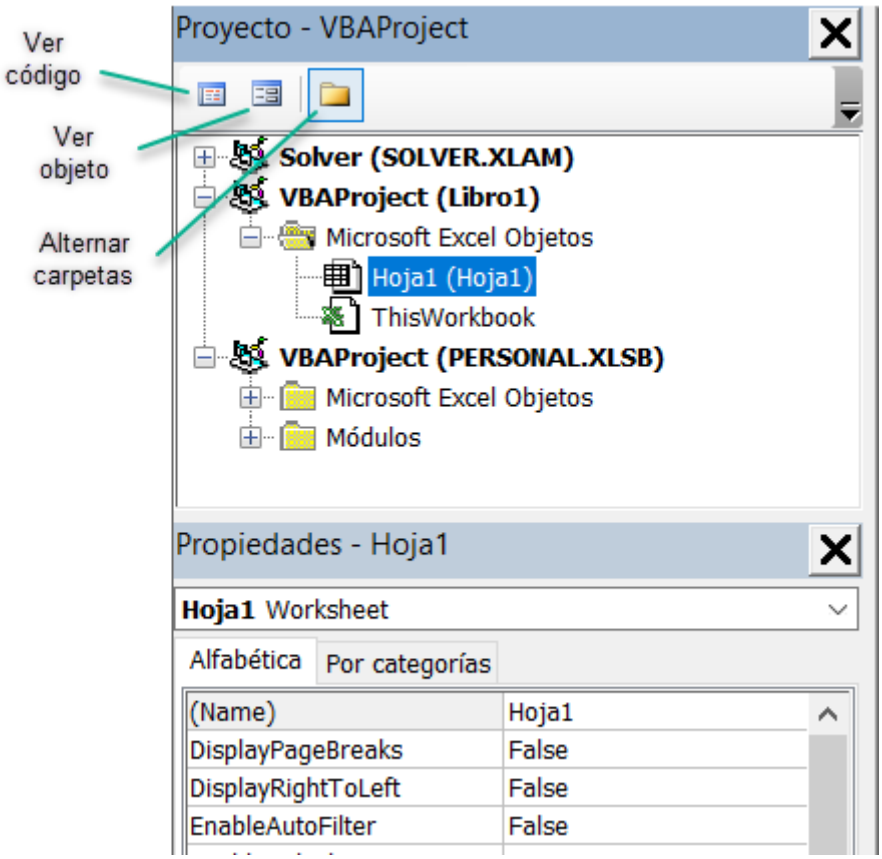


Imagen 1.2 El Explorador de proyectos muestra una lista de los proyectos abiertos en ese momento. La ventana Propiedades muestra la configuración del objeto seleccionado en el Explorador de proyectos.

2 La ventana Propiedades

La ventana **Propiedades** permite revisar y establecer las propiedades de los objetos del proyecto. El nombre seleccionado en el **Explorador de proyectos** se visualizará en el cuadro de objetos situado justo debajo de la barra de título de la ventana **Propiedades**. Por ejemplo, la **Imagen 1.2** muestra las propiedades del objeto `Hoja1`. Las propiedades del objeto pueden visualizarse en orden alfabético o por categoría, haciendo clic en la pestaña correspondiente.

- **Alfabética**: Muestra de forma alfabética todas las propiedades del objeto seleccionado. Podemos cambiar la configuración de la propiedad seleccionando el nombre de la propiedad y escribiendo o seleccionando el nuevo ajuste.
- **Por categorías**: Enumera todas las propiedades del objeto seleccionado por categorías. Podemos reducir la lista para ver las categorías o expandirla para ver las propiedades. El signo más (+) a la izquierda del icono del nombre de la categoría indica que la lista de categorías puede ampliarse. El signo menos (-) indica que la categoría puede reducirse.

Se puede acceder a la ventana **Propiedades** de tres formas:

- Desde el menú Ver, seleccionando Ventana **Propiedades**.
- Desde el teclado presionando **F4**.
- Desde la barra de herramientas **Estándar**, pulsando el botón **Ventana Propiedades**.

3 La ventana Código

La ventana **Código** se utiliza para la programación VBA, así como para ver y modificar el código de las macros grabadas y los procedimientos VBA ya existentes. Cada uno de los módulos puede abrirse en una ventana de código separada. Hay varias formas de activar la ventana **Código**:

- Desde el **Explorador de proyectos**, seleccionando el módulo o formulario que deseemos y haciendo clic en el botón para ver el código.
- Desde la barra de menús, selecciona **Ver – Código**.
- Con el teclado, pulsando **F7**.

En la Imagen 3.1 observamos que, en la parte superior de la ventana **Código** hay dos desplegables que permitirán movernos rápidamente dentro del código VBA. En el desplegable de la izquierda podemos seleccionar el objeto cuyo código queremos ver. El de la derecha nos permite elegir rápidamente un procedimiento o evento. Cuando abrimos esto, los nombres de todos los procedimientos se muestran ordenados alfabéticamente. Si seleccionamos un procedimiento en el desplegable **Declaraciones**, el foco saltará a la primera línea de este procedimiento.

Arrastrando la barra de división que se muestra en la **Imagen 3.1** hasta una posición seleccionada, podemos dividir la ventana **Código** en dos paneles. Entonces podemos ver diferentes secciones de un procedimiento largo o un procedimiento diferente. Esto se utiliza a menudo para copiar o cortar secciones de código entre las dos partes divididas.

Para volver a la pantalla con una sola ventana, simplemente arrastramos la barra de división hasta la parte superior de la ventana **Código**.

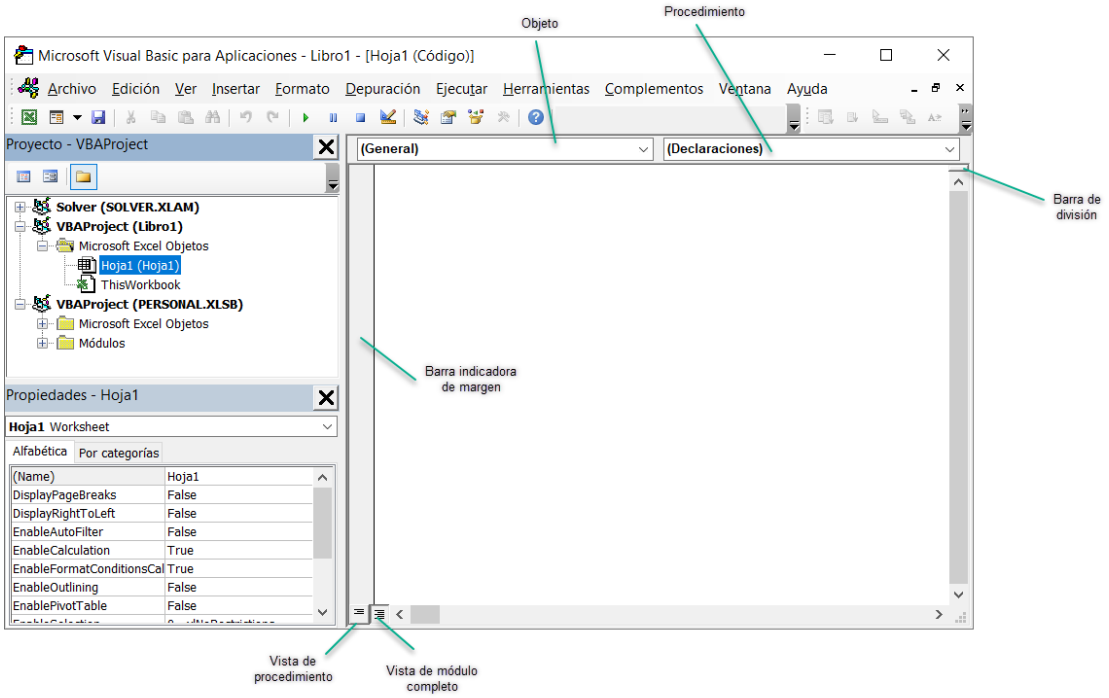


Imagen 3.1 La ventana Código tiene varios elementos que facilitan la localización de los procedimientos y la revisión del código VBA.

En la parte inferior izquierda de la ventana encontramos dos botones. El botón **Ver procedimiento** muestra un solo procedimiento en la ventana. El botón **Ver módulo completo** muestra todos los procedimientos del módulo seleccionado. Podemos utilizar la barra de desplazamiento vertical para desplazarnos a través del código del módulo.

La barra indicadora de margen se utiliza para mostrar indicadores útiles durante la edición y depuración de las macros. Para echar un vistazo rápido a algunos de estos indicadores, ojea el **Capítulo 9**.

4 Configuración de las opciones del editor de VBA

Existen otras ventanas que también se utilizan frecuentemente en el entorno de VBA.

La **Imagen 5.1** muestra la lista de ventanas que se pueden acoplar en la ventana del editor de VBA. Aprenderemos a utilizar alguna de estas ventanas en el **Capítulo 3** (Examinador de objetos y ventana Inmediato) y en el **Capítulo 9** (Ventana Locales y ventana Inspección).

5 Ayuda en materia de sintaxis y programación

La **Imagen 5.2** muestra la barra de herramientas **Edición,** que contiene varios botones. Éstos nos permiten introducir instrucciones VBA correctamente formateadas de forma fácil y rápida. Si la barra **Edición** no se muestra en el editor, podemos activarla desde el menú **Ver – Barras de herramientas – Editar**.

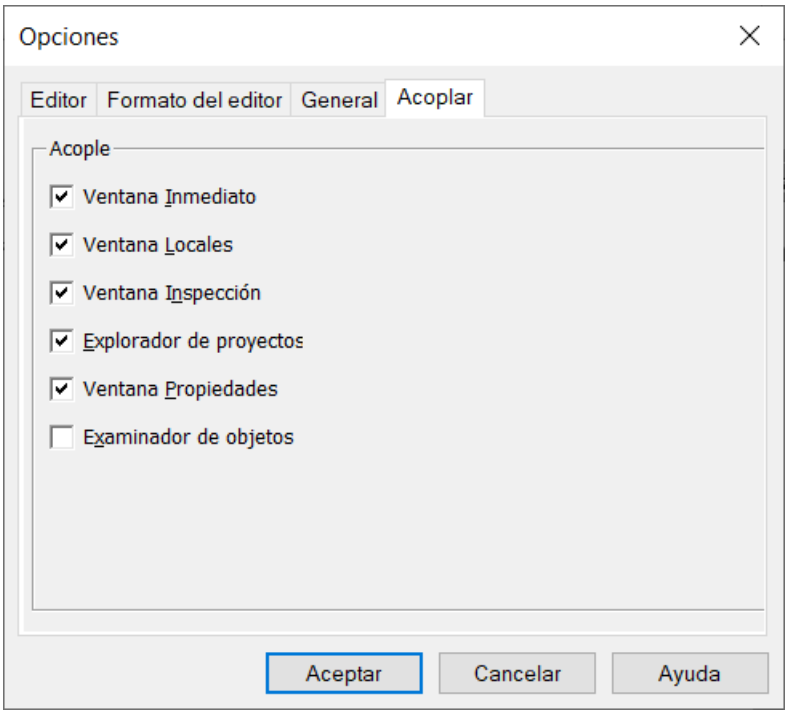


Imagen 5.1 La pestaña Acoplar del cuadro Opciones permite elegir las ventanas que deseas acoplar en la pantalla del editor de VBA.

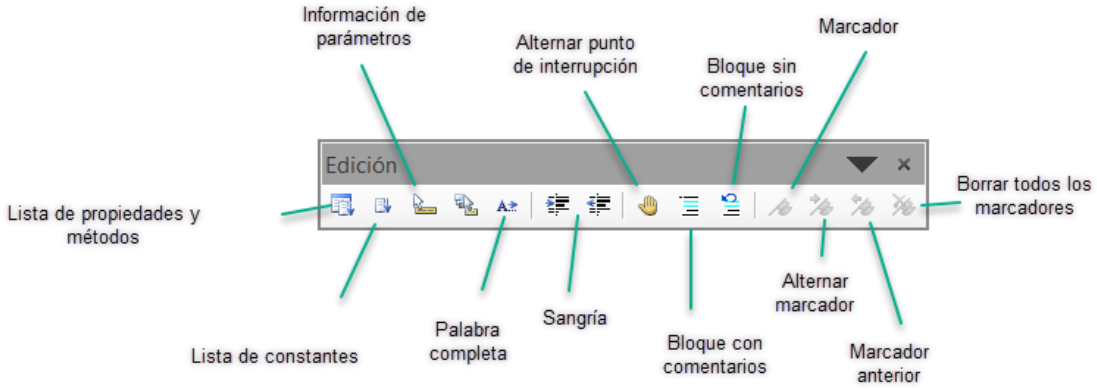


Imagen 5.2 Los botones de la barra de herramientas Edición facilitan la escritura y el formato de las instrucciones de VBA.

Escribir procedimientos en VBA requiere utilizar instrucciones y funciones integradas. Como la mayoría de los usuarios no pueden memorizar las sintaxis de todas las instrucciones disponibles en VBA, la **tecnología IntelliSense** nos proporciona asistencia en la sintaxis y programación mientras introducimos las instrucciones. Podemos hacer que aparezcan ventanas especiales que nos guíen en el proceso de creación del código VBA.

5.1 Lista de propiedades y métodos

Cada objeto puede contener varias propiedades y métodos. Cuando se introduce el nombre del objeto y un punto que separa el nombre de su propiedad y método, puede aparecer un menú emergente. Este menú muestra las propiedades y métodos disponibles para el objeto al que precede, como se muestra en la **Imagen 5.3**. Para activar esta función automatizada, seleccionamos **Herramientas – Opciones**. En el cuadro **Opciones** hacemos clic en la pestaña

Editor. Debemos asegurarnos de que la casilla de verificación **Lista de miembros automática** está seleccionada.

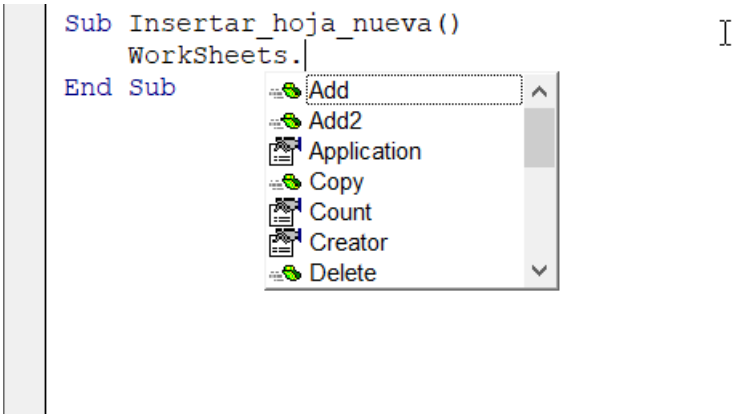


Imagen 5.3 Mientras se introducen las instrucciones, VBA sugiere las propiedades y los métodos que se pueden utilizar con el objeto.

Para elegir un elemento del menú emergente que aparece, debemos comenzar a escribir el nombre de la propiedad o el método que queremos seleccionar. Cuando Excel resalta el nombre, pulsamos **Intro** para insertar la palabra en el código y comenzar una línea nueva. Si queremos seguir escribiendo instrucciones debemos utilizar el tabulador. También podemos hacer doble clic en el elemento para insertarlo en el código. Para cerrar el menú emergente sin insertar ninguna propiedad o método, simplemente presionamos la tecla **Esc**. Al hacerlo, VBA no lo volverá a mostrar en el mismo objeto. Para volver a mostrarlo podemos:

- Presionar **Ctrl+J**.
- Utilizar la tecla de retroceso para borrar el punto separador. A continuación, volvemos a escribirlo.
- Hacer clic con el botón derecho del ratón en la ventana **Código** y seleccionar **Lista de propiedades y métodos** del menú contextual.
- Seleccionar **Editar – Lista de propiedades y métodos**.
- Hacer clic en el botón **Lista de propiedades y métodos** de la barra de herramientas **Edición**.

5.2 Lista de constantes

Una constante es un valor que indica un estado o resultado específico. Excel tiene muchas constantes predefinidas. Aprenderemos acerca de las constantes y sus tipos en el **Capítulo 3**.

Supón que deseamos que una macro active la vista previa de salto de página de la hoja. En una ventana de Excel hay cuatro tipos de vistas:

- **Normal**. Es la vista predefinida para la mayoría de las tareas de Excel.
- **Diseño de página**. Esta vista nos permite ver el documento tal y como aparecerá en la página impresa.
- **Vista previa de salto de página**. Nos permite ver dónde se cortarán las páginas cuando imprimamos el documento.

- **Vistas personalizadas**. Nos permiten guardar el conjunto de ajustes de visualización e impresión como una vista personalizada.

Las tres primeras opciones de vista están representadas por una constante. Los nombres de las constantes siempre comienzan con los caracteres ***"xl"***. En cuanto introducimos en la ventana **Código** la instrucción:

```
ActiveWindow.View =
```

aparecerá un menú emergente con los nombres de las constantes válidas para esta propiedad, como se muestra en la **Imagen 5.4**.

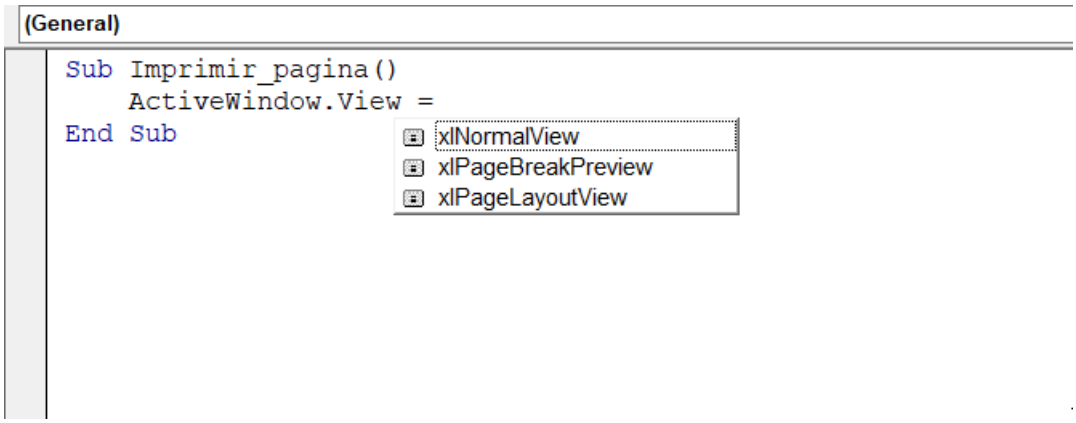


Imagen 5.4 El menú emergente muestra la lista de constantes válidas para la propiedad introducida.

Para introducir una constante desde el menú emergente, utilizaremos las mismas técnicas que para las propiedades de la sección anterior.

Este menú puede activarse haciendo clic en **Ctrl+Mayús+J** o desde el botón **Lista de constantes** de la barra de herramientas **Estándar**.

5.3 Información del parámetro

Si anteriormente hemos trabajado con las funciones de hoja de Excel, ya sabremos que muchas de ellas requieren uno o más argumentos (o parámetros). Por ejemplo, esta es la sintaxis de la función más utilizada:

SUMA(número1;[número2];...)

Donde **número1;número2…** son de 1 a 255 argumentos que se pueden sumar.

Al igual que las funciones de hoja, los métodos VBA pueden requerir uno o más argumentos. Si un método requiere un argumento, se pueden ver los nombres de los argumentos obligatorios y opcionales en el cuadro de información que aparece justo debajo del cursor en cuanto escribimos el paréntesis de inicio, como se muestra en la **Imagen 5.5**. En el cuadro de información, el argumento que se debe introducir es el que se encuentra en negrita. Cuando se introduce el primer argumento y se introduce la coma, VBA muestra el siguiente argumento en negrita.

Como en las sintaxis de las funciones de hoja, los corchetes **[]** indican que el argumento es opcional.

Podemos mostrar la información de parámetros con el teclado, introduciendo el método o la función seguido del paréntesis izquierdo, y pulsar **Ctrl+Mayús+I**. También es posible hacer clic en el botón **Información de parámetros** de la barra de herramientas **Edición** o seleccionar el menú **Editar – Información de parámetros**.

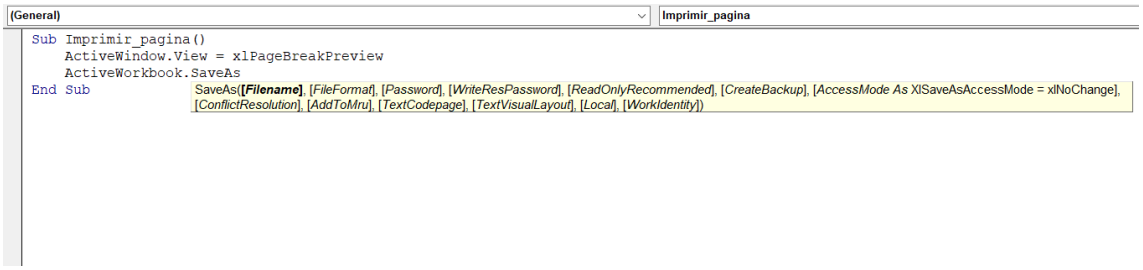


Imagen 5.5 Una lista de argumentos utilizados por un método VBA.

La información de parámetros facilita la introducción de argumentos a un método VBA. Además, nos recuerda otras dos cosas importantes para que el método funcione correctamente: el orden de los argumentos y el tipo de datos necesarios de cada uno. Aprenderemos sobre los tipos de datos en el **Capítulo 3**.

5.4 Información rápida

Cuando seleccionamos una instrucción, función, método, nombre de procedimiento o constante en la ventana **Código** y luego hacemos clic en el botón **Información rápida** de la barra de herramientas **Edición** (o presionamos **Ctrl+I**), VBA muestra la sintaxis del elemento resaltado, así como el valor de una constante, como se muestra en la **Imagen 5.6**. La función de información rápida puede activarse o desactivarse mediante el cuadro de diálogo **Opciones**. Para usar la función, hacemos clic en la pestaña **Editor** y seleccionamos la opción **Información rápida automática**.

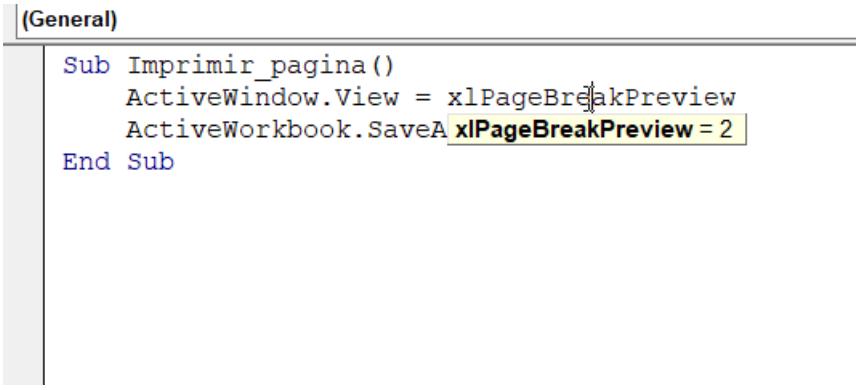


Imagen 5.6 La herramienta Información rápida muestra una lista de argumentos requeridos por un método o función, un valor de una constante o el tipo de objeto o propiedad seleccionado.

5.5 Palabra completa

Otra forma de aumentar la velocidad al escribir procedimientos VBA en la ventana **Código** es con la herramienta **Palabra completa**. Al introducir las primeras letras de una palabra clave,

debemos presionar **Ctrl+Barra espaciadora** o hacer clic en el botón **Palabra completa** en la barra de herramientas **Edición**. VBA rellenará las letras restantes completando la introducción de la palabra clave. Por ejemplo, cuando escribimos cuatro letras de la palabra `Application` (Appl) en la ventana de código y pulsamos **Ctrl+Barra espaciadora**, VBA completará el resto de la palabra y en lugar de "*Appl*" se visualizará la palabra completa.

5.6 Sangrías

Si la opción **Sangría automática** está activada, podemos sangrar automáticamente las líneas de código con el número de caracteres especificados en el cuadro **Herramientas – Opciones**. El valor predeterminado de la sangría es de cuatro caracteres. Es posible cambiar esto fácilmente desde el mismo cuadro de diálogo (ver **Imagen 5.1**, pestaña **Editor**).

¿Por qué usar sangrías en el código? Una sangría en algunas líneas de código hace el procedimiento más legible. La sangría se recomienda especialmente para introducir líneas de código para tomar decisiones o repetir acciones. Aprenderemos a crear este tipo de instrucciones de VBA en los **Capítulos 5** y **6**.

Aprenderemos ahora a aplicar las características de sangrado a las líneas de código de la macro `Macro_material` con la que practicamos en el **Capítulo 1**:

1. Abre el libro **Capítulo 1 – Materiales.xlsm** con el que estuviste trabajando en el capítulo anterior.
2. Presiona **Alt+F11** para cambiar a la ventana del editor de VBA.
3. Selecciona **Ver – Barras de herramientas – Edición** para mostrar la barra de herramientas. Si la barra aparece en el medio de la pantalla, haz doble clic en su barra de título para anclarla en la parte superior del editor de VBA.
4. En el **Explorador de proyectos**, selecciona el proyecto correspondiente al archivo y activa el **Módulo 1**, que contiene el código de la macro `Macro_material`.
5. Selecciona el bloque de código situado entre las palabras clave `With` y `End With`.
6. Haz clic en el botón **Sangría derecha** o pulsa la tecla **Tab**. El bloque de instrucciones seleccionado se moverá cuatro espacios hacia la derecha (en caso de que estés utilizando el ajuste predeterminado del cuadro de diálogo **Opciones**).
7. Haz clic en el botón **Sangría izquierda** en la barra **Edición** o pulsa **Mayús+Tab** para desplazar las líneas de código seleccionadas a la ubicación anterior.
8. Cierra el libro sin guardar los cambios.

5.7 Bloques de comentarios

En el **Capítulo 1** aprendimos que introducir una comilla simple o apóstrofe al principio de una línea la convertía en un comentario. Los comentarios no solo hacen el código más fácil de interpretar, sino que también son útiles para probarlo y solucionar posibles problemas.

Por ejemplo, cuando ejecutamos una macro puede que el resultado no sea el que esperamos. En vez de borrar las líneas que podrían contener el error, colocaremos un apostrofe al principio de la línea para convertirla en un comentario.

- Para comentar una o varias líneas de código, simplemente las seleccionamos y hacemos clic en el botón **Bloque con comentarios** en la barra de herramientas **Edición**.

- Cuando deseemos convertir esas líneas de nuevo en instrucciones ejecutables por VBA, las seleccionaremos de nuevo y, a continuación, hacemos clic en el botón **Bloque sin comentarios**.

Si no seleccionamos ningún texto antes de hacer clic en el botón **Bloque con comentarios**, el apóstrofe se colocará en la línea en la que se encuentre el cursor en ese momento.

6 Cómo utilizar el Examinador de objetos

Es posible desplazarnos fácilmente a través de la enorme cantidad de elementos y características de VBA gracias al **Examinador de objetos**. Para acceder a él utiliza cualquiera de los siguientes métodos en el editor de VBA:

- Presiona **F2**.
- Desde el menú **Ver – Examinador de objetos**.
- Desde el botón **Examinador de objetos** de la barra de herramientas **Estándar**.

El **Examinador de objetos** nos permite navegar por los objetos disponibles, por los procedimientos VBA, así como ver sus propiedades, métodos y eventos. Con la ayuda del **Examinador de objetos** podemos movernos rápidamente entre los procedimientos de los proyectos y buscar objetos y métodos desde las bibliotecas de tipos de objetos.

La ventana del **Examinador de objetos** está dividida en tres secciones, como se muestra en la **Imagen 6.1**. La parte superior de la ventana muestra una lista desplegable con los nombres de todas las bibliotecas y proyectos disponibles para el proyecto VBA con el que estamos trabajando.

Una biblioteca es un archivo especial que contiene información sobre los objetos en una aplicación. Se pueden añadir nuevas bibliotecas desde el cuadro de diálogo **Referencias** (menú **Herramientas**). La opción **<Todas>** muestra los objetos de todas las bibliotecas que tengamos instaladas en el equipo. Al seleccionar la biblioteca **Excel**, veremos sólo los nombres de los objetos exclusivos de Microsoft Excel. A diferencia de la biblioteca de Excel, la biblioteca de VBA contiene los nombres de todos los objetos de VBA.

Debajo del cuadro desplegable veremos un cuadro de texto de búsqueda con el que encontrar rápidamente la información en la biblioteca especificada. Para buscar solo palabras completas, podemos hacer clic con el botón derecho del ratón en cualquier lugar de la ventana del **Examinador de objetos** y seleccionar **Buscar palabra completa** del menú contextual.

La sección de resultados de búsqueda del **Examinador de objetos** muestra la biblioteca, la clase y los elementos miembros que cumplen los criterios introducidos en el cuadro de búsqueda, como se muestra en la **Imagen 6.2**.

Al escribir el texto de búsqueda y hacer clic en el botón de búsqueda (los prismáticos), VBA expande el cuadro del **Examinador de objetos** para mostrar los resultados. Podemos ocultar o mostrar los resultados de la búsqueda haciendo clic en el botón situado a la derecha del botón de búsqueda.

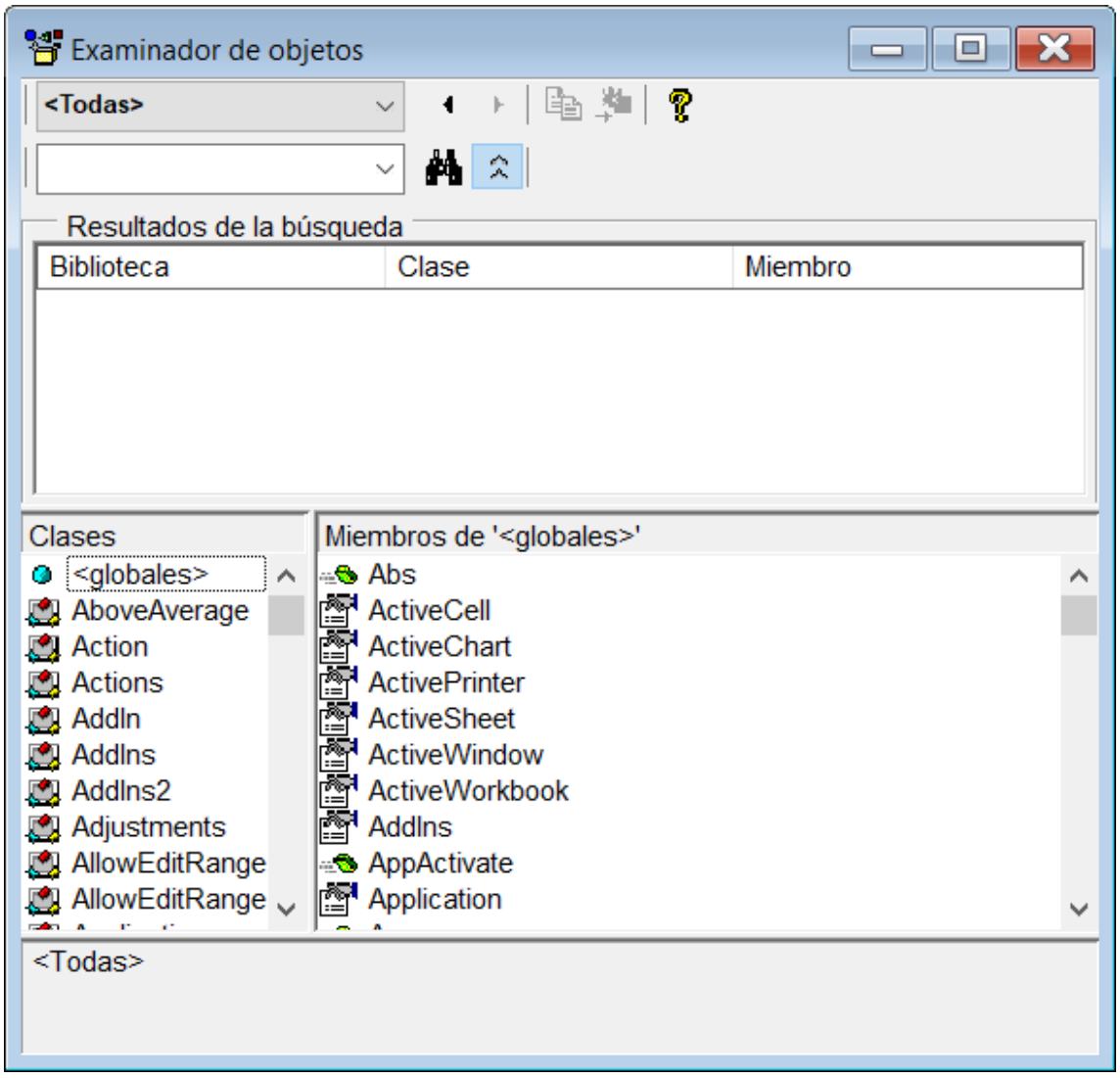


Imagen 6.1 La ventana Examinador de objetos te permite navegas por todos los objetos, propiedades y métodos disponibles en el proyecto actual de VBA.

El cuadro de lista **Clases** muestra las clases de objetos existentes en la biblioteca. Si seleccionamos un proyecto VBA, esta lista muestra los objetos del proyecto. En la **Imagen 6.2** se selecciona la clase del objeto `Interaction`. Cuando se selecciona una clase de objeto, la lista de la derecha (**Miembros**) muestra las propiedades, métodos y eventos disponibles para esa clase. Por defecto, los miembros se muestran en orden alfabético. Sin embargo, podemos organizarlos por tipo de grupo (propiedades, métodos o eventos) mediante el comando **Miembros del grupo** del menú contextual del **Explorador de objetos**.

Al seleccionar un proyecto VBA en el cuadro de lista de arriba, el cuadro de lista **Miembros** mostrará todos los procedimientos disponibles en este proyecto. Para examinar el código de un procedimiento simplemente hacemos doble clic en su nombre. Si seleccionamos la biblioteca VBA, se mostrará un listado de funciones y constantes de VBA. En caso de necesitar más información sobre la clase seleccionada o uno de sus miembros, haremos clic en el botón de interrogación en la parte superior de la ventana del **Explorador de objetos**.

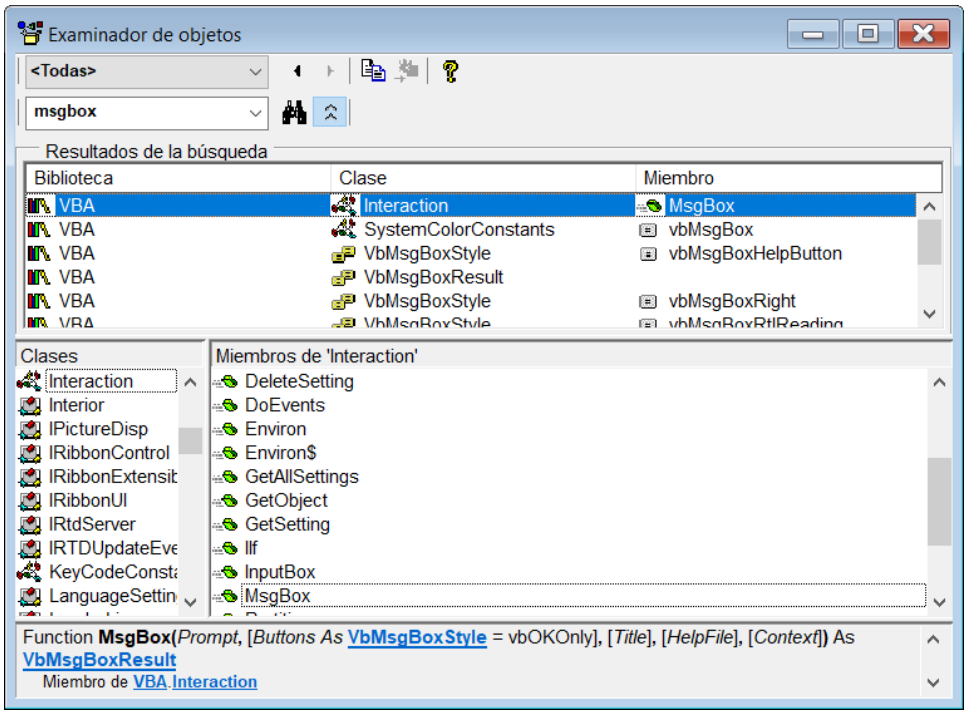


Imagen 6.2 Buscando respuestas desde el Examinador de objetos.

La parte inferior de la ventana del **Explorador de objetos** muestra un área con la definición del miembro seleccionado. Al hacer clic en el texto del hipervínculo, podemos saltar rápidamente a la clase del miembro seleccionado en la ventana del **Explorador de objetos**. El texto que se muestra en el área inferior puede ser copiado al portapapeles de Windows para pegarlo en una ventana **Código**. Si la ventana **Código** está visible mientras el **Explorador de proyectos** está abierto, podemos ahorrar tiempo arrastrando la plantilla de código resaltada y soltándola en la ventana **Código**.

Es posible ajustar fácilmente el tamaño de las diferentes secciones del **Explorador de objetos** arrastrando las líneas horizontales y verticales que las dividen.

Ahora que hemos descubierto el **Explorador de objetos**, te estarás preguntando cómo podrías utilizarlo para programar con VBA. Imagina un cuadro de texto en el centro de una hoja del libro. ¿Cómo hacer que Excel mueva el cuadro de forma que se coloque en la esquina superior izquierda de la hoja? La siguiente tarea responde a esa pregunta:

1. Crea un libro nuevo.
2. Haz clic en **Insertar > Texto > Cuadro de texto**.
3. Ahora dibuja el cuadro en el centro de la hoja e introduce cualquier texto dentro, como se muestra en la **Imagen 6.3**.
4. Haz clic en cualquier celda fuera del área del cuadro de texto.
5. Presiona **Alt+F11** para mostrar el editor de VBA.
6. Haz clic en **Insertar – Módulo** para añadir una nueva hoja de módulo.
7. En la ventana **Propiedades**, introduce un nombre para el módulo: `Movimiento`.
8. Haz clic en el menú **Ver – Examinador de objetos** o presiona **F2**.
9. En el cuadro desplegable, selecciona la opción **Excel**.

10. Introduce la palabra “textbox” como texto de búsqueda como se muestra en **Imagen 6.4** y luego haz clic en el botón **Buscar**.

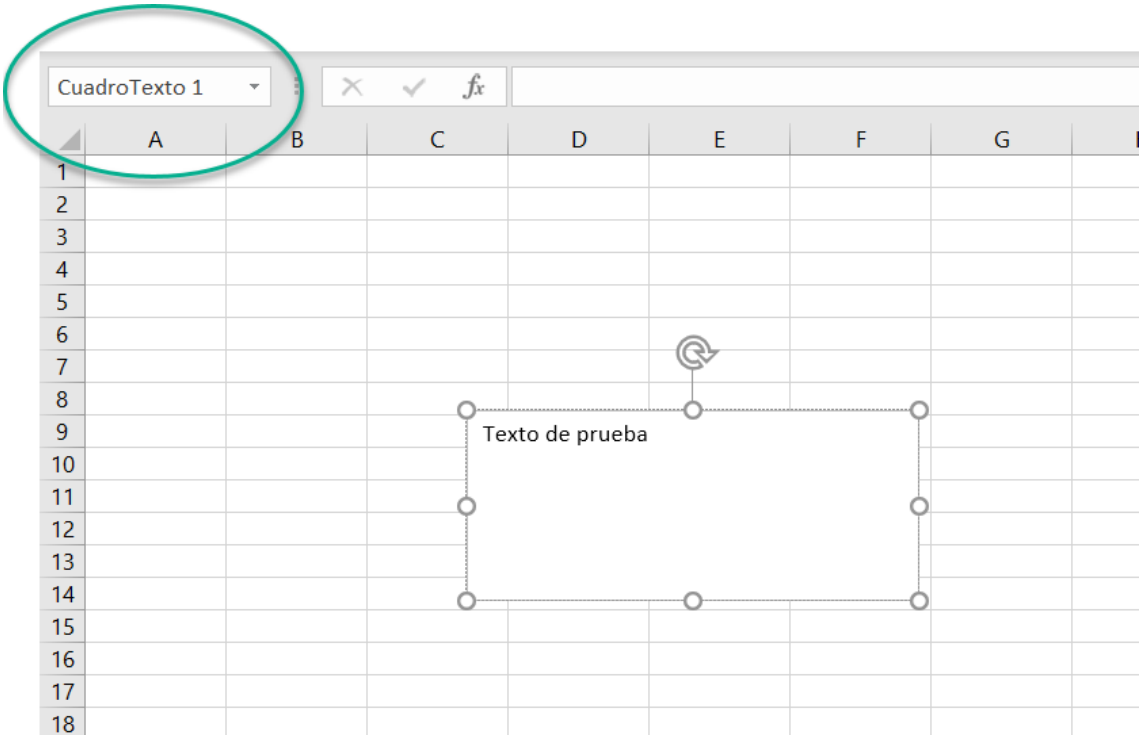


Imagen 6.3 Excel muestra el nombre del objeto insertado en el cuadro de nombres.

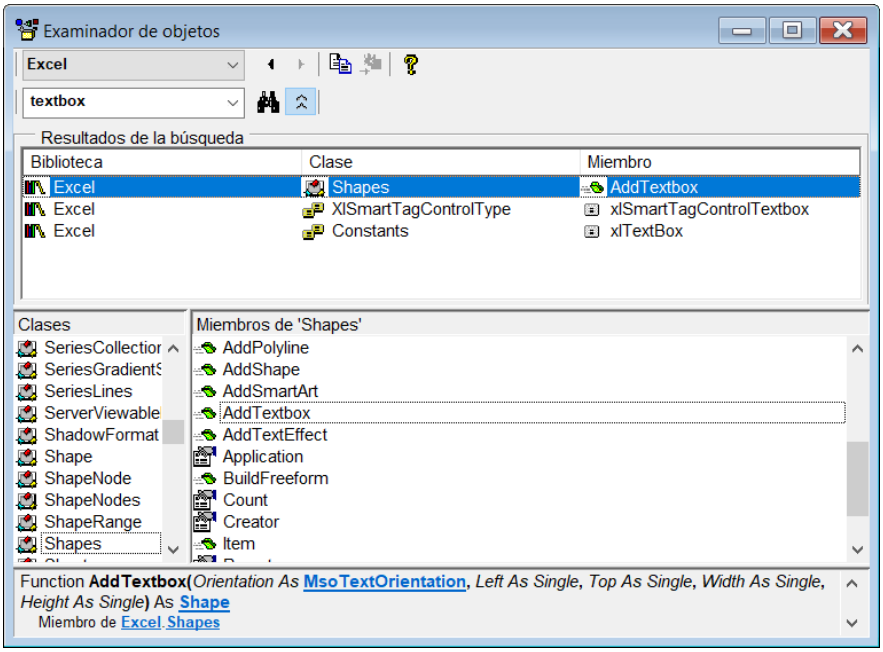


Imagen 6.4 Desde el Examinador de objetos encontramos las instrucciones VBA apropiadas para escribir cualquier procedimiento.

VBA busca en la biblioteca de Excel y muestra los resultados de la búsqueda. Observamos que aparece el objeto `Shapes` (ver **Imagen 6.4**). Examinando los miembros de `Shapes`, vemos que el método `AddTextBox` se utiliza para añadir un nuevo cuadro de texto a una hoja. En la parte inferior del **Examinador de objetos** se ve

la sintaxis correcta para utilizar este método. Si seleccionamos el método `AddTextBox` y presionamos **F1**, podremos ver la ventana de ayuda con más detalles sobre cómo utilizar este método. La **Ayuda** nos dice que las propiedades `Left` y `Top` determinan la posición del cuadro de texto.

11. Cierra la ventana del **Examinador de objetos** y la ventana de **Ayuda** si se encuentran abiertas.
12. Haz doble clic en el módulo `Movimiento` e introduce el procedimiento `Mover_TextBox` como se muestra a continuación.

```
Sub Mover_TextBox()
    With ActiveSheet.Shapes("CuadroTexto 1")
        .Select
        .Left = 0
        .Top = 0
    End With
End Sub
```

El procedimiento `Mover_TextBox` selecciona el **CuadroTexto 1** en la colección `Shapes`.

CuadroTexto 1 es el nombre por defecto colocado en la hoja. Cuando añades un nuevo objeto a la hoja, Excel le asigna un número (índice). En lugar de utilizar el nombre del objeto puedes referirte al miembro de una colección por su índice. Por ejemplo, en lugar de:

```
With ActiveSheet.Shapes("CuadroTexto 1")
```

Introduce:

```
With ActiveSheet.Shapes(1)
```

13. Haz clic en el menú **Ejecutar – Ejecutar Sub/UserForm** para ejecutar este procedimiento.
14. Presiona **Alt+F11** para pasar a la ventana de Excel. El cuadro de texto se ha colocado en la esquina superior izquierda.
15. Guarda el archivo como **Capítulo 2 – Cuadro de texto.xlsm**, pero mantenlo abierto, ya que continuarás trabajando con él.

Vamos a manipular otro objeto con VBA.

1. Dibuja un círculo en la misma hoja donde insertaste el cuadro de texto anterior. Hazlo desde la ficha **Insertar > Ilustraciones > Formas**. Mantén presionada la tecla **Mayús** mientras lo dibujas para crear un círculo perfecto.
2. Haz clic fuera del círculo para deseleccionarlo.
3. Presiona **Alt+F11** para activar el editor de VBA.
4. En la ventana Código del módulo `Movimiento`, escribe un procedimiento VBA que coloque el círculo dentro del cuadro de texto anterior. Ten en cuenta que Excel enumera los objetos de forma consecutiva. Al primer objeto se le asigna el número 1,

al segundo el número 2 y así sucesivamente. No importa el tipo de objeto, ya sea un cuadro de texto, un círculo o un rectángulo.

5. El procedimiento `Mover_circulo` muestra cómo mover un círculo a la esquina superior izquierda de la hoja activa:

```
Sub Mover_circulo()
    With ActiveSheet.Shapes(2)
        .Select
        .Left = 0
        .Top = 0
    End With
End Sub
```

Mover el círculo es igual que mover el cuadro de texto o cualquier otro objeto que se coloque en la hoja. Observa que en lugar de referirse al círculo por su nombre (Elipse 2), el procedimiento utiliza el índice del objeto.

6. Ejecuta el procedimiento `Mover_circulo`.
7. Presiona **Alt+F11** para volver a la ventana de Excel.
8. El círculo debería aparecer ahora en la parte superior del cuadro de texto.

6.1 Localizar procedimientos con el Examinador de objetos

Además de localizar objetos, propiedades y métodos, el **Examinador de objetos** es una herramienta útil para localizar y acceder a procedimientos escritos en varios proyectos de VBA. El siguiente ejercicio muestra cómo se puede ver de un vistazo qué procedimientos están almacenados en el proyecto seleccionado.

1. En el **Examinador de objetos**, selecciona **VBAProject** de la lista desplegable como se muestra en la Imagen 7.1. El lado izquierdo del **Examinador de objetos** muestra los nombres de los objetos que están incluidos en el proyecto seleccionado. El cuadro **Miembros** de la derecha muestra los nombres de todos los procedimientos.
2. En la lista **Miembros** haz doble clic en el procedimiento `Mover_circulo`.
3. Excel localiza el procedimiento seleccionado en la ventana **Código**.

7 La biblioteca de objetos de VBA

En los ejemplos anteriores utilizaste las propiedades de los objetos que son miembros de la colección `Shapes` en la biblioteca de objetos de Excel. Esta biblioteca contiene objetos específicos para el uso de Excel.

Aparte de esta biblioteca, Microsoft Excel también cuenta con la biblioteca de objetos de VBA, con los que podrás gestionar archivos, establecer la fecha y la hora, interactuar con los usuarios, convertir unos tipos de datos en otros, manipular cadenas de texto o realizar cálculos matemáticos. En el siguiente ejercicio usarás una de las funciones VBA incorporadas para crear una nueva subcarpeta de Windows sin salir de Excel.

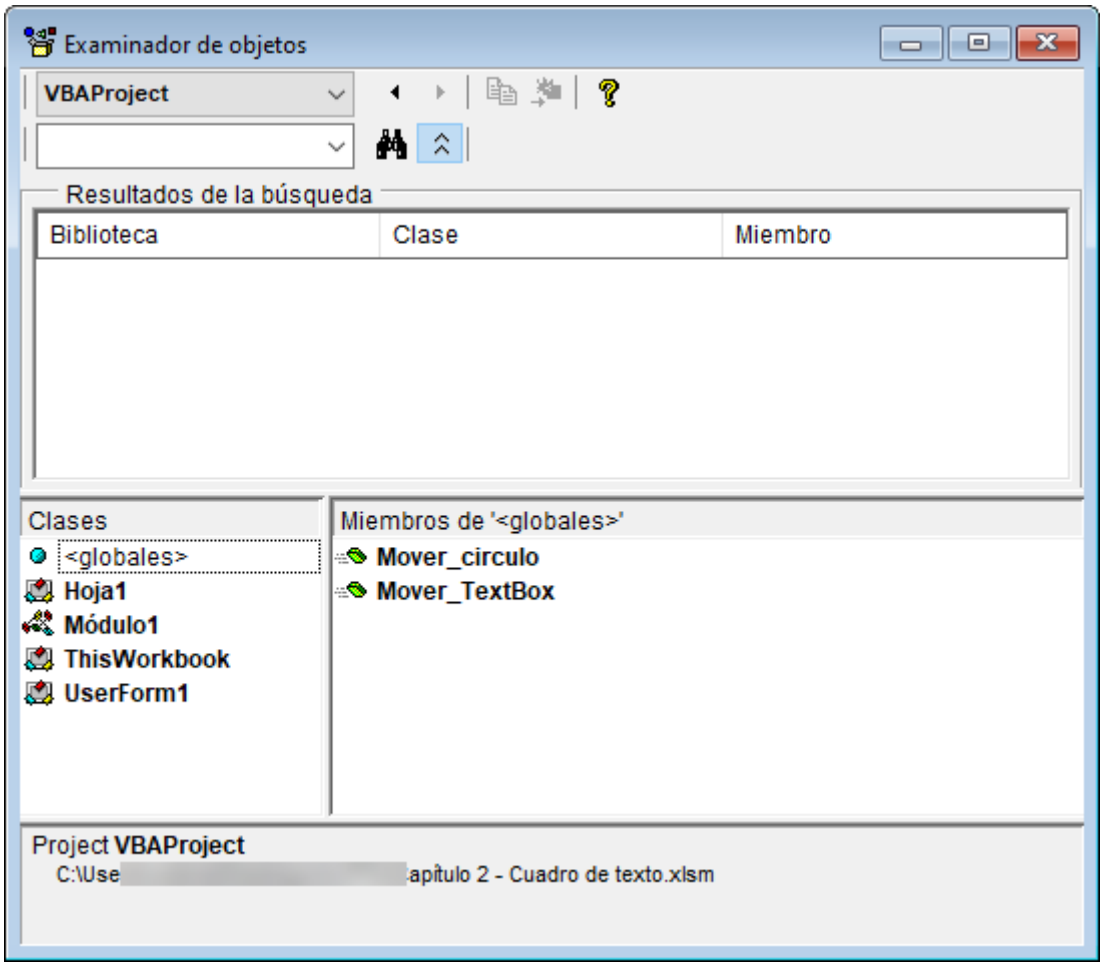


Imagen 7.1 El Examinador de objetos enumera todos los procedimientos disponibles en un proyecto VBA

1. Presiona **Alt+F11** para volver al módulo `Movimiento` donde introdujiste los dos últimos procedimientos.
2. En una nueva línea escribe el nombre del nuevo procedimientos: `Sub Carpeta_nueva()`
3. Presiona **Intro**. VBA introducirá la palabra clave `End Sub.`
4. Presiona **F2** para activar el **Examinador de objetos**.
5. Haz clic en la flecha desplegable de la biblioteca y selecciona VBA.
6. Introduce “file” como texto de búsqueda en el cuadro y pulsa el botón **Buscar**.
7. Desplázate hacia abajo en el cuadro **Miembros** y busca el método `MkDir`, como se muestra en la Imagen 7.2.
8. Haz clic en el botón **Copiar** (el botón del centro en la parte superior del **Examinador de Objetos**) para copiar el nombre del método seleccionado en el portapapeles de Windows.
9. Vuelve a la ventana de código del módulo `Movimiento` y pega la instrucción copiada dentro del procedimiento `Carpeta_nueva`.
10. Introduce un espacio, seguido de “C:\Excel”. Asegúrate de introducir el nombre de la ruta entre comillas. El procedimiento `Carpeta_nueva` debería verse así:

```
Sub Carpeta_nueva()
    MkDir "C:\Excel"
```

```
End Sub
```

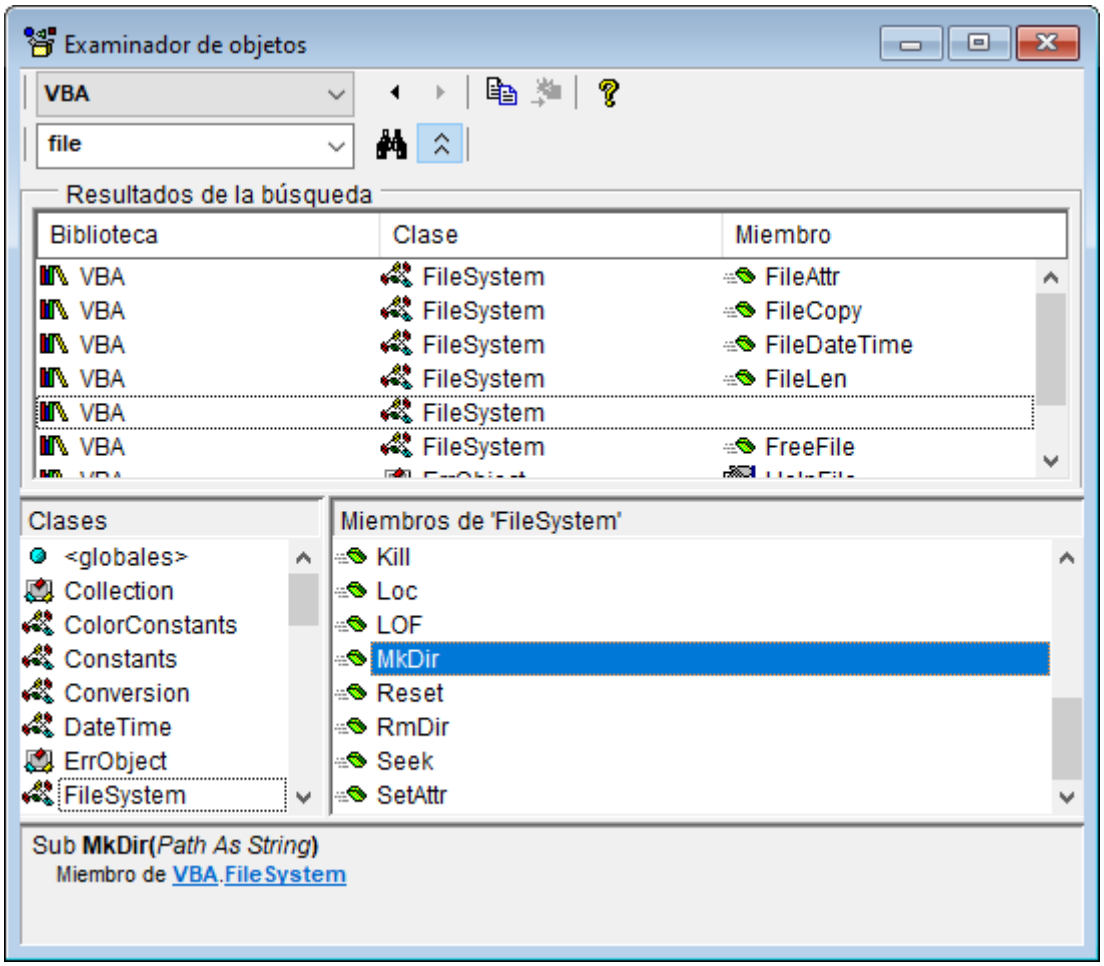


Imagen 7.2 Cuando escribas procedimientos desde cero, consulta el Examinador de objetos para conocer los nombres de las funciones VBA incorporadas.

11. Posiciona el cursor dentro del código del procedimiento y selecciona **Ejecutar – Ejecutar Sub/UserForm** para ejecutar el procedimiento `Carpeta_nueva`.
    Cuando lo ejecutas, VBA crea una nueva carpeta en la unidad C. Para ver dicha carpeta, accede desde el **Explorador de Windows**.
    Después de crear esta carpeta, es posible que te des cuenta de que no la necesitas. Aunque se podría eliminar la carpeta fácilmente desde el Explorador de Windows, vamos a deshacernos de ella a través de VBA. El **Examinador de Objetos** muestra muchos otros métodos útiles para trabajar con carpetas y archivos. El método `RmDir` es tan simple de usar como `MkDir`.
12. Para eliminar la carpeta Excel del disco duro podrías reemplazar el método `MkDir` por `RmDir`, y luego volver a ejecutar el procedimiento `Carpeta_nueva`. Pero en vez de esto vamos a escribir uno nuevo llamado `Eliminar_carpeta` desde la ventana **Código** del módulo `Movimiento`, tal y como se muestra:

```
Sub Eliminar_carpeta()
    RmDir "C:\Excel"
```

```
End Sub
```

El método `RmDir` permite eliminar las carpetas no deseadas del disco duro.

13. Coloca el cursor dentro del código del procedimiento `Eliminar_carpeta` y selecciona **Ejecutar – Ejecutar Sub/UserForm** para ejecutarlo.
14. Comprueba desde el **Explorador de Windows** si la carpeta Excel se ha eliminado del equipo.

8 La ventana Inmediato

La ventana **Inmediato** se utiliza para probar instrucciones, funciones y otros elementos de VBA antes de usarlos en los procedimientos. Es una gran herramienta para experimentar con el lenguaje.

La ventana **Inmediato** te permite escribir declaraciones VBA y probar su resultado inmediatamente sin tener que escribir un procedimiento. Se puede comparar a un bloc de notas. Úsalo para probar declaraciones. Si la declaración funciona con el resultado que esperas, puedes copiarla (o arrastrarla) hasta la ventana **Código**.

La ventana **Inmediato** puede ser movida a cualquier lugar del editor de VBA o se puede acoplar para que aparezca siempre en la misma zona de la pantalla. La configuración del acoplamiento se puede activar y desactivar en la pestaña **Acoplar** del cuadro de diálogo **Opciones** (**Herramientas – Opciones**).

- Para acceder rápidamente a la ventana **Inmediato** simplemente presiona **Ctrl+G** mientras te encuentres en el editor de VBA.
- Para cerrar la ventana **Inmediato** pulsa el botón **Cerrar** en la parte superior derecha de la ventana.

Antes de empezar a crear procedimientos VBA completos (que lo harás en el siguiente capítulo), comienza con algunos ejercicios de calentamiento para familiarizarte con el vocabulario de VBA.

La mejor forma de aprender es introducir una instrucción simple de VBA. Excel la comprobará y mostrará el resultado en la siguiente línea.

Vamos a configurar la pantalla y a continuación, unas líneas sencillas de código explicado.

1. En el editor de VBA haz clic en el menú **Ver – Ventana Inmediato**.
2. Organiza la pantalla de forma que se muestren tanto la ventana de Excel como el editor de VBA. En caso de que tengas dos monitores, puedes utilizar uno para cada ventana.
3. En el editor de VBA, presiona **Ctrl+G** para activar la ventana **Inmediato**.
4. Introduce en ella la siguiente instrucción y pulsa **Intro**:

```
Worksheets.Add
```

Cuando presionas **Intro**, VBA se pone a trabajar. Si has introducido correctamente la anterior declaración, se añadirá una nueva hoja de trabajo. Debería aparecer la pestaña de la Hoja2 resaltada en la parte inferior del libro.

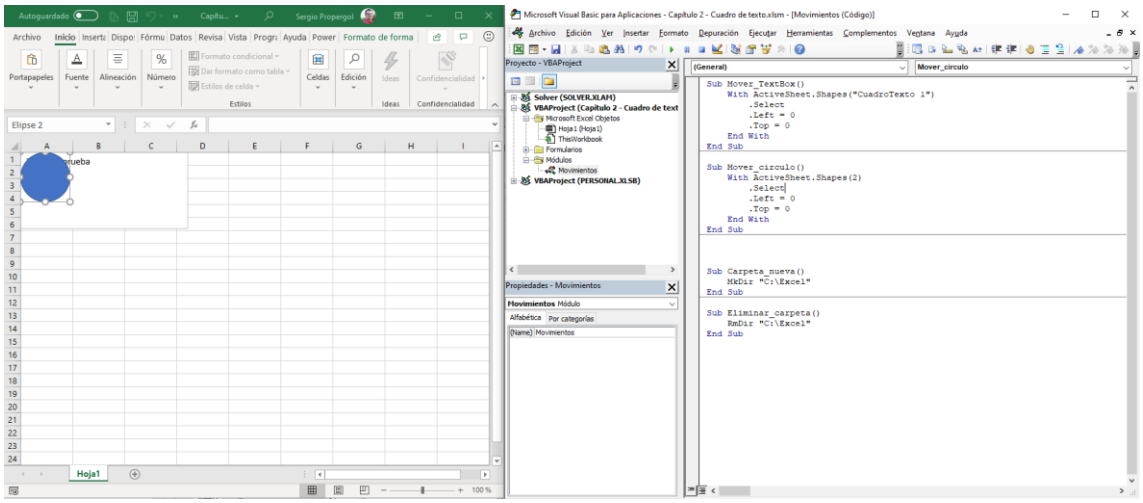


Imagen 8.1 Posicionando las ventanas una al lado de la otra podrás observar la ejecución de las instrucciones introducidas en la ventana Inmediato.

5. En la ventana **Inmediato**, escribe esta otra declaración VBA y pulsa **Intro** cuando hayas finalizado:

   ```
   Range("A1:A4").Select
   ```

   Cuando presionas **Intro**, Excel selecciona las celdas A1, A2, A3 y A4 de la hoja activa.

6. Ahora introduce la siguiente instrucción:

   ```
   [A1:A4].Value = 55
   ```

   Al presionar **Intro**, VBA coloca el número 55 en todas las celdas del rango especificado. Esta declaración es una forma abreviada de referirse al objeto `Range`. La sintaxis completa es más legible:

   ```
   Range("A1:A4").Value = 55
   ```

7. Introduce la siguiente instrucción en la ventana **Inmediato**:

   ```
   Selection.ClearContents
   ```

Al presionar **Intro**, VBA borra los resultados de la declaración anterior en las celdas seleccionadas. Las celdas A1:A4 están ahora vacías.

8. Introduce una última instrucción:

   ```
   ActiveCell.Select
   ```

   Cuando presionas **Intro**, VBA hace que la celda A1 se active.

La Imagen 8.2 muestra todas las instrucciones introducidas en la ventana **Inmediato** de este ejercicio. Cada vez que se pulsó la tecla **Intro**, Excel ejecutó la instrucción de la línea donde se

encontraba el cursor. Si quieres volver a ejecutar la misma instrucción, haz clic en cualquier lugar de la línea que contiene la instrucción y presiona **Intro**.

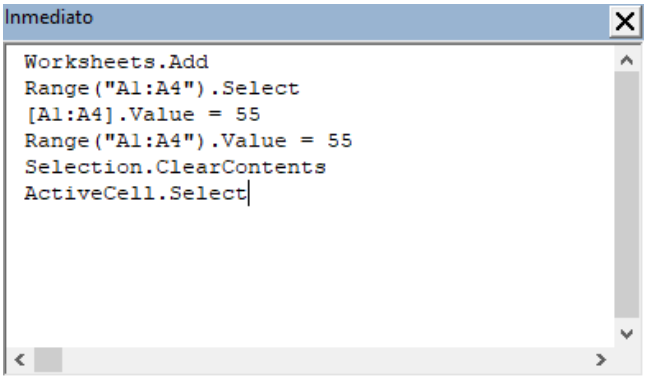


Imagen 8.2 Las instrucciones introducidas en la ventana Inmediato se ejecutan en cuanto se presiona la tecla Intro.

8.1 Cómo obtener información de la ventana Inmediato

Hasta ahora has utilizado la ventana **Inmediato** para realizar acciones. Estas acciones podrían haberse hecho manualmente haciendo clic con el ratón en varias celdas de la hoja e introduciendo datos.

La ventana **Inmediato** también puede utilizarse para responder preguntas. Supongamos que quieres saber qué celdas están seleccionadas en ese momento, el valor de la celda activa, el nombre de la hoja activa o el número de la ventana actual. Desde la ventana **Inmediato** puedes obtener fácilmente respuestas a estas y otras preguntas.

En el ejercicio anterior, se introdujeron varias instrucciones. Volvamos a la ventana **Inmediato** para hacer algunas de estas preguntas. Excel recuerda las últimas instrucciones introducidas en esta ventana incluso después de cerrarla. Ten en cuenta que el contenido de la ventana **Inmediato** se borra automáticamente al salir de Excel.

1. Haz clic con el botón derecho del ratón en la segunda línea de la ventana **Inmediato**, donde previamente introdujiste la instrucción `Range("A1:A4").Select`.
2. Presiona **Intro** para que Excel vuelva a seleccionar las celdas A1:A4.
3. Haz clic al final de todo el contenido de la ventana **Inmediato** para crear una línea nueva. Introduce la siguiente instrucción y presiona **Intro**:

   ```
   ?Selection.Address
   ```

   Al pulsar **Intro**, Excel no seleccionará nada en la hoja. En su lugar mostrará el resultado de la instrucción en una línea nueva de la ventana **Inmediato**. En este caso Excel devuelve la dirección absoluta de las celdas que están seleccionadas en este momento ($A$1:$A$4).

   El signo de interrogación (?) le dice a Excel que muestre el resultado de la instrucción en la ventana **Inmediato**. En su lugar puedes utilizar la palabra `Print` como se muestra en el siguiente paso.

4. En una línea nueva de la ventana **Inmediato**, introduce la siguiente declaración y pulsa **Intro**:

   ```
   Print ActiveWorkbook.Name
   ```

   Excel introduce el nombre del libro activo en una nueva línea en la ventana **Inmediato**.

   ¿Qué tal si buscamos el nombre de la aplicación?

5. En una nueva línea de la ventana **Inmediato** introduce la siguiente instrucción y presiona **Intro**.

   ```
   ?Application.Name
   ```

   Excel mostrará el nombre completo (Microsoft Excel).

La ventana **Inmediato** también puede utilizarse para hacer cálculos:

4. En una nueva línea de la ventana **Inmediato**, introduce la siguiente declaración y pulsa **Intro**:

   ```
   ?12/3
   ```

Excel muestra el resultado de la división en la siguiente línea. Pero ¿qué pasa si quieres saber inmediatamente cuál es el resultado de 3+2 y 12*8?

En lugar de introducir estas instrucciones en líneas separadas, puedes introducirlas en una línea como en el siguiente ejemplo:

```
?3+2;12*8
```

Fíjate en el punto y coma que separa los dos bloques de instrucciones. Cuando presionas la tecla **Intro**, Excel muestra los resultados 5 y 96 separados por unos espacios.

Para eliminar las instrucciones de la ventana **Inmediato** asegúrate de que el cursor se encuentra en ventana **Inmediato**, presiona **Ctrl+A** para resaltar todas las líneas y luego presiona **Eliminar**.

9 Hojas, rangos y celdas

Cuando estés listo para escribir tu propio procedimiento VBA para automatizar una tarea de la hoja de cálculo, lo más probable es que comiences a buscar instrucciones que te permitan manipular las celdas. Necesitarás saber cosas como:

- Seleccionar celdas.
- Cómo introducir datos en las celdas.
- Asignar nombres de rango.
- Dar formato a las celdas.
- Mover, copiar y eliminar.
- …

Aunque estas tareas se pueden realizar fácilmente con el ratón o el teclado, dominar estas técnicas en VBA requiere un poco más de práctica. Debes usar el objeto `Range` para referirte a

una celda, un rango de celdas, una fila o una columna. Hay tres propiedades que te permiten acceder al objeto **Range**: la propiedad **Range**, la propiedad **Cells** y la propiedad **Offset**.

9.1 La propiedad Range

La propiedad **Range** devuelve una celda o un rango de celdas. La referencia al rango debe estar en formato A1 y entre comillas (por ejemplo "A1"). La referencia puede incluir el operador de rango, que es el carácter de dos puntos (por ejemplo "A1:B2") o el operador de unión, que es una coma (por ejemplo "A1","B2").

Seleccionar una celda (p. ejemplo A3)	`Range("A3").Select`
Seleccionar un rango de celdas (p. ejemplo A1:C5)	`Range("A1:5").Select`
Seleccionar celdas no contiguas (p. ejemplo A1, B6, C8).	`Range("A1, B6, C8").Select`
Seleccionar celdas y rangos no contiguos (p. ejemplo A11:D11, C12, D3)	`Range("A11:D11, C12, D3").Select`

9.2 La propiedad Cells

Puedes utilizar la propiedad **Cells** para devolver una sola celda. Cuando se selecciona una sola celda, **Cells** requiere dos argumentos. El primero indica el número de fila y el segundo el número de columna. Los argumentos se introducen entre paréntesis. Cuando se omiten los argumentos, Excel selecciona todas las celdas de la hoja activa. Echa un vistazo a la siguiente tabla:

Seleccionar una sola celda (p. ejemplo A3).	`Cells(3, 1).Select`
Seleccionar un rango de celdas (p. ejemplo A6:A10).	`Range(Cells(6, 1), Cells(10, 1)).Select`
Seleccionar todas las celdas de la hoja.	`Cells.Select`

Fíjate en cómo puedes combinar la propiedad **Range** y la propiedad **Cells**:

```
Range(Cells(6, 1), Cells(10, 1)).Select
```

En este ejemplo, la primera propiedad de las celdas devuelve la celda A6 y la segunda devuelve la celda A10. Ambas celdas devueltas por la propiedad **Cells**, son usadas como referencias para el objeto **Range**.

El resultado es que Excel seleccionará el rango de celdas en el que la celda superior se especifica por el resultado de la primera propiedad **Cells** y la celda inferior se define por el resultado de la segunda propiedad **Cells**.

Una hoja de cálculo es una colección celdas. También puedes usar la propiedad **Cells** con un único argumento que identifique la posición de una celda en la colección de celdas de una hoja de cálculo. Excel numera las celdas de la siguiente manera: La celda A1 es la primera celda de la hoja. B1 es la segunda, C1 es la tercera, y así sucesivamente. La celda 16.384 es la última celda de la primera fila de la hoja. Ahora vamos a escribir algunas instrucciones:

Seleccionar la celda A1.	`Cells(1).Select`
	o
	`Cells.Item(1).Select`

Seleccionar la celda C1.	`Cells(3).Select`
	o
	`Cells.Item(3).Select`

Seleccionar la celda XFD1.	`Cells(16384).Select`
	o
	`Cells.Item(16384).Select`

Observa que la palabra **Item** es una propiedad que devuelve un solo miembro de la colección. Como **Item** es el miembro predeterminado de una colección, puedes referirte a un **Item** de la hoja sin usar explícitamente esa palabra.

Ahora que has descubierto dos formas de seleccionar celdas (con Range y con Cells), te preguntarás por qué deberías molestarte en usar **Cells** si es más complicada de leer. Es cierto que la propiedad **Range** es más legible (a la hora de seleccionar celdas en la hoja siempre has utilizado referencias del tipo A1:C5). Utilizar **Cells** en vez de **Range** es más conveniente cuando se trata de trabajar con celdas como una colección. Utiliza **Cells** para acceder a una sola celda o a todas las celdas de una colección.

9.3 La propiedad Offset

Otra forma muy versátil de referirse a una celda de la hoja es con la propiedad **Offset**. A menudo, cuando se automatizan tareas, puede que no sepas exactamente dónde se encuentra una celda en concreto. ¿Cómo puedes seleccionar una celda cuya dirección no conoces? La solución es seleccionar una celda relativa a la que sí conoces.

La propiedad **Offset** devuelve un nuevo rango desplazando la selección inicial hacia arriba o hacia abajo un número determinado de filas. También puedes desplazar la selección a la derecha o a la izquierda un número determinado de columnas. En el cálculo de la posición de

la nueva celda, `Offset` utiliza dos argumentos. El primero indica la fila y el segundo la columna. Vamos a probar algunas instrucciones.

Seleccionar una celda situada una fila hacia abajo y tres columnas hacia la derecha de la celda A1.	`Range("A1").Offset(1, 3).Select`
Seleccionar una celda situada dos filas hacia arriba y una columna hacia la izquierda de la celda D15.	`Range("D15").Offset(-2, -1).Select`
Seleccionar una celda situada una fila por encima de la celda activa. Si la celda activa está en la primera fila, se obtiene un error.	`ActiveCell.Offset(-1, 0).Select`

En el primer ejemplo, Excel selecciona la celda D2. En el segundo, se selecciona la celda C13. Si las celdas A1 y D15 ya estuviesen seleccionadas podrías reescribir las dos primeras instrucciones de la siguiente forma:

```
Selection.Offset(1, 3).Select
Selection.Offset(-2, -1).Select
```

Observa que en el tercer ejemplo de la tabla anterior se muestra un 0 como segundo argumento. El cero introducido como primer o segundo argumento de la propiedad `Offset` indica la fila o columna actual. La instrucción `Cell.Offset(-1, 0).Select` causará un error si la celda activa se encuentra en la primera fila.

9.4 La propiedad Resize

Al trabajar con la propiedad Offset puede que necesites cambiar el tamaño del rango de celdas seleccionado. Supongamos que tienes seleccionado el rango A5:A10. ¿Qué ocurre si desplazamos la selección dos filas hacia abajo y dos columnas hacia la derecha y luego cambiamos el tamaño del rango seleccionado? Digamos que el nuevo rango debería resaltar las celdas C7:C8. La propiedad Offset puede ocuparse solo de la primera parte de esta tarea. La segunda parte requiere otra propiedad. Excel cuenta con la propiedad `Resize`, que modifica el tamaño del rango seleccionado. Puedes combinar la propiedad `Offset` con la propiedad `Resize` para responder a la pregunta anterior. Antes de combinarlas vamos a aprender a utilizarlas por separado:

1. Organiza tu pantalla de forma que se muestren tanto la ventana de Excel como la de VBA una al lado de la otra.
2. Activa la ventana **Inmediato** (si no lo está ya) e introduce las siguientes instrucciones:

```
Range("A5:A10").Select
Selection.Offset(2, 2).Select
```

```
Selection.Resize(2, 4).Select
```

La primera instrucción selecciona el rango A5:A10. La celda A5 es la celda activa. La segunda instrucción cambia el rango seleccionado por C7:C12. La celda C7 está ubicada dos filas por debajo de la celda activa (A5) y dos columnas a la derecha. Ahora la celda activa es C7.

La última instrucción redimensiona el rango actual. En lugar del rango C7:C12, se seleccionan C7:F8. Al igual que la propiedad `Offset`, `Resize` tiene dos argumentos. El primero es el número de filas que se quieren incluir en la selección y el segundo especifica el número de columnas. Por lo tanto, la instrucción `Selection.Resize(2, 4).Select` redimensiona la selección actual a dos filas y cuatro columnas.

Las dos últimas instrucciones se pueden combinar de la siguiente forma:

```
Selection.Offset(2, 2).Resize(2, 4).Select
```

En esta línea, la propiedad `Offset` calcula el comienzo de un nuevo rango, la propiedad `Resize` determina el nuevo tamaño del rango y el método `Select` selecciona el rango de celdas especificado.

9.5 La propiedad End

Si a menudo debes acceder rápidamente a ciertas celdas “lejanas” en tu hoja, es posible que ya conozcas atajos de teclado como:

- **Ctrl+flecha arriba**.
- **Ctrl+flecha abajo**.
- **Ctrl+flecha izquierda**.
- **Ctrl+flecha derecha**

En VBA puedes usar la propiedad `End` para moverte rápidamente por la hoja. Vamos a movernos por una hoja escribiendo las instrucciones que aparecen en la siguiente tabla:

Seleccionar la última celda de la fila.	`ActiveCell.End(xlToRight).Select`
Seleccionar la última celda de la columna.	`ActiveCell.End(xlDown).Select`
Seleccionar la primera celda de la fila.	`ActiveCell.End(xlToLeft).Select`
Selecciona la primera celda de la columna.	`ActiveCell.End(xlUp).Select`

Observa que la propiedad `End` requiere un argumento que indique la dirección en la que te quieres mover. Usa las siguientes constantes para saltar en la dirección especificada: `xlToRight, xlToLeft, xlUp, xlDown`.

Grabando la selección de celdas

Por defecto la grabadora de macros selecciona las celdas utilizando la propiedad `Range`. Si pones a funcionar la grabadora y seleccionas la celda A2, introduces cualquier texto y seleccionas las celda A5, verás las siguientes líneas de código en el editor de VBA:

```
Range("A2").Select
ActiveCell.FormulaR1C1 = "texto"
Range("A5").Select
```

Puedes hacer que la grabadora utilice la propiedad `Offset` si le dices que use referencias relativas. Para ello, haz clic en **Vista > Macros > Usar referencias relativas** y luego haz clic en **Grabar macro**. La grabadora de macros generará el siguiente código:

```
ActiveCell.Offset(-1, 0).Range("A1").Select
ActiveCell.FormulaR1C1 = "text"
ActiveCell.Offset(3, 0).Range("A1").Select
```

Cuando se graba un procedimiento utilizando referencias relativas, el procedimiento siempre selecciona una celda relativa a la celda activa. La primera y tercera instrucción hacen referencia a la celda A1 aunque en ningún momento se ha dicho nada sobre esta celda. Como recordarás del Capítulo 1, la grabadora de macros tiene su propia forma de hacer las cosas. Para hacer esta tarea más simple, puedes borrar la referencia a `Range("A1")`:

```
ActiveCell.Offset(-1, 0).Select
ActiveCell.FormulaR1C1 = "text"
ActiveCell.Offset(3, 0).Select
```

9.6 Mover, copiar y eliminar celdas

Mientras te encuentras creando una plantilla de Excel, a menudo debes mover y copiar celdas y eliminar su contenido. VBA permite automatizar estas tares de edición con tres métodos muy sencillos: `Cut`, `Copy` y `Clear`. A continuación se muestran algunos ejemplos:

Mover el contenido de la celda A5 a A4.	`Range("A5").Cut` `Destination:=Range("A4")`
Copiar una fórmula de la celda A3 al rango D5:F5.	`Range("A3").Copy` `Destination:=Range("D5:F5")`
Eliminar el contenido de A4.	`Range("A4").Clear` `o`

	`Range("A4").Cut`

Observa que las dos primeras instrucciones requieren un argumento llamado **`Destination`**. Este argumento especifica la dirección de una celda o rango de celdas donde se quiere colocar los datos cortados o copiados. El último ejemplo utiliza el método **`Cut`** sin el argumento de destino para eliminar los datos de la celda especificada.

El método **`Clear`** borra absolutamente todo de la celda o rango. Es decir, elimina, además del contenido, los formatos y comentarios. Si deseas ser más específico en el borrado de celdas, utiliza los siguiente métodos:

- **`ClearConstents`**: Únicamente borra los datos de las celdas.
- **`ClearFormats`**: Solo borra los formatos aplicados.
- **`ClearNotes`**: Elimina las notas de todas las celdas especificadas.
- **`ClearHyperlinks`**: Elimina todos los hipervínculos del rango.
- **`ClearOutline`**: Elimina los esquemas del rango.

10 Filas y columnas

Excel utiliza las propiedades **`EntireRow`** y **`EntireColumn`** para seleccionar toda la fila o columna. Se muestran algunos ejemplos:

Selecciona la fila donde se encuentra la celda activa.	`Selection.EntireRow.Select`
Selecciona la columna donde se encuentra la celda activa.	`Selection.EntireColumn.Select`

Al seleccionar un rango de celdas, es posible que quieras saber cuántas filas o columnas están incluidas en la selección. Vamos a hacer que Excel cuente las filas y columnas en el

`Range("A1:D5 ")`.

1. Escribe la siguiente instrucción de VBA en la ventana **Inmediato** y presiona **Intro**:

```
Range("A1:D15").Select
```

Si la ventana de Excel se muestra en la pantalla, VBA resaltará el rango A1:D15.

2. Para saber cuántas filas hay en el rango seleccionado, introduce la siguiente declaración:

```
?Selection.Rows.Count
```

Al presionar **Intro**, VBA muestra la respuesta en la siguiente línea. El rango contiene 15 filas.

3. Para conocer el número de columnas del rango, utiliza la siguiente instrucción:

```
?Selection.Columns.Count
```

La respuesta a esta pregunta es que el rango ocupa cuatro columnas.

4. En la ventana **Inmediato**, coloca el cursor en cualquier lugar dentro de la palabra `Rows` o `Columns` y presiona **F1** para obtener más información sobre estas propiedades.

10.1 Cómo obtener información de la hoja

¿Qué tamaño tiene una hoja de un libro de Excel? ¿Cuántas columnas y filas contiene? Si alguna vez olvidas estos detalles, usa la propiedad `Count` como muestra la siguiente tabla:

Contar el número de filas.	`?Rows.Count`
Contar el número de columnas.	`?Columns.Count`

11 Introducción de datos y formato de celdas

Los tipos de datos que podemos introducir en una hoja de Excel pueden ser texto, números o fórmulas. Para introducirlos en una celda o rango, puedes utilizar la propiedad `Value` o la propiedad `Formula` del objeto `Range`.

- Con la propiedad `Value`:

```
ActiveSheet.Range("A1:C4").Value = "=4 * 25"
```

- Con la propiedad `Formula`:

```
ActiveSheet.Range("A1:C4").Formula = "=4 * 25"
```

En ambos ejemplos el resultado es 100, y se muestra en las celdas A1:C4 (El producto de 4 por 25).

Introducir en la celda A5 el texto “Hola mundo”.	`Range("A5").Formula = "Hola mundo"`

Introducir el número 123 en la celda D21.	`Range("D21").Formula = 123`
	o
	`Range("D21").Value = 123`

Introducir en B4 la fórmula D21*3	`Range("B4").Formula = "=D21 * 3"`

11.1 Cómo devolver la información introducida en una hoja

En algunos procedimientos de VBA, seguramente tendrás que mostrar el contenido de una celda o rango de celdas. Aunque puedes utilizar las propiedades **`Value`** o **`Formula`**, esta vez no se pueden combinar.

- La propiedad **`Value`** muestra el resultado de una fórmula introducida en una celda. Si, por ejemplo, la celda contiene la fórmula =4*25, la instrucción **`Range("a1").Value`** devolverá el valor de 100.
- Si quieres mostrar la fórmula en lugar del resultado debes utilizar la propiedad **`Formula`**. La instrucción **`Range("A1").Formula`** devolverá =4*25 en lugar de 100.

11.2 Obtener información sobre el formato de las celdas

Una tarea muy común de la hoja de cálculo es la de aplicar formato a una o varias celdas. Tus procedimientos podrían necesitar averiguar el tipo de formato aplicado. Para recuperar este formato, utiliza la propiedad **`NumberFormat`**.

```
?Range("A1").NumberFormat
```

Al introducir esta instrucción en la ventana **Inmediato**, Excel muestra la palabra “General”, lo que indica que la celda no tiene ningún formato especial. Para cambiar el formato de una celda a Moneda, introduce la siguiente instrucción:

```
Range("A1").NumberFormat = "#.##0,00 €"
```

Si escribes “125” en la celda A1 después de haber dado formato con la instrucción anterior, se mostrará el valor 125,00 €. Puedes buscar los códigos de formato posibles en el cuadro de diálogo **Formato de celdas** (Presionando **Ctrl+1** en la ventana de Excel).

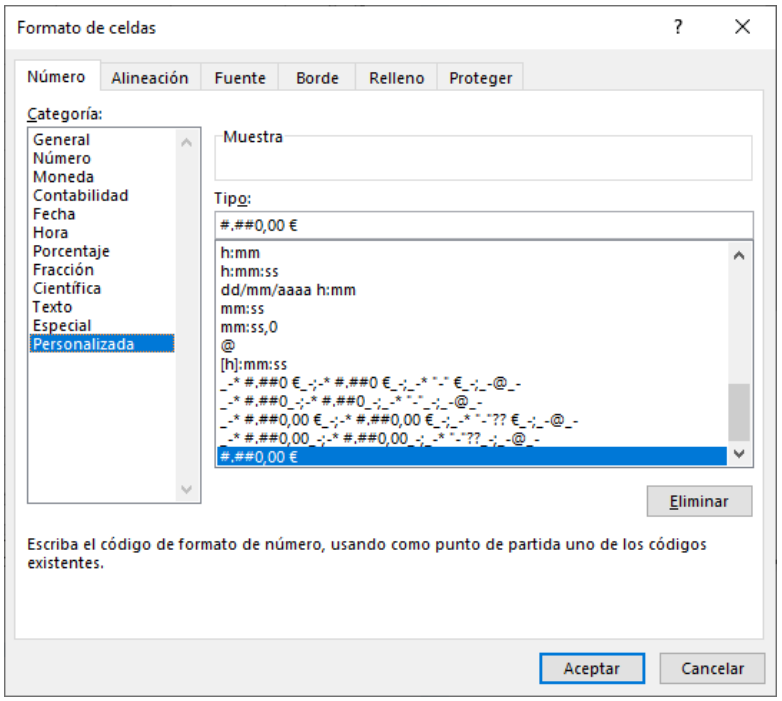


Imagen 11.1 Es posible aplicar diferentes formatos a celdas y rangos seleccionados utilizando los códigos de formatos, como se muestra en la categoría Personalizada del cuadro Formato de celdas.

12 Libros y hojas

Ahora que ya conocemos el funcionamiento de VBA con celdas y rangos, es hora de subir un nivel y aprender a controlar tanto un libro de Excel de forma individual como una colección completa de libros. No puedes crear una nueva hoja de cálculo si no sabes cómo abrir un libro. No puedes quitar de la pantalla un libro sino sabes cómo cerrarlo. No puedes trabajar con un libro existente si no sabes cómo abrirlo, etc. Estas tareas tan "sencillas" desde la ventana de Excel, tienen sus propios métodos en VBA. Las siguientes tablas nos darán las habilidades necesarias para gestionar tanto libros como hojas:

Crear un libro nuevo.	`Workbooks.Add`
Mostrar el nombre del primer libro.	`?Workbooks(1).Name`
Mostrar el número de libros abiertos.	`?Workbooks.Count`
Activar el segundo libro abierto.	`Workbooks(2).Activate`
Cerrar el libro "Capítulo 2 - Cuadro de texto.xlsm" guardando los cambios.	`Workbooks("Capítulo 2 - Cuadro de texto.xlsm ").Close SaveChanges:=True`

Abrir el libro “Capítulo 2 - Cuadro de texto.xlsm”.	`Workbooks.Open " C:\Users\Usuario\Desktop\VBA\ Capítulo 2 - Cuadro de texto.xlsm"`
Activar el libro “Capítulo 2 - Cuadro de texto.xlsm”.	`Workbooks("Capítulo 2 - Cuadro de texto.xlsm ").Activate`
Guardar el libro activo con el nombre “Horarios.xlsm”.	`ActiveWorkbook.SaveAs Filename:= "Horarios.xlsm"`
Cerrar el primer libro.	`Workbooks(1).Close`
Cerrar el libro activo sin guardar los cambios.	`ActiveWorkbook.Close SaveChanges:=False`
Cerrar todos los libros abiertos.	`Workbooks.Close`

Algunas tareas fundamentales para trabajar con hojas son, por ejemplo, agregar una nueva hoja a un libro, seleccionar la hoja o un grupo de hojas, cambiarles el nombre, copiarlas, moverlas o eliminarlas. En VBA cada una de estas tareas se realiza con un método o propiedad especial:

Insertar una hoja nueva.	`Worksheets.Add`
Mostrar el nombre de la primera hoja del libro.	`?Worksheets(1).Name`
Seleccionar la tercera hoja del libro.	`Worksheets(3).Select`
Seleccionar las hojas 1, 3 y 4.	`Worksheets(Array(1,3,4)).Select`
Activar la hoja “Ventas”.	`Worksheets("Ventas").Activate`
Mover la “Hoja2” delante de la “Hoja1”.	`Worksheets("Hoja2").Move Before:=Worksheets("Hoja1")`
Renombrar la “Hoja2” a “Ventas”.	`Worksheets("Hoja2").Name ="Ventas"`
Conocer el número de hojas del libro.	`?Worksheets.Count`
Eliminar la hoja “Gastos” del libro activo.	`Worksheets("Gastos").Delete`

Observemos la diferencia entre los métodos `Select` y `Activate`:

- Los dos pueden ser utilizados indistintamente si solo se selecciona una hoja de trabajo.

- Si se seleccionan varias hojas, el método `Activate` nos permite decidir cuál de las hojas será la hoja activa. Como sabemos solo puede estar activa una de ellas al mismo tiempo.

13 Ventanas

Cuando trabajamos con varios libros de Excel y necesitamos comparar o consolidar datos, o simplemente deseamos ver diferentes partes de la misma hoja, se deben utilizar los comandos del grupo **Ventana** de la ficha **Vista**.

Mostrar el libro activo en una ventana nueva.	`ActiveWorkbook.NewWindow`
Mostrar en la pantalla todos los libros abiertos.	`Windows.Arrange`
Activar la segunda ventana.	`Windows(2).Activate`
Mostrar el título de la ventana activa.	`?ActiveWindow.Caption`
Cambiar el título de la ventana activa por "Ayuda Excel".	`ActiveWindow.Caption = "My Window"`

Es posible decidir cómo se muestran las diferentes ventanas de Excel en la pantalla. El método `Arrange` tiene muchos argumentos, como se muestra en la siguiente tabla. El argumento que nos permite controlar la forma en la que se ubican las ventanas en la pantalla se llama `ArrangeStyle`. Si omitimos este argumento, todas las ventanas se colocan en mosaico.

`xlArrangeStyleTiled`	1	Ventanas en mosaico (valor por defecto).
`xlArrangeStyleCascade`	7	Ventanas en cascada.
`xlArrangeStyleHorizontal`	2	Ventanas en horizontal.
`xlArrangeStyleVertical`	3	Ventanas en vertical.

En lugar de los nombres de las constantes podemos utilizar los valores equivalentes.

Para mostrar en cascada todas las ventanas de Excel, utilizamos la siguiente instrucción:

```
Windows.Arrange ArrangeStyle:=xlArrangeStyleCascade
```

O simplemente:

```
Windows.Arrange ArrangeStyle:=7
```

14 La aplicación Excel

El objeto **Application** representa la propia aplicación de Excel. Al controlar el objeto **Application** podemos realizar muchas tareas, como guardar el aspecto de la pantalla al finalizar el trabajo o salir de la aplicación. Como sabemos, Excel nos permite guardar la configuración de la pantalla mediante el botón **Guardar espacio de trabajo** en la ficha **Vista**. Pero esta tarea también se puede realizar fácilmente con VBA:

```
Application.SaveWorkspace "Proyecto"
```

Esta instrucción guarda la configuración de la pantalla en el archivo del espacio de trabajo llamado **Proyecto**.

La próxima vez que necesitemos trabajar con los mismos archivos y disposición de las ventanas, simplemente abriremos el archivo Proyecto.xlwx para que Excel cargue los archivos correctos y restaure la pantalla con esa configuración. Ahora vamos a escribir algunas declaraciones que utilizan el objeto **Application**.

Comprueba el nombre de la aplicación activa.	`?Application.Name`
Cambia el título de la aplicación Excel por “Mi aplicación Excel”.	`Application.Caption ="Mi aplicación Excel"`
Cambia de nuevo el título de la aplicación por “Microsoft Excel”.	`Application.Caption ="Microsoft Excel"`
Muestra el sistema operativo que estamos utilizando.	`?Application.OperatingSystem`
Muestra el nombre de la persona registrada en la aplicación.	`?Application.OrganizationName`
Muestra el nombre de la carpeta donde se encuentra el archivo ejecutable de Excel (Excel.exe).	`?Application.Path`
Cierra Excel.	`Application.Quit`

15 Resumen

Este capítulo nos ha dado una visión general de la ventana del Editor de VBA. Hemos aprendido muchos términos básicos de VBA y los hemos practicado ejecutando declaraciones concretas en la ventana Inmediato.

En el próximo capítulo aprenderemos cómo se pueden almacenar datos para su uso posterior en variables.

También exploraremos los tipos de datos y constantes.

Capítulo 3
El modelo de objetos, las variables y las constantes

En la programación, al igual que en la vida, ciertas cosas deben hacerse inmediatamente mientras que otras pueden posponerse para más tarde. Cuando pospones una tarea, puedes introducirla en tu lista mental de tareas y clasificarla por su tipo o importancia.

Cuando delegas esa tarea o finalmente la realizas tú mismo, la tachas de la lista.

Este capítulo muestra cómo los procedimientos de VBA pueden memorizar "cosas" importantes para más tarde utilizarlas en declaraciones o cálculos. Aprenderás cómo un procedimiento puede mantener una "tarea" pendiente en una variable, cómo se declaran esas variables y cómo se relacionan con los tipos y constantes de datos.

1 Objetos de Excel, propiedades y métodos

Es posible crear procedimientos que controlen muchas características de Microsoft Excel utilizando VBA. También es posible controlar muchas otras aplicaciones. El poder de VBA viene de su habilidad para controlar y manejar varios objetos. Pero, ¿qué es un objeto?

Un objeto es un elemento que puedes controlar con VBA. Libros de trabajo, una hoja, un rango de una hoja, un gráfico o una barra de herramientas son solo algunos ejemplos de los objetos que se pueden controlar. Excel contiene un gran número de objetos manipulables de diferentes formas. Todos estos objetos están organizados en forma de jerarquía. Algunos objetos pueden contener otros objetos. Por ejemplo, Microsoft Excel es el objeto **`Application`**. Este objeto contiene otros objetos, como libros o barras de comandos. El objeto **`Workbooks`** puede contener otros objetos, como hojas o gráficos. En este capítulo aprenderás a controlar muchos de los objetos de Excel: rangos, ventanas, hojas, libros, etc. Comenzaremos con el objeto **`Range`**, ya que no llegaremos muy lejos sin saber manipular rangos de celdas…

Ciertos objetos son parecidos. Por ejemplo, si creas un nuevo libro de Excel y examinas las hojas que contiene, no encontrarás ninguna diferencia. Un grupo de objetos parecidos se llama colección. Una colección de hojas incluye todas las hojas de trabajo de un libro. Las colecciones, a su vez, también son objetos. En Excel las colecciones más utilizadas son:

- `Workbooks`: Representa todos los libros de Excel abiertos en ese momento.
- `Worksheets`: Representa todas las hojas de cálculo del libro de trabajo. Cada objeto Hoja representa una sola hoja.
- `Sheets`: Representa todas las hojas en un libro específico. Estas hojas pueden ser de cualquier tipo (hojas de cálculo o gráfico).
- `Windows`: Representa todas las ventanas de Excel. La colección `Windows` para el objeto `Application` contiene todas las ventanas de la aplicación, mientras que para el objeto `Workbook`, sólo contiene las ventanas del libro específico.

Cuando se trabaja con colecciones, es posible realizar la misma acción en todos los objetos de la colección. Cada objeto tiene algunas características que permiten describirlo. En VBA esas características se denominan **Propiedades**. Por ejemplo, un objeto `Wokbook` tiene una propiedad `Name`, y el objeto `Range` cuenta con propiedades como `Column`, `Font`, `Formula`, `Name`, `Row`, `Style` y `Value`. Las propiedades de los objetos se pueden modificar. Cuando se establece la propiedad de un objeto, se controla su apariencia o su posición. Las propiedades de los objetos solo pueden tener un valor específico a la vez. Por ejemplo, un libro no puede tener dos nombres diferentes al mismo tiempo.

La parte más difícil de entender de VBA es el hecho de que algunas propiedades también pueden ser objetos. Observemos el objeto `Range`. Podemos cambiar la apariencia del rango seleccionado estableciendo la propiedad `Font`. Pero la fuente puede tener diferentes nombres (Times New Roman, Arial, …), tamaños diferentes (10, 12, 15, …) y diferentes estilos (negrita, cursiva, subrayado, …). Estas son las propiedades de la fuente. Así que, si la fuente tiene propiedades, podemos considerar que es un objeto.

Las propiedades son geniales. Permiten cambiar el aspecto del objeto. Pero, ¿cómo se pueden controlar las acciones? Antes de hacer que Excel lleve a cabo algunas tareas, necesitas conocer otro concepto: **los métodos**. Cada acción que quieres que realice el objeto se llama método. Uno de los métodos más importantes de VBA es `Add`, que te permitirá añadir un nuevo libro u hoja. Los objetos pueden utilizar varios métodos. Por ejemplo, el objeto `Range` tiene métodos especiales que permiten borrar el contenido de la celda (`ClearContents`), borrar solo los formatos (`ClearFormats`) y borrar tanto el contenido como los formatos (`Clear`). Otros métodos permiten seleccionar, copiar o mover objetos.

Los métodos pueden tener parámetros opcionales que especifican cómo se debe llevar a cabo la acción. Por ejemplo, el objeto `Workbook` tiene un método llamado `Close`. Puedes cerrar cualquier libro de Excel utilizando este método. Si hay cambios en el libro, Excel mostrará un mensaje para que decidas si guardar los cambios o no. Puedes utilizar el método `Close` con el parámetro `SaveChanges` establecido en `False` para cerrar el libro sin guardar los cambios que hayas podido realizar en él. Observa el ejemplo:

```
Workbooks("Capítulo 1 - Inicio.xlsm").Close SaveChanges:=False
```

Cuando estás aprendiendo algo nuevo se hace necesario algún recurso que contenga toda la información necesaria.

Todos los objetos de Excel, así como sus propiedades y métodos se encuentran en la Referencia del Modelo de Objetos, a la que se puede acceder desde el menú **Ayuda – Ayuda de Microsoft Visual Basic para Aplicaciones** desde la ventana del editor de VBA. La **Imagen 1.1** muestra el modelo de objetos. Se puede acceder a esta página a través de esta URL:

https://docs.microsoft.com/es-es/office/vba/api/overview/excel/object-model

Los objetos se enumeran en orden alfabético para facilitar su consulta. Al hacer clic en el objeto verás subcategorías que muestran las propiedades, métodos y eventos de éste (veremos los eventos en el Capítulo 15). Revisar constantemente la referencia del modelo de objetos es una gran forma de aprender acerca de los objetos de Excel y las colecciones de objetos. El tiempo que inviertas en él, te dará muchos conocimientos cuando necesites escribir procedimientos complejos.

Imagen 1.1 El modelo de objetos de Excel contiene la documentación de todos los objetos, propiedades, métodos y eventos con los que es posible trabajar.

2 Escribir declaraciones simples y complejas en VBA

Ahora que conoces los elementos básicos de VBA (objetos, propiedades y métodos) es hora de comenzar a usarlos. ¿Cómo se combinan estos tres elementos para crear correctamente una instrucción? Como ya sabes, cada idioma tiene reglas gramaticales que seguimos para hacernos entender. Aunque nos comuniquemos en español, inglés, alemán u otro idioma, siempre estamos aplicando ciertas reglas en el habla o la escritura. En la programación se usa el concepto “sintaxis” para especificar las reglas del lenguaje. Es posible buscar la sintaxis de cada objeto, propiedad o método en la ayuda online o en la ventana del **Examinador de objetos**.

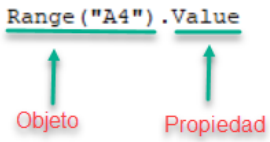
Para asegurarte de que Excel entienda siempre lo que quieres decir, debes seguir las siguientes reglas:

1. Hacer referencia a la propiedad de un objeto:

   Si la propiedad no tiene argumentos, la sintaxis es la siguiente.

   ```
   Objeto.Propiedad
   ```

   Siempre se debe mantener el orden. Por ejemplo, para referirte al valor introducido en la celda A4 de la hoja, puedes introducir la siguiente instrucción:

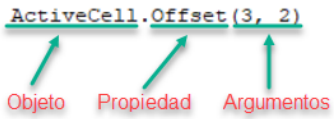


   Observa el punto entre el objeto y su propiedad. Cuando necesitas acceder a la propiedad un objeto debes utilizar siempre el operador punto (.) como se muestra:

   ```
   ActiveSheet.Shapes(2).Line.Weight
   ```

   Este ejemplo hace referencia a la propiedad **`Weight`** del borde del segundo objeto de la colección **`Shapes`** situado en la hoja activa.

   Algunas propiedades requieren uno o más argumentos. Por ejemplo, cuando se utiliza la propiedad **`Offset`**, puedes seleccionar una celda relativa a la celda activa. La propiedad **`Offset`** requiere dos argumentos. El primero indica el número de fila (**`rowOffset`**) y el segundo determina el número de columna (**`columnOffset`**).



   En este ejemplo, suponiendo que la celda activa es A1, **`Offset(3, 2)`** hará referencia a la celda situada tres filas hacia abajo y dos columnas hacia la derecha. Es decir, hace referencia a la celda C4. Como los argumentos colocados entre paréntesis son a menudo difíciles de entender, es una buena práctica preceder al valor del argumento con su nombre:

   ```
   ActiveCell.Offset(rowOffset:=3, columnOffset:=2)
   ```

   Observa los dos puntos y el signo igual, siempre acompañan al nombre del argumento. Cuando se utilizan los nombres de los argumentos, puedes establecerlos en cualquier orden. La instrucción anterior también podría escribirse de esta forma:

   ```
   ActiveCell.Offset(columnOffset:=2, rowOffset:=3)
   ```

   El orden de los argumentos no cambia el significado. Todavía sigue haciendo referencia a la celda C4, suponiendo que A1 sea la celda activa.

   Atención: si en este ejemplo, cambias el orden de los argumentos y no introduces los nombres de los argumentos, estarás haciendo referencia a otra celda.

2. Cambiar el valor de la propiedad de un objeto:

```
Objeto.Propiedad = Valor
```

"Valor" es el nuevo valor que deseas asignar a la propiedad del objeto. Este parámetro puede ser:

- Un número: La siguiente instrucción introduce el número 25 en la celda A4:

```
Range("A4").Value = 25
```

- Un texto (introducido entre comillas): La siguiente instrucción cambia la fuente de la celda activa a *Times New Roman*:

```
ActiveCell.Font.Name = "Times New Roman"
```

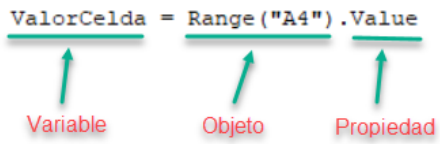
- Un valor lógico (verdadero o falso): La siguiente instrucción aplica el formato Negrita a la celda activa:

```
ActiveCell.Font.Bold = True
```

3. Devolver el valor actual de la propiedad del objeto:

```
Variable = Objeto.Propiedad
```

La variable es el lugar donde VBA va a guardar el valor de la propiedad. Un poco más adelante, en este capítulo aprenderás algo más sobre ello.



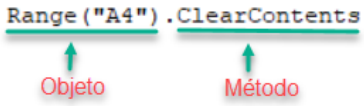
Esta instrucción almacena el valor de la celda A4 en la variable llamada **ValorCelda**.

4. Hacer referencia al método de los objetos:

En caso de que el método no tenga argumentos, la sintaxis sería la siguiente:

```
Objeto.Método
```

Al igual que las propiedades, los métodos también van precedidos del objeto al que hacen referencia y separados por un punto. En el siguiente ejemplo se elimina el contenido de la celda A4:



Si el método requiere argumentos la sintaxis es la siguiente:

```
Objeto.Método (Argumento1, Argumento2, …, ArgumentoN)
```

Por ejemplo, el método `GoTo` selecciona rápidamente cualquier rango del libro. La sintaxis sería esta:

```
Objeto.GoTo(Referencia, Desplazamiento)
```

El argumento `Referencia` es la celda o rango de destino. El argumento `Desplazamiento` se puede establecer como "Verdadero" para desplazarse por la ventana o "Falso" para mantenerla fija. Por ejemplo, la siguiente instrucción selecciona la celda P100 de la Hoja1 y se desplaza por la ventana:

```
Application.GoTo _
Reference:=Worksheets("Sheet1").Range("P100"), _
Scroll:=True
```

Esta instrucción no cabe en una sola línea por lo que se ha dividido en varias utilizando el carácter de subrayado (_).

Ahora supongamos que quieres borrar el contenido de la celda A4. Si lo hicieses manualmente, seleccionarías la celda A4 y presionarías la tecla **Suprimir**. Para realizar la misma acción con VBA, primero debes averiguar cómo hacer que Excel seleccione la celda apropiada. A4, como cualquier otra celda de la hoja, está representada por el objeto `Range`. VBA no tiene un método "Delete" para eliminar el contenido de las celdas. En su lugar, habría que utilizar el método `ClearContents`:

```
Range("A4").ClearContents
```

Fíjate en el punto entre el nombre del objeto y su método. Esta instrucción elimina el contenido de la celda A4. Pero, supongamos que también hay varios libros abiertos. ¿Cómo hacer que Excel elimine el contenido de la celda A4 ubicada en la primera hoja de trabajo del libro **Capítulo3 – Fundamendos.xlsm**? Si no quieres terminar borrando el contenido de la celda A4 en el libro u hoja equivocados, debes escribir una instrucción más detallada para que VBA sepa dónde localizar la celda adecuada:

```
Application.Workbooks("Capítulo 3 - Fundamentos.xlsm") _
.Worksheets("Sheet1").Range("A4").ClearContents
```

Esta instrucción se lee de derecha a izquierda de la siguiente forma: Borrar el contenido de la celda A4, que es parte de un rango que se encuentra en la hoja Hoja1, contenida en un libro llamado Capítulo3 – Fundamentos.xlsm y que, a su vez forma parte de la aplicación Excel.

Asegúrate de introducir la letra "s" al final de los nombres de la colección (`Workbooks` y `Worksheets`). Todas las referencias a los nombres de los libros, hojas y celdas deben ir entre comillas.

2.1 Cómo romper correctamente una línea de código

Cuando comiences a escribir procedimientos de VBA desde cero, necesitarás saber cómo dividir una instrucción en dos o más líneas para hacer el procedimiento más legible. VBA cuenta con un carácter especial de continuación de línea que puede utilizarse al final de una

línea para indicar que la siguiente es una continuación de la anterior, como en el siguiente ejemplo:

```
Selection.PasteSpecial _
        Paste:=xlValues, _
        Operation:=xlMultiply, _
        SkipBlanks: =False, _
        Transpose:=False
```

Como puedes comprobar el carácter de final de línea es el guion bajo o subrayado (_). Antes de introducirlo debes escribir un espacio. Puedes usarlo antes o después de:

- Operadores: Por ejemplo &, +, NOT, AND.
- Una coma.
- Un signo igual.
- Un operador de asignación (:=)

No es posible utilizar el guion bajo dentro de un texto entre comillas.

3 Cómo guardar los resultados de las declaraciones en VBA

En el Capítulo 2, mientras trabajabas con la ventana **Inmediato**, introdujiste varias declaraciones de VBA cuyos resultados eran datos. Por ejemplo, cuando introdujiste `?Rows.Count`, descubriste que una hoja de Excel contiene 1.048.576 filas.

Sin embargo, cuando escribes estos procedimientos de VBA fuera de la ventana **Inmediato** no puedes usar signos de interrogación. Si quieres saber el resultado después de ejecutar una instrucción, debes pedirle a VBA que lo memorice.

En la programación, los resultados devueltos por las instrucciones de VBA pueden escribirse en variables. Como las variables pueden contener varios tipos de datos, esta sección se centra en ellos. Una vez que entiendas los fundamentos de los tipos de datos, será fácil abordar la parte de las variables.

4 Introducción a los tipos de datos

Cuando creas un procedimiento de VBA tienes un objetivo en la mente: manipular datos. Como tus procedimientos manejarán diferentes tipos de información, debes entender cómo almacena los datos VBA.

El tipo de datos determina cómo se almacenan en la memoria de tu ordenador. Por ejemplo, pueden almacenarse como números, texto, fechas, objetos, etc. Si olvidas indicar a VBA el tipo de datos, éste le asigna el tipo de dato `Variant`. Este tipo es capaz de averiguar qué tipo de datos se están manipulando y adaptarse a él.

Los tipos de datos de VBA se muestran en la siguiente tabla. Además de los tipos de datos predefinidos, puedes definir tus propios tipos de datos. Como cada tipo de datos ocupa una cantidad diferente de espacio en la memoria del ordenador, algunos de ellos son más

“pesados” que otros. Por lo tanto, para optimizar la memoria y hacer que el procedimiento se ejecute más rápido, debes seleccionar el tipo de datos que utilice el menor número de bytes y que, al mismo tiempo, sea capaz de manejar los datos que tiene que manipular.

Boolean	2	Tiene sólo dos estados: Verdadero y Falso. Estas variables se utilizan generalmente como flags o condicionales.
Byte	1	Corresponde a una variable de 8 bits que puede almacenar valores de 0 a 255. Es muy útil para el almacenamiento de datos binarios.
Integer	2	Puede oscilar entre -32768 y 32767. Los enteros se deben utilizar cuando se trabaja con valores que no pueden contener números fraccionarios.
Long (Long Integer)	4	Puede variar entre -2.147.483.648 y 2.147.483.647. Las variables Long sólo puede contener valores enteros.
LongLong LongLong Integer)	8	
LongPtr	4 en 32 bits 8 en 64 bits	
Single (punto flotante de precisión sencilla)	4	Va desde -3,402823E38 a -1,401298E-45 para valores negativos y desde 1,401298E-45 a 3,402823E38 para valores positivos. Cuando se necesitan números fraccionarios dentro de este rango, este es el tipo de apropiado para su uso.
Double (punto flotante de precisión doble)	8	Se utiliza cuando se necesita una gran precisión. Estas variables pueden variar desde -1.79769313486232E308 a -4,94065645841247E-324 para valores negativos y de 4,94065645841247E-324 a 1.79769313486232E308 para valores positivos.

Currency (entero con escala)	8	Es en realidad un tipo entero internamente. En su uso se escala por un factor de 10.000 para agregar cuatro dígitos a la derecha del punto decimal. Permite hasta 15 dígitos a la izquierda del punto decimal, resultando en un rango de aproximadamente -922.337.000.000.000 a +922.337.000.000.000.
Decimal	14	Es un subtipo de dato Variant, puede almacenar valores en un rango que va desde -79.228.162.514.264.337.593.543.950.335 hasta 79.228.162.514.264.337.593.543.950.335 si el valor no contiene cifras decimales. Tiene una precisión de hasta 28 decimales con valores desde -7,9228162514264337593543950335 hasta 7,9228162514264337593543950335.
Date	8	Acepta la fecha o la hora, o ambas cosas. Los valores posibles van desde el 1 de enero del año 100 al 31 de diciembre del año 9999.
String (longitud variable)	10 + longitud de la cadena	Puede contener un máximo de aproximadamente 2 mil millones de caracteres. Cada carácter tiene un valor que va desde 0 hasta 255 basado en el juego de caracteres ASCII.
String (longitud fija)	Longitud de la cadena	
Object	4	Se utiliza cuando en el tiempo de compilación no se conoce a qué tipo de datos puede señalar la variable. Independientemente del tipo de datos al que haga referencia, una variable Object no contiene el valor en sí, sino una referencia a su valor.
Variant (con números)	16	Almacenan valores numéricos y no numéricos. Son los más flexibles de todos los tipos disponibles, ya que almacenan valores muy grandes de casi cualquier tipo (coincide con el tipo de datos numérico Doble). Se usa sólo cuando no se está seguro del tipo o cuando se están introduciendo datos externos y no se está seguro de las especificaciones del tipo de datos.
Variant (con texto)	22 + longitud de la cadena	

Definida por el usuario	Varía	El intervalo de cada elemento es el mismo que el intervalo de su tipo de datos.

Nota: Para más información puedes visitar la página de Microsoft donde se explican cada uno de los tipos de datos.

https://docs.microsoft.com/es-es/office/vba/language/reference/user-interface-help/data-type-summary

5 El uso de variables

Una variable no es más que un nombre que se utiliza para referirse a un elemento de datos. Cada vez que quieras recordar el resultado de una instrucción de VBA, piensa en un nombre que lo represente. Por ejemplo, si el número 1.048.576 debe recordar el número total de filas de una hoja (información muy importante cuando quieres importar datos externos a Excel), puedes inventar un nombre como TodasFilas, NumeroFilas o TotalFilas. Los nombres de variables pueden contener caracteres, números y algunos signos de puntuación, excepto: , # $ % & @ !

El nombre de la variable no puede comenzar con un número o contener espacios. Si deseas que el nombre de la variable contenga más de una palabra, utiliza el carácter guion bajo (_) como separador. Aunque el nombre de una variable puede contener hasta 254 caracteres, es mejor utilizar nombres cortos y simples (nos ahorrarán tiempo de escritura cuando hagamos referencia a la variable en el procedimiento). A VBA no le importa si utilizas mayúsculas, minúsculas o ambos, pero la mayoría de los programadores utilizan letras minúsculas. Otros utilizan la primera letra de cada palabra en mayúsculas y el resto en minúsculas.

Atención: Puedes utilizar cualquier nombre para las variables, excepto las palabras reservadas que usa VBA. Las declaraciones que tienen un significado especial para VBA no se pueden utilizar. Por ejemplo, `Name`, `Len`, `Empty`, `Local`, `Currency` o `Exit`, generarán un mensaje de error si tratas de utilizarlas como nombres de variable.

Un consejo: Nombra tus variables de forma que te ayuden a recordar sus funciones. Algunos programadores utilizan un prefijo para identificar el tipo de variable. Por ejemplo, cuando comienza por “str” significa que es una variable que almacena texto (del tipo `String`).

5.1 Cómo crear una variable

Puedes crear una variable declarándola con un comando especial o simplemente utilizándola en una instrucción. Cuando declaras tu variable, estás haciendo que VBA conozca su nombre y el tipo de datos. Esto se llama declaración explícita de variables.

Declarar explícitamente una variable tiene algunas ventajas:

- Acelera la ejecución del procedimiento, ya que VBA conoce el tipo de datos y reserva solo la memoria necesaria para almacenar la información.
- El código es más fácil de leer y entender porque todas las variables se enumeran al principio del procedimiento.
- Ayuda a prevenir errores causados por nombres de variables mal escritos. VBA corrige automáticamente el nombre de la variable basándose en el nombre utilizado en la declaración.

Cuando no declaras la variable antes de utilizarla, ésta se declara automáticamente la primera vez que la utilizas en el código. A las variables no declaradas, VBA les asigna automáticamente el tipo de datos **`Variant`**. Aunque la declaración implícita es conveniente (permite crear variables sobre la marcha y asignar valores sin conocer de antemano el tipo de datos de los valores que se asignan), puede causar varios problemas:

- Si se escribe mal el nombre de la variable en el procedimiento, VBA puede mostrar un error en tiempo de ejecución o crear una nueva variable. Seguramente perderás algo de tiempo solucionando el problema (que podría haberse evitado fácilmente si hubieras declarado la variable al principio del procedimiento).
- Como VBA no sabe qué tipo de datos quieres almacenar, le asigna el tipo de datos **`Variant`**. Esto hace que el procedimiento se ejecute más lento porque VBA debe comprobar el tipo de datos cada vez que accede a la variable. Como una **`Variant`** puede almacenar cualquier tipo de dato, VBA debe reservar más memoria para almacenar los datos.

5.2 Cómo declarar una variable

Una variable se declara mediante la palabra clave **`Dim`**. Dim significa “dimensión”. Va seguida del nombre de la variable y luego del tipo.

Supongamos que quieres que el procedimiento muestre la edad de un empleado. Antes de poder calcular la edad, debes decirle al procedimiento la fecha de nacimiento del empleado. Para ello puedes declarar una variable llamada **`FechaNacimiento`** de la siguiente forma:

```
Dim FechaNacimiento As Date
```

Observa que la palabra **`Dim`** va seguida del nombre de la variable. Este nombre puede ser cualquier combinación de caracteres que no sea una de las palabras clave de VBA.

A continuación, debes especificar el tipo de datos que tendrá la variable escribiendo la palabra “As” después del nombre, seguida de uno de los tipos de datos que vimos anteriormente. El tipo de datos **`Date`** le dice a VBA que la variable **`FechaNacimiento`** almacenará una fecha. Para almacenar la edad del empleado, declara una variable de edad de la siguiente manera:

```
Dim Edad As Integer
```

Esta variable almacenará el número de años entre la fecha de hoy y la fecha de nacimiento del empleado. Como la edad se muestra como un número entero, a esta variable se le ha asignado el tipo **`Integer`**.

También es posible que tu procedimiento lleve un registro del nombre del empleado, de forma que declara otra variable para almacenar el nombre y el apellido del empleado:

```
Dim NombreCompleto As String
```

Declarar las variables se considera una buena práctica de programación, pues facilita la lectura de los procedimientos y ayuda a prevenir ciertos tipos de errores.

Ahora que ya sabes cómo declarar variables, veamos un procedimiento que los usa:

```
Sub CalcularEdad()
    ' Declaración de variables
    Dim NombreCompleto As String
    Dim FechaNacimiento As Date
    Dim Edad As Integer
    ' Asignar valores a las variables
    NombreCompleto = "María Antonia Mendoza"
    FechaNacimiento = #1/3/1981#
    ' Calcular edad
    Edad = Year(Now()) - Year(FechaNacimiento)
    ' Mostrar el resultado en la ventana inmediato
    Debug.Print NombreCompleto & " tiene " & Edad & " años."
End Sub
```

Las variables se declaran al principio del procedimiento en el que se van a utilizar. En este procedimiento las variables se declaran en líneas separadas, pero pueden declararse varias en la misma línea, separando cada nombre de la variable con una coma:

```
Dim NombreCompleto As String, FechaNacimiento As Date, _
Edad As Integer
```

Observa que la palabra `Dim` aparece solo una vez al principio de la línea.

Cuando VBA ejecuta las declaraciones de las variables, reserva espacio en la memoria para almacenarlas. A continuación, se asignarán valores específicos a estas variables.

Para asignar un valor a una variable, comienza escribiendo el nombre de la variable seguido del signo igual. El valor introducido a la derecha del signo igual es el dato que deseas almacenar en la variable. Los datos que introduzcas aquí deben ser del mismo tipo que introdujiste en la declaración de la variable. Los datos de texto deben estar entre comillas y las fechas entre almohadillas (#).

Usando los datos suministrados por la variable `FechaNacimiento`, VBA calcula la edad de un empleado y almacena el resultado del cálculo en la variable Edad. A continuación, se muestra en la ventana **Inmediato**, el nombre completo del empleado, así como su edad.

Concatenación

Es posible combinar dos o más cadenas de texto para formar una nueva cadena. La operación de unión se llama "concatenación". Acabamos de ver un ejemplo de cadenas concatenadas en los procedimientos **CalcularEdad** y **Numerofilas**. La concatenación está representada por un ámpersand (&). Por ejemplo, **"Mi nombre es " & Nombre**, dará como resultado "Mi nombre es Sergio". El nombre de la persona está determinado por el contenido de la variable **Nombre**. Fíjate en el espacio en blanco que hay detrás de "es " . Sirve para separar el texto del valor de la variable.

Veamos qué ocurre cuando se declara una variable con el tipo de datos incorrecto. La finalidad del siguiente procedimiento es calcular el número total de filas de una hoja y luego mostrar los resultados en un cuadro de diálogo:

```
Sub NumeroFilas()
    Dim Filas As Integer
    Filas = Rows.Count
    MsgBox "La hoja tiene " & Filas & " filas."
End Sub
```

Un tipo de datos incorrecto puede causar un error. En el procedimiento anterior, cuando VBA intenta escribir el resultado de la instrucción **Rows.Count** en la variable **Filas**, el procedimiento falla y Excel muestra el mensaje "Se ha producido el error '6' en tiempo de ejecución – Desbordamiento". Este error se produce por asignar un tipo de datos incorrecto para esa variable. El número de filas de una hoja no se ajusta al rango de datos **Integer**. Para corregir el problema, debes elegir un tipo de datos que pueda almacenar un número más alto:

```
Sub NumeroFilas()
    Dim Filas As Long
    Filas = Rows.Count
    MsgBox "La hoja tiene " & Filas & " filas."
End Sub
```

También puedes corregir el problema borrando el tipo de variable. Cuando ejecutes de nuevo el procedimiento, VBA asignará el tipo de datos **Variant** a la variable. Aunque las **Variant** consumen más memoria que cualquier otro tipo de variable y disminuyen la velocidad a la que se ejecutan los procedimientos, si las estamos utilizando en procedimientos pequeños, la diferencia es imperceptible.

5.3 Especificar el tipo de dato de la variable

Si no especificas el tipo de datos de la variable en la declaración **Dim**, terminas con una variable no definida. Estas variables sin definir se consideran siempre de tipo **Variant**.

Es muy recomendable crear variables tipificadas. Cuando declaras una variable de un cierto tipo de datos, el procedimiento se ejecuta más rápido porque VBA no tiene que detenerse a analizar la variable para determinar su tipo.

VBA puede trabajar con muchos tipos de variables numéricas. Las variables **Integer** solo puede contener números enteros de -32.768 a 32.767.

Otro tipo de variables numéricas son **Long**, **Single**, **Double** y **Currency**. Las **Long** pueden contener números enteros de -2.147.483.648 a 2.147.483.647. A diferencia de las anteriores, las variables **Single** y **Double**, pueden contener decimales. **String** se utiliza para referirse a texto. Cuando declaras una variable de datos **String** puedes decirle a VBA cómo será de grande. Por ejemplo, en la instrucción:

```
Dim Extension As String * 3
```

Se declara la variable **Extension** como tipo texto de tres caracteres de longitud. Si no asignas una longitud específica, la variable **String** será dinámica.

Esto significa que VBA dejará suficiente espacio en la memoria del ordenador para gestionar cualquier cantidad de texto que se le asigne.

Después de declarar una variable, sólo se puede almacenar en ella el tipo de información que se determinó al crearla. Asignar valores de texto (**String**) a variables numéricas o valores numéricos a variables de texto da como resultado el mensaje de Error "No coinciden los tipos" o hace que VBA modifique el valor. Por ejemplo, si una variable fue declarada para contener números enteros y los datos utilizan decimales, VBA no tendrá en cuenta los decimales y utilizará sólo la parte entera del número. Cuando se ejecuta el siguiente procedimiento, **MiNumero**, VBA modifica los datos para que se ajusten al tipo de datos de la variable (**Integer**) y, en lugar de 23,11, la variable termina teniendo un valor de 23.

```
Sub Mi_Numero()
    Dim Numero As Integer
    Numero = 23.11
    MsgBox Numero
End Sub
```

Si no se declara una variable con la instrucción **Dim**, todavía existe otra forma de designar un tipo de dato para ella utilizando un carácter especial al final del nombre de la variable. Para declarar la variable **Nombre** como **String**, puedes añadir el signo de dólar al nombre de la variable:

```
Dim Nombre$
```

Esta instrucción es la misma que **Dim Nombre As String**. Los caracteres especiales para designar variables son los siguientes:

Integer	%
Long	&
Single	!
Double	#
Currency	@
String	$

Como ves en la tabla, estos caracteres de declaración sólo pueden utilizarse con seis tipos de datos.

Nacimiento de una variable

Cuando VBA crea una nueva variable, se dice que la inicia. Entonces toman su valor por defecto.

Las variables numéricas se establecen en cero (0), las variables booleanas se inician en False, las variables de cadena son una cadena de texto vacía (“”) y las variables de fecha se inician en el 30 de diciembre de 1899.

5.4 Practicando la asignación de valores a variables

Ahora que ya conoces el procedimiento para nombrar y declarar variables y has visto ejemplos de uso de variables en procedimientos, vamos a seguir practicando con este ejercicio:

1. Crea un libro nuevo y llámalo “Capítulo 3 – Variables.xlsm”. Guárdalo en la carpeta Archivos Manual VBA que creaste en C:\.
2. Activa el editor de VBA.
3. En el **Explorador de proyectos**, selecciona el nuevo proyecto VBAProject(Capítulo 3 – Variables.xlsm). En la ventana **Propiedades** cambia el nombre del proyecto por Capitulo3.
4. Inserta un módulo nuevo desde el menú **Insertar – Módulo**.
5. Selecciona el Módulo 1 y cámbiale el nombre utilizando la ventana **Propiedades**. El nuevo nombre será “Variables”.
6. En la ventana **Código** introduce el procedimiento `Calculo_costes` siguiente:

```
Option Explicit
Sub Calculo_Costes()
    vtaPrecio = 35
    vtaImpuestos = 0.21
```

```
    Range("A1").Formula = "Calculadora de costes"
    Range("A4").Formula = "Precio"
    Range("B4").Formula = vtaPrecio
    Range("A5").Formula = "Impuestos"
    Range("A6").Formula = "Importe"
    Range("B5").Formula = vtaPrecio * vtaImpuestos
    coste = vtaPrecio + (vtaPrecio * vtaImpuestos)

    With Range("B6")
        .Formula = coste
        .NumberFormat = "0.00 €"
    End With

    strMensaje = "El importe total es de " & coste & " €."
    Range("A8").Formula = strMensaje
End Sub
```

El procedimiento anterior calcula el coste de un producto suponiendo que el precio del producto es de 35 € y que los impuestos a pagar son el 21 %.

El procedimiento utiliza cuatro variables: **vtaPrecio**, **vtaImpuestos**, **coste** y **strMensaje**.

Como ninguna de estas variables ha sido declarada explícitamente, todas tienen el mismo tipo de datos (**Variant**). Las variables **vtaPrecio** y **vtaImpuestos** se crearon asignándoles valores al principio del procedimiento. A la variable **coste** se le asigno el valor resultante del: **vtaPrecio + (vtaPrecio * vtaImpuestos)**. La variable **strMensaje** muestra un mensaje de texto. Este mensaje se introduce en una celda de la hoja de cálculo.

Cuando se asignan valores a las variables, se coloca un signo igual después del nombre de la variable. Tras esto, debes introducir el valor de la variable. Este valor puede ser un número, una fórmula o un texto entre comillas. Mientras que los valores asignados a las variables **vtaImpuestos** y **coste** son fáciles de entender, el valor que se almacena en **strMensaje** es un poco más complicado.

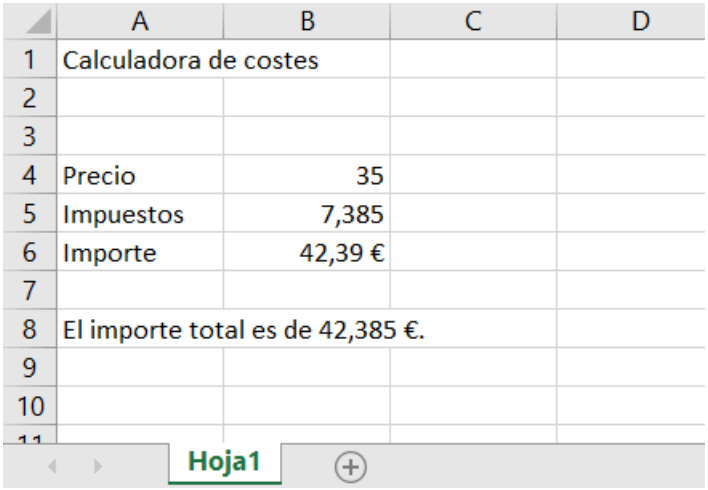
```
strMensaje = "El importe total es de " & coste & " €."
```

- La cadena de texto “El importe total es de ” está encerrada entre comillas. Observa que hay un espacio justo delante de las comillas finales.
- El signo de euro se utiliza para resaltar que el tipo de datos es “moneda”. Como se trata de un carácter, está encerrado entre comillas junto con el punto del final de la frase.

- El carácter `&` permite unir o concatenar otra cadena de texto o el contenido de una variable. El ámpersand se debe usar cuando quieras añadir una nueva información a la cadena anterior.
- La variable `coste` es el coste real del producto y se muestra cuando se ejecuta el procedimiento.

7. Ejecuta el procedimiento. Sitúa el cursor en cualquier lugar dentro del procedimiento `Calculo_Costes` y haz clic en **Ejecutar – Ejecutar Sub / UserForm**.
   Al ejecutarlo, VBA puede mostrar el mensaje "Error de compilación: Variable no definida". Si esto sucede haz clic en **Aceptar** para cerrar el cuadro del mensaje. VBA seleccionará la variable `vtaPrecio` y resaltará el nombre del procedimiento. La barra de título muestra "Microsoft Visual Basic – Capítulo 3 – Variables.xlsm [interrupción]". El modo de interrupción de VBA te permite corregir el problema antes de continuar. Más adelante en este libro, aprenderás a arreglar problemas en el modo interrupción. Por ahora, sal de este modo seleccionando **Ejecutar – Reiniciar**. Ahora ve a la parte superior de la ventana **Código** y borra la frase `Option Explicit` que aparece en la primera línea. Esta frase significa que todas las variables utilizadas en este módulo deben ser declaradas explícitamente.
   En la siguiente sección se explica esta declaración. Cuando elimines la línea, vuelve a hacer clic en **Ejecutar – Ejecutar Sub /UserForm**. Para volver a ejecutar el procedimiento. Esta vez el código funcionará sin problema.
8. Una vez finalizada la ejecución, presiona **Alt+F11** para ir a la ventana de Excel. También puedes hacer clic en el icono correspondiente de la barra de tareas de Windows.

   El resultado del procedimiento es el que se muestra en la Imagen 5.1.



	A	B	C	D
1	Calculadora de costes			
2				
3				
4	Precio	35		
5	Impuestos	7,385		
6	Importe	42,39 €		
7				
8	El importe total es de 42,385 €.			
9				
10				

Hoja1

Imagen 5.1 El procedimiento puede introducir datos y hacer cálculos en una hoja.

La celda A8 muestra el contenido de la variable `strMensaje`.

Observa que el importe calculado en la celda B6 tiene dos decimales, pero el que muestra la variable `strMensaje` muestra tres. Para mostrar el importe del producto con dos decimales en la celda A8, debes aplicar el formato requerido a la variable coste, no a la celda.

VBA tiene funciones especiales que permiten cambiar el formato de los datos. Para cambiar el formato de la variable coste, utilizarás la función **Format**. Esta función tiene la siguiente sintaxis:

```
Format (expresión, formato)
```

La expresión es el valor o variable alque se quieres dar formato, y formato es el tipo de formato que deseas aplicar.

9. En el editor de VBA selecciona todo el código del procedimiento y pégalo debajo, en la primera línea vacía. Si lo deseas, separa ambos procedimientos con algunos espacios.
10. Cambia el nombre del procedimiento. Se llamará **Calculo_Costes_Modificado**.
11. Modifica el cálculo de la variable coste, introduciendo la siguiente línea:

```
coste = format(vtaPrecio + (vtaPrecio * vtaImpuestos),"0.00 €")
```

12. Elimina la siguiente línea del bloque **With**… **End With**:

```
.NumberFormat = "0.00 €"
```

13. Reemplaza la línea **Range("B5").Formula = vtaPrecio * vtaImpuestos** por la siguiente:

```
Range("B5").Formula = format(vtaPrecio * vtaImpuestos,"0.00 €")
```

14. Ejecuta el procedimiento **Calculo_Costes_Modificado**.

Después de ejecutarlo, el texto que aparece en la celda A8 muestra el importe del producto con dos decimales.

Seguramente te estés preguntando por qué deberías molestarte en declarar las variables si VBA puede manejar muy bien las variables no declaradas. Los dos procedimientos anteriores son muy cortos, por lo que no tienes que preocuparte de cuántos bytes de memoria se consumirán cada vez que VBA utilice la variable **Variant**.

En los procedimientos cortos, sin embargo, no es la memoria lo que importa sino los errores que se cometen al escribir los nombres de las variables.
¿Qué ocurriría si al escribir el segundo procedimiento se omite la “o” en la variable **coste**?

Observa la Imagen 5.2, donde puedes comprobar que la celda B6 aparece vacía porque no encuentra la declaración de asignación de la variable **coste**.

Esto nos lleva a la siguiente sección, la cual explica cómo asegurarte de que este tipo de errores no se produzcan.

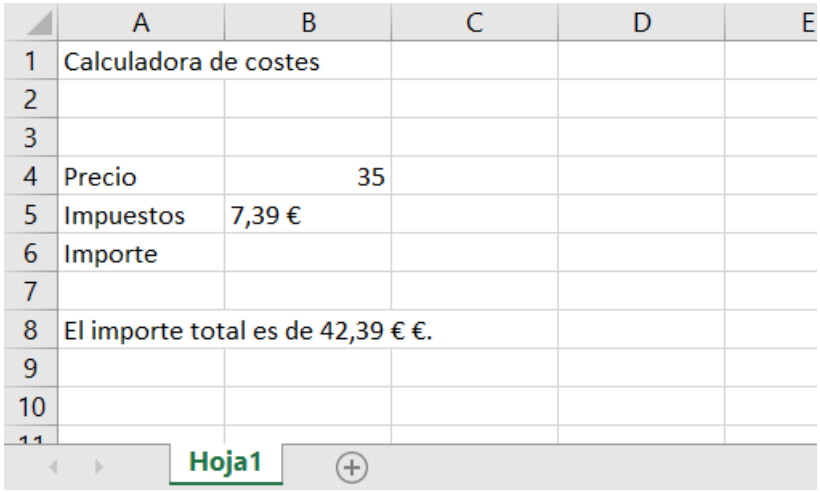


Imagen 5.2 Los nombres de las variables mal escritos producirán resultados incorrectos.

5.5 Cómo forzar la declaración de variables

VBA cuenta con la instrucción `Option Explicit` que te recuerda inmediatamente que declares explícitamente todas las variables. Esta declaración debe ser introducida en la parte superior de cada uno de los módulos. La declaración `Option Explicit` hará que VBA genere un mensaje de error cuando intentes ejecutar un procedimiento que contenga variables no declaradas.

Practiquemos esta instrucción:

1. Regresa a la ventana **Código** donde introdujiste el procedimiento `Calculo_costes`.
2. En la parte superior de la ventana escribe la declaración `Option Explicit` y presiona **Intro**.
3. Ejecuta el procedimiento `Calculo_costes`. VBA mostrará el mensaje de error "Error de compilación: Variable no definida".
4. Haz clic en **Aceptar** para cerrar el cuadro de diálogo. VBA resaltará el nombre de la variable `vtaPrecio`. Declara explícitamente la variable.
5. Vuelve a ejecutar el procedimiento. Cuando VBA se encuentra con otro nombre de variable no declarado, mostrará el mismo error.
6. Introduce las siguientes instrucciones debajo de la declaración del procedimiento (la instrucción `Sub`).

```
    Dim vtaPrecio As Currency
    Dim vtaImpuestos As Single
    Dim coste As Currency
    Dim strMensaje As String
```

Al finalizar la modificación del procedimiento, el código debería ser el siguiente:

```
Option Explicit
Sub Calculo_Costes()

    ' Declaración de variables
```

```
    Dim vtaPrecio As Currency
    Dim vtaImpuestos As Single
    Dim coste As Currency
    Dim strMensaje As String

    vtaPrecio = 35
    vtaImpuestos = 0.211

    Range("A1").Formula = "Calculadora de costes"
    Range("A4").Formula = "Precio"
    Range("B4").Formula = vtaPrecio
    Range("A5").Formula = "Impuestos"
    Range("A6").Formula = "Importe"
    Range("B5").Formula = vtaPrecio * vtaImpuestos
    coste = vtaPrecio + (vtaPrecio * vtaImpuestos)

    With Range("B6")
        .Formula = coste
        .NumberFormat = "0.00 €"
    End With

    strMensaje = "El importe total es de " & coste & " €."
    Range("A8").Formula = strMensaje
End Sub
```

5.6 El alcance de las variables

Las variables pueden tener diferentes zonas de influencia en un procedimiento VBA. El término "alcance" define si una variable está disponible para el mismo procedimiento, otro procedimiento, otros módulos y otros proyectos VBA.

Las variables tienen tres niveles de alcance:

- A nivel de procedimiento.
- A nivel de módulo.
- A nivel de proyecto.

5.6.1 A nivel de procedimiento (variables locales)

Desde este capítulo, ya sabes cómo declarar una variable con `Dim`. La posición de la palabra `Dim` en la hoja del módulo determina el alcance de una variable. Las variables declaradas con la palabra `Dim` dentro de un procedimiento VBA tienen un alcance a nivel de procedimiento.

Option Explicit en todos los módulos

Para incluir automáticamente la cláusula `Option Explicit` en todos los módulos nuevos que crees sigue los siguientes pasos:

1. Ve al menú **Herramientas – Opciones**.
2. En la pestaña **Editor**, asegúrate de que la casilla de verificación **Requerir declaración de variables** se encuentra activada. Si no es así, actívala.
3. Haz clic en **Aceptar** para cerrar el cuadro de diálogo.

A partir de ahora, todos los módulos nuevos se crearán con la cláusula `Option Explicit` en la parte de arriba. Si deseas exigir la declaración de variables en un módulo ya creado debes incluirla de forma manual.

Las variables a nivel de procedimiento se denominan también variables locales. Estas variables solo pueden utilizarse en el procedimiento en el que fueron declaradas. Las variables no declaradas siempre tienen un alcance a nivel de procedimiento. El nombre de una variable debe ser único dentro de su ámbito. Esto significa que no se pueden declarar dos variables con el mismo nombre en el mismo procedimiento. Por ejemplo, el procedimiento `Calculo_Costes` puede tener la variable `vtaImpuestos` y el procedimiento `Calculo_Costes_modificado`, en el mismo módulo, también puede tener su propia variable `vtaImpuestos`, siendo independientes entre sí.

5.6.2 A nivel de módulo

Las variables locales ayudan a ahorrar memoria del ordenador. Nada más finalizar el procedimiento, la variable termina y VBA devuelve el espacio de memoria ocupado por ella. Sin embargo, en programación, a menudo se desea que la variable esté disponible para otros procedimientos de VBA después de que el procedimiento en el que se declaró haya terminado de ejecutarse. Esta situación requiere que se modifique el alcance de la variable. En lugar de una variable a nivel de procedimiento, es necesario declararla a nivel de módulo. Para declarar una variable a nivel de módulo, debes colocar la declaración `Dim` en la parte superior de la hoja del módulo, sin que forme parte de ningún procedimiento. Habitualmente suelen escribirse justo debajo de la cláusula `Option Explicit`. Por ejemplo, para que la variable `vtaImpuestos` esté disponible para otros procedimientos, se debe declarar de esta forma:

```
Option Explicit
Dim vtaImpuestos As Single
Sub CalcCost()
    ...Instrucciones del procedimiento...
End Sub
```

En el ejemplo anterior, la palabra `Dim` se encuentra en la parte superior del módulo, debajo de `Option Explicit`. Antes de poder ver la variable en funcionamiento necesitas otro procedimiento que utilice la misma variable (`vtaImpuestos`). Sigue los pasos del ejemplo:

1. En la ventana **Código** del módulo **Variables**, corta la línea `Dim vtaImpuestos As Single` y pégala justo debajo de la cláusula `Option Explicit`.
2. En el mismo módulo donde se encuentra el procedimiento `Calculo_Costes`, introduce el código del procedimiento `Informe_Ventas`:

```
Sub Informe_Ventas()
    Dim vtaPrecio As Currency
    Dim coste As Currency
    vtaPrecio = 55.99
    coste = vtaPrecio + (vtaPrecio * vtaImpuestos)
    MsgBox vtaImpuestos
    MsgBox coste
End Sub
```

   En el procedimiento `Informe_Ventas` se declaran dos variables del tipo `Currency`. A la variable `vtaPrecio` se le asigna un valor de 55,99. Esta variable es diferente a la que creaste para el procedimiento `Calculo_Costes`.

   El procedimiento `Informe_Ventas` calcula el coste de compra. El coste incluye los impuestos, que se almacenan en la variable `vtaImpuestos`. Como el impuesto sobre las ventas es el mismo que el utilizado en el procedimiento `Calculo_Costes`, la variable `vtaImpuestos` se ha declarado a nivel de módulo.

3. Ejecuta el procedimiento `Informe_Ventas`.
   Como aún no se ha ejecutado el procedimiento `Calculo_Costes`, VBA no conoce el valor de la variable `vtaImpuestos`. Por esto el primer mensaje muestra un cero.
4. Ejecuta el procedimiento `Calculo_Costes`.
   Cuando finaliza la ejecución, el contenido de la variable `vtaImpuestos` es igual a 0,211. Si `vtaImpuestos` fuese una variable local, su contenido estaría vacío al finalizar el procedimiento `Calcular_Coste`.
   Cuando se ejecuta `Calcular_Costes`, VBA borra el contenido de todas las variables excepto `vtaImpuestos`, que se declaró a nivel de módulo.
5. Ejecuta de nuevo `Informe_Ventas`. Durante su ejecución VBA recupera el valor de la variable `vtaImpuestos` y lo utiliza en el cálculo.

¿Qué es una variable privada?

Cuando declaras variables a nivel de módulo, puedes utilizar la palabra clave `Private` en lugar de `Dim`. Por ejemplo:

```
Private vtaImpuestos As Single
```

Las variables privadas sólo están disponibles para los procedimientos que forman parte del módulo donde fueron declaradas. Siempre se declaran en la parte superior del módulo, después de la cláusula `Option Explicit`.

Mantener en privado las variables de proyecto

Para evitar que el contenido de una variable a nivel de proyecto sea referenciado fuera del proyecto, se puede utilizar la declaración **`Private Module`** en la parte superior de la hoja del módulo y por debajo de **`Option Explicit`**. Por ejemplo:

```
Option Explicit
Option Private Module
Dim vtaImpuestos As Single

Sub Calculo_Costes()
… Instrucciones del procedimiento …
End Sub
```

5.6.3 Variables a nivel de proyecto

Las variables que se declaran en un módulo con la palabra **`Public`** en vez de **`Dim`** tienen un alcance a nivel de proyecto. Esto significa que pueden ser utilizadas en cualquier módulo de VBA. Si quieres trabajar con una variable en todos los proyectos abiertos de VBA debes declararla de esta forma:

```
Option Explicit
Public vtaImpuestos As Single
Sub CalcCost()
   ...Instrucciones del procedimiento...
End Sub
```

Observa que la variable **`vtaImpuestos`** está declarada en la parte superior del módulo para que se pueda utilizar en cualquier otro procedimiento de VBA.

5.7 Vida de las variables

Además del alcance, las variables tienen una vida útil que determina el tiempo durante el que debe conservar su valor. Las variables a nivel de modulo y a nivel de proyecto conservan sus valores mientras el proyecto esté abierto. Sin embargo, VBA puede reiniciar estas variables si así lo requiere la lógica del programa. Las variables locales declaradas con la palabra **`Dim`** pierden sus valores una vez el procedimiento ha finalizado.

Las variables locales duran mientras el procedimiento se está ejecutando, y se reinician cada vez que el programa se ejecuta. VBA permite extender el tiempo de vida de una variable local si cambias la forma de declararla.

5.8 Determinar el tipo de dato de una variable

Es posible averiguar el tipo de datos de una variable utilizando una de las funciones que incorpora VBA.

La función **VarType** de vuelve un número entero que indica el tipo de la variable. La siguiente tabla muestra la sintaxis de la función **VarType** y los valores que devuelve.

```
VarType(nombre_variable)
```

vbEmpty	0	Vacío (sin iniciar)
vbNull	1	Nulo (datos no válidos)
vbInteger	2	Entero
vbLong	3	Entero largo
vbSingle	4	Precisión sencilla
vbDouble	5	Precisión doble
vbCurrency	6	Valor monetario
vbDate	7	Fecha
vbString	8	Texto
vbObject	9	Objeto
vbError	10	Valor de error
vbBoolean	11	Booleano
vbVariant	12	Variant
vbDataObject	13	Objeto de acceso a datos
vbDecimal	14	Valor decimal
vbByte	17	Byte
vbLongLong	20	Entero
vbUserDefinedType	36	Definida por el usuario
vbArray	8192	Matriz

Vamos a hacer una prueba en la ventana **Inmediato**.

1. Abre la ventana **Inmediato** desde el menú **Ver – Ventana Inmediato**.
2. Introduce el siguiente código que asigna valores a algunas variables:

```
Edad = 18
Fecha_Nacimiento = #21/04/1988#
```

```
Nombre= "Juan"
```

3. Ahora pregúntale a VBA qué tipo de datos contiene cada una de las variables:

   ```
   ?VarType(Edad)
   ```

   Cuando presionas **Intro**, se muestra el número 2, que corresponde al tipo de datos `Integer`.
   Si introduces `?VarType(Fecha_Nacimiento)`, VBA devuelve el número 7 (fecha).

6 Las constantes

El contenido de una variable puede cambiar mientras se ejecuta el procedimiento. Si el procedimiento necesita referirse a valores que se van a mantener fijos, entonces debes utilizar constantes.

Una constante es como una variable, con la diferencia de que siempre tiene el mismo valor. VBA requiere que declares las constantes antes de usarlas. Las constantes se declaran utilizando la palabra `Const`, como en estos casos:

```
Const Nombre_Cuadro = "Introducción de datos"
Const vtaImpuestos = 0.211
Const Vel_luz = 30000
```

Las constantes, al igual que las variables, tienen un alcance. Para que una constante esté disponible dentro de un procedimiento único, declárala a nivel de procedimiento justo debajo del nombre del procedimiento:

```
Sub Aniversario()
    Const Dia = 2
    MsgBox Dia
End Sub
```

Si deseas utilizar una constante en todos los procedimientos de un módulo, usa la palabra clave `Private` delante de la declaración de la constante. Por ejemplo:

```
Private Const Unidad = "C:\"
```

La constante declarada como `Private` tiene que situarse en la parte superior del módulo, justo antes del primer procedimiento. Si quieres que una constante esté disponible en todos los módulos del libro, utiliza la palabra `Public` delante de `Const`, por ejemplo:

```
Public Const Num_elementos = 255
```

Las constantes públicas tienen que ser declaradas en la parte superior del módulo antes del primer procedimiento. Cuando se declara una constante, se puede utilizar cualquiera de los siguientes tipos de datos: `Boolean`, `Byte`, `Integer`, `Long`, `Single`, `Double`, `Date`, `String` o `Variant`.

Al igual que las variables, es posible declarar varias constantes si están separadas por comas:

```
 Const Num_codigo As Integer = 34, Fecha_comienzo As Date =
#1/31/2020#
```

El uso de constantes hace que los procedimientos de VBA sean más legibles y fáciles de mantener. Por ejemplo, si te refieres a un determinado valor en tu procedimiento, utiliza una constante en lugar del valor. De esta forma, si el valor se modifica (por ejemplo, si suben los impuestos), puedes modificar simplemente el valor en la declaración de la constante en lugar de examinar todo el código para ver en qué parte lo has introducido.

6.1 Constantes predefinidas

Tanto Excel como VBA cuentan con una larga lista de constantes predefinidas que no necesitan ser declaradas. Estas constantes se pueden consultar utilizando la ventana del **Examinador de objetos**:

1. Abre el **Examinador de objetos** desde el menú **Ver – Examinador de objetos** (o presionando **F2**).
2. En el desplegable de la parte superior, selecciona Excel.
3. Introduce "Constants" como texto de búsqueda y pulsa **Intro** o haz clic en el botón **Buscar**. VBA muestra el resultado de la búsqueda en el área inferior.
4. Desplázate hacia abajo en el cuadro **Clases** y selecciona **Constantes** como se muestra en la Imagen 6.1. En el lado derecho del examinador se puede ver una lista de todas las constantes predefinidas que están disponibles en la biblioteca de objetos de Excel. Observa que los nombres de todas las constantes comienzan con el prefijo "xl".
5. Para buscar constantes de VBA, selecciona VBA en el cuadro desplegable de la parte superior del examinador. Observa también que las constantes de VBA comienzan con el prefijo "vb".

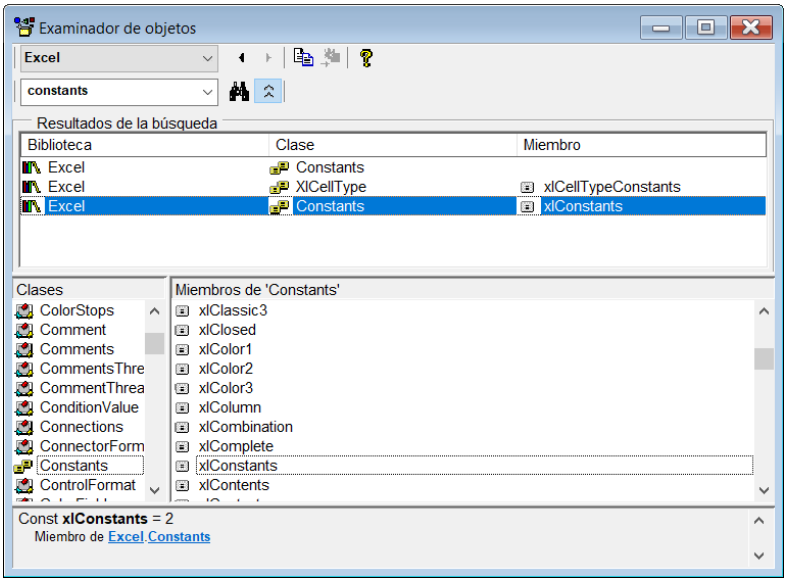


Imagen 6.1 Utilizando el Examinador de objetos para encontrar las constantes de Excel.

7 Conversión de tipos de datos

Aunque VBA es capaz de gestionar conversiones de unos tipos de datos a otros, también cuenta con varias funciones directas de conversión de datos que te permiten convertir un tipo

de dato en otro. Estas funciones deben utilizarse cuando quieres que el resultado de una operación se exprese en un tipo de dato en concreto en lugar de loestablecido por defecto. Por ejemplo, en lugar de mostrar el resultado de un cálculo como número entero, quizá se trate de un tipo de moneda y así quieras mostrarlo:

```
Sub Cambiar_dinero()
    'Se declaran variables de dos tipos diferentes
    Dim Cantidad As Single
    Dim Dinero As Currency
    Cantidad = 345.34
    Dinero = CCur(Cantidad)
    Debug.Print "La cantidad es = " & Dinero & " €."
End Sub
```

Al utilizar la función `CCur`, se reconocen las opciones de moneda de la configuración de tu ordenador. Igual ocurre con la función `CDate`. Al utilizarla, puedes asegurar que la fecha se formatee de acuerdo con la configuración de idioma de tu ordenador. Utiliza la función `IsDate` para averiguar si un valor devuelto se puede convertir en una fecha u hora.

```
Sub Convertir_en_fecha()
    'Suponemos que la celda A1 contiene la fecha 01/01/2020
    Dim MiDato As String
    Dim MiOtroDato As Date
    MiDato = Sheet2.Range("A1").Value
    If IsDate(MiDato) Then
    MiOtroDato = CDate(MiDato)
    End If
    Debug.Print MiOtroDato
End Sub
```

En los casos en los que necesites redondear el valor al número par más cercano encontrarás muy útiles las funciones `CInt` y `CLng`, como se demuestra en el siguiente procedimiento:

```
Sub Mostrar_entero()
    'Se declaran variables de dos tipos diferentes
    Dim Cantidad As Single
    Dim CantidadEntera As Integer
    Cantidad = 345.64
    CantidadEntera = CInt(Cantidad)
    Debug.Print "Cantidad original = " & Cantidad
    Debug.Print "Nueva cantidad = " & CantidadEntera
End Sub
```

Como puedes ver en el código de los procedimientos anteriores, la sintaxis de las funciones de conversión es la siguiente:

```
Función_conversión (Nombre_variable)
```

CBool	Booleano
CByte	Byte
CCur	Moneda
CDate	Fecha
CDbl	Doble
CDec	Decimal
CInt	Entero
CLng	Largo
CLngLng	Largo
CLngPtr	LargoPtr
CSng	Sencillo
CStr	Texto
CVar	Variant

Practiquemos con ello:

1. Inserta un nuevo módulo desde el menú **Insertar – Módulo** en el libro "Capítulo 3 – Variables.xlsm".
2. Utiliza la ventana **Propiedades** para cambiarle el nombre al módulo. Llámalo "ConversionTipos".
3. Introduce el código de los procedimientos de esta sección: `Cambiar_dinero`, `Convertir_en_fecha` y `Mostrar_entero`.
4. A continuación, crea una nueva hoja en el libro e introduce la fecha 01/01/2020 en la celda A1.
5. Ejecuta todos los procedimientos y comprueba los resultados en la ventana **Inmediato**.

8 Las variables Static

Una variable declarada con la palabra `Static` es un tipo especial de variable local.

Las variables estáticas se declaran a nivel de procedimiento. A diferencia de las variables locales declaradas con `Dim`, las estáticas no pierden su contenido cuando el código de ejecución no está en el procedimiento. Por ejemplo, cuando un procedimiento VBA con una variable estática llama a otro procedimiento después de que VBA ejecute las declaraciones del

procedimiento llamado y vuelva al procedimiento inicial, la variable estática sigue conservando el valor original. El procedimiento `Coste_de_compra` que se muestra en la siguiente práctica, demuestra el uso de la variable estática llamada `CompraAcumulada`.

1. Escribe o copia el siguiente código en el módulo **Variables** del archivo con el que estás trabajando.

```
Sub CostOfPurchase()
    ' Se declaran las variables
    Static CompraAcumulada
    Dim NuevaCompra As String
    Dim CosteCompra As Single
    NuevaCompra = InputBox("Introduce el coste de una compra:")
    CosteCompra = CSng(NuevaCompra)
    CompraAcumulada = CompraAcumulada + CosteCompra
    ' Muestra los resultados
    MsgBox "El coste de la compra es: " & NuevaCompra
    MsgBox "El resultado acumulado es: " & CompraAcumulada
End Sub
```

   El procedimiento anterior comienza con la declaración de una variable estática llamada `CompraAcumulada` y otras dos variables locales: `NuevaCompra` y `CosteCompra`. La función `InputBox` muestra un cuadro de diálogo y espera a que el usuario introduzca un valor. Cuando lo haces y presionas **Aceptar**, VBA asigna este valor a la variable `NuevaCompra`.

   La función `InputBox` se trabaja en detalle en el Capítulo 4.

   Como el resultado de la función es siempre una cadena de texto, la variable `NuevaCompra`, se declara con el tipo de datos `String`. Como no es posible utilizar cadenas en cálculos matemáticos, en la siguiente instrucción se utiliza una función de conversión de tipo (`CSng`) para introducir el valor del texto en una variable numérica del tipo `Single`. La función `Csng` requiere un argumento: el valor que se quiere traducir.

   Para conocer más detalles sobre la función `CSng`, sitúa el cursor en cualquier lugar dentro de la palabra `CSng` y presiona **F1**.

   El número obtenido como resultado de la función `Csng` se almacena en la variable `CosteCompra`.

   La siguiente instrucción, `CompraAcumulada = CompraAcumulada + CosteCompra` suma al valor de `CompraAcumulada`, el nuevo valor introducido por la función `InputBox`.

2. Sitúa el cursor en cualquier parte del procedimiento `Coste_de_compra` y presiona **F5**. Cuando aparezca el cuadro de diálogo, introduce un número. Por ejemplo,

introduce 100 y haz clic en **Aceptar** o presiona **Intro**. VBA muestra el mensaje "El coste de la compra es: 100". Haz clic en **Aceptar** y VBA mostrará un segundo cuadro que indica que "El resultado acumulado es: 100".

3. Cuando se ejecuta este procedimiento por primera vez, el contenido de `CompraAcumulada` es la misma que el contenido de la variable `CosteCompra`.
4. Repite el mismo procedimiento. Cuando aparezca el cuadro de diálogo, introduce 50 y haz clic en **Aceptar** o pulsa **Intro**. VBA muestra el mensaje "El coste de la compra es: 50". Haz clic en **Aceptar** y VBA mostrará el segundo cuadro de mensaje "El resultado acumulado es: 150".
5. Cuando se ejecuta el procedimiento por segunda vez, el valor de la variable estática se incrementa con el nuevo valor introducido en el cuadro de diálogo. Puedes ejecutar el procedimiento tantas veces como quieras. La variable `CompraAcumulada` mantendrá el total mientras el proyecto esté abierto.

9 Las variables de objeto

Las variables que has aprendido en las secciones anteriores se utilizan para almacenar datos. En esto se basa el uso de variables "normales".

Pero además de trabajar con este tipo de variables, existen variables especiales que se refieren a los objetos de Excel. Se denominan "variables de objeto".

En el Capítulo 2, trabajaste con varios objetos en la ventana **Inmediato**. Ahora aprenderás cómo puedes representar un objeto mediante esta variable.

Las variables de objeto no almacenan datos, sin embargo, dicen dónde se encuentran los datos. Por ejemplo, con una variable de objeto, puedes decirle a VBA que los datos están en la celda E10 de una determinada hoja. Cuando se escriben procedimientos de VBA, a menudo es necesario escribir instrucciones largas, como por ejemplo:

```
Worksheets("Hoja2").Range(Cells(1, 1), Cells(10, 5).Select
```

En lugar de usar referencias al objeto tan largas, puedes declarar una variable de objeto que le dirá a VBA dónde se encuentran los datos. Las variables de objeto se declaran de forma similar a las variables que ya conoces. La única diferencia es que después de la palabra `As`, se introduce la palabra `Object` como tipo de datos. Por ejemplo:

```
Dim MiRango As Object
```

Esta instrucción declara la variable de objeto `MiRango`.

Bueno, no es suficiente con declarar la variable de objeto. También hay que asignarle un valor específico antes de poder usarla en tus procedimientos.

Para asignar un valor a una variable de objeto, se utiliza la palabra `Set`, seguida del nombre de la variable y el signo igual (=). A continuación, se escribe el valor de la variable. Por ejemplo:

```
Set MiRango = Worksheets("Hoja2").Range(Cells(1, 1), _
Cells(10, 5))
```

Esta declaración asigna un valor a la variable **MiRango**. Este valor se refiere a las celdas A1:E10 de la hoja2. Si omites la palabra **Set**, VBA responderá con el mensaje de error "Se ha producido el error 91 en tiempo de ejecución: Variable de objeto o bloque With no establecido".

Practiquemos con un ejemplo:

1. En la ventana **Código** del módulo **Variables**, introduce las siguientes instrucciones:

```
Sub VariablesObjeto()
    Dim MiRango As Object
    Sheets.Add
    Set MiRango = Worksheets("Hoja2").Range(Cells(1, 1), _
    Cells(10, 5))
    MiRango.BorderAround Weight:=xlMedium
    With MiRango.Interior
    .ColorIndex = 6
    .Pattern = xlSolid
    End With
    Set MiRango = Worksheets("Hoja2").Range(Cells(12, 5), _
    Cells(12, 10))
    MiRango.Value = 54
    Debug.Print IsObject(MiRango)
End Sub
```

Examinemos el código línea por línea. El procedimiento comienza con la declaración de la variable de objeto **MiRango** en el rango A1:E10 de la Hoja2.

A partir de ahora, cada vez que quieras referirte a este rango, en lugar de usar la dirección completa del objeto, utilizarás el nombre de la variable de objeto. El objetivo de este procedimiento es crear un borde alrededor del rango A1:E10. En lugar de escribir una instrucción larga:

```
Worksheets("Hoja2").Range(Cells(1, 1), _
Cells(10, 5)).BorderAround Weight:=xlMedium
```

Puedes utilizar otra bastante más reducida:

```
MiRango.BorderAround Weight:=xlMedium
```

En el procedimiento anterior, la siguiente serie de declaraciones cambia el color del rango de celdas seleccionadas (A1:E10). Una vez más, no es necesario escribir la instrucción larga para hacer referencia al objeto a manipular. En lugar de ello, utiliza la variable **MiRango**. La siguiente instrucción asigna una nueva referencia a la variable de objeto **MiRango**. VBA olvida la referencia anterior y, la próxima vez que uses **MiRango**, se hará referencia al nuevo rango (E12:J12).

Tras introducir el número 54 en el rango nuevo, el procedimiento te muestra cómo puedes asegurarte de que una variable específica es del tipo Objeto. La instrucción

`Debug.Print IsObject(MiRango)` mostrará "Verdadero" en la ventana **Inmediato** si `MiRango` es una variable de objeto. `IsObject` es una función de VBA que indica si un valor específico es una variable de objeto.

2. Sitúa el cursor en cualquier punto del procedimiento y presiona **F5** para ejecutarlo.

Ventajas de utilizar variables de objeto

- Se pueden usar en lugar del objeto real.
- Los valores son más cortos y fáciles de recordar que los valores reales a los que apuntan.
- Puedes cambiar su valor mientras el procedimiento está ejecutándose.

9.1 Uso de variables de objetos específicos

Una variable de objeto puede referirse a cualquier tipo de objeto. Dado que VBA tiene muchos tipos de objetos, es una buena idea crear variables de objeto que se refieran a un objeto específico para que tus macros sean más legibles y rápidas.

Por ejemplo, en el procedimiento anterior, en lugar de declarar la variable como `Object`, puedes declararla como `Range`.

```
Dim MiRango As Range
```

Si deseas hacer referencia a una hoja específica, puedes declarar la variable como objeto `Worksheets`.

```
Dim MiHoja As Worksheet
Set MiHoja = Worksheets("Marketing")
```

Cuando la variable de objeto ya no es necesaria, puedes asignarle el valor `Nothing`. Esto libera la memoria y los recursos del sistema.

```
Set MiHoja = Nothing
```

10 Resumen

En este capítulo hemos visto nuevos conceptos de VBA, como los tipos de datos, las variables y las constantes. Has aprendido a declarar varios tipos de variables y a definir sus tipos. También has visto la diferencia entre una variable y una constante.

Ahora que sabes qué son las variables y cómo usarlas, puedes crear procedimientos VBA que te permitan manipular datos de una forma más avanzada que en los capítulos anteriores.

En el siguiente capítulo, ampliarás tus conocimientos de VBA aprendiendo a escribir procedimientos de funciones personalizadas. Además, aprenderás sobre las funciones integradas que permitirán que tus procedimientos VBA interactúen con los usuarios.

Capítulo 4
Los procedimientos Function

En capítulos anteriores aprendiste que un procedimiento es un grupo de instrucciones que, al ejecutarse, permite realizar tareas específicas. En este libro te familiarizarás con los siguientes tipos de procedimientos VBA:

- **Subrutinas** (`Sub`): Realizan algunas tareas útiles sin devolver ningún valor. Comienzan con la palabra `Sub` y finalizan con `End Sub`. Pueden ser grabadas con la grabadora de macros o escritas a mano en el editor de VBA. En el capítulo 1 aprendiste varias formas de ejecutar este tipo de procedimiento.
- **Funciones** (`Function`): Realizan tareas específicas que devuelven valores. Comienzan con la palabra `Function` y finalizan con `End Function`. Pueden ser ejecutadas desde una subrutina o acceder a ellas desde una hoja, como cualquier función de Excel.
- `Property`: Se utilizan con objetos personalizados. Se usan para establecer y obtener el valor de las propiedades de un objeto o establecer una referencia a un objeto. Aprenderás a crear objetos personalizados y a utilizar los procedimientos `Property` en el Capítulo 8.

En este capítulo aprenderás a crear y ejecutar funciones personalizadas. Además, descubrirás cómo se utilizan las variables para pasar valores a procedimientos `Sub` y `Function`. Más adelante en el capítulo, abordaremos las dos funciones integradas más útiles: `MsgBox` e `InputBox`.

1 Los procedimientos Function

Con los cientos de funciones de hoja de Excel, puedes realizar una gran variedad de cálculos. Sin embargo, habrá ocasiones en las que necesitarás un cálculo personalizado. Con VBA podrás realizar este cálculo rápidamente tú mismo creando un procedimiento de función. Puedes construir cualquier función que no proporcione Excel. Entre las razones para crear funciones personalizadas con VBA se encuentran las siguientes:

- Analizar datos y realizar cálculos.
- Modificar los datos, estructurarlos y mostrar la información.

- Tomar una decisión específica según los datos suministrados o calculados.

1.1 Cómo crear un procedimiento Function

Al igual que las funciones de Excel, los procedimientos `Function` realizan cálculos y devuelven resultados. La mejor forma de aprender es creando uno, así que empecemos. Después de configurar un nuevo proyecto VBA, crearás un procedimiento `Function` simple que sume dos valores.

1. Crea un nuevo libro de Excel y guárdalo con el nombre "**Capítulo 4 – Funciones.xlsm**".
2. Dirígete al editor de VBA y selecciona el nombre del proyecto que acabas de crear.
3. En la ventana **Propiedades** cambia el nombre del proyecto. Llámalo `ProyectoFunction`.
4. Selecciona el proyecto en el **Explorador de proyectos** e inserta un módulo.
5. En la ventana **Propiedades**, cámbiale el nombre a `Ejemplo1`.
6. En el **Explorador de proyectos**, selecciona **Ejemplo1** y haz clic en cualquier parte de la ventana **Código**. Haz clic en **Insertar – Procedimiento**. Aparecerá el cuadro de diálogo **Agregar procedimiento**.
7. Introduce los siguientes datos en el cuadro:

   Nombre: **Suma2**

   Tipo: **Función**

   Ámbito: **Público**

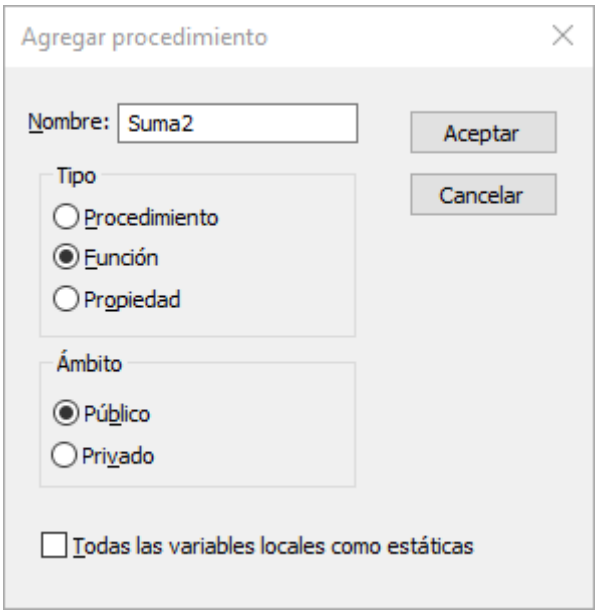


Imagen 1.1 Cuando se utiliza el cuadro de diálogo Agregar procedimiento, VBA crea automáticamente el tipo de procedimiento que se elige.

8. Haz clic en **Aceptar** para cerrar el cuadro de diálogo. VBA introduce un procedimiento `Function` vacío que tiene el siguiente aspecto.

```
Public Function Suma2()
```

```
End Function
```

9. Modifica la declaración de la siguiente forma:

```
Public Function Suma2(m, n)


End Function
```

El objetivo de esta función es sumar dos valores. En lugar de sumar valores introducidos a mano, puedes hacer que la función sea más flexible proporcionándole los argumentos en forma de variables. Al hacer esto, podrás sumar los dos números que especifiques.
Cada una de las variables `(m, n)` representa un valor. Al ejecutar la función, debes suministrar los valores para cada una.

10. Escribe la siguiente instrucción entre las líneas `Function` y `End Function`:

```
Suma2 = m + n
```

Esta instrucción le dice a VBA que sume los valores guardados en las variables `m` y `n` y devuelva el resultado a la función `Suma2`. Para especificar el valor que quieres que devuelva la función, escribe el nombre de la función seguido del signo igual y el valor que desees que devuelva. La instrucción anterior suma las variables `m` y `n` y almacena el resultado en la función.
Este es el resultado final:

```
Public Function Suma2(m, n)
    Suma2 = m + n
End Function
```

La primera línea declara el nombre de la función. La palabra clave `Public` indica que la función es accesible a todos los procedimientos de todos los módulos. La palabra `Public` es opcional.
Observa que el nombre de la función va seguido de un par de paréntesis. Entre ellos se enumeran los datos que la función utilizará en el cálculo. Todos los procedimientos `Function` finalizan con la declaración `End Function`.

2 Varias formas de ejecutar procedimientos Function

A diferencia de un procedimiento `Sub`, uno `Function` puede ejecutarse solo de dos formas: utilizándolo en una fórmula de una hoja o llamarlo desde otro procedimiento. A continuación, aprenderás técnicas especiales para ejecutar estas funciones.

Los nombres de los procedimientos Function

Los nombres de las funciones deben indicar el objetivo que desempeñará la función y ajustarse a las reglas de nombres de variables del capítulo anterior.

El ámbito de las funciones

En el capítulo anterior, aprendiste que el alcance de la variable determina los módulos y procedimientos en los que se puede utilizar. Al igual que las variables, los procedimientos de VBA tienen un alcance. El alcance de un procedimiento determina si puede ser llamado por procedimientos en otros módulos. Por defecto, todos los procedimientos VBA son públicos. Esto significa que pueden ser llamados por otros procedimientos en cualquier módulo. Como los procedimientos son públicos por defecto, puedes saltarte (si lo deseas) la palabra `Public`. Y si la sustituyes por la palabra `Private`, el procedimiento estará disponible sólo para otros procedimientos en el mismo módulo, no para procedimientos en otros módulos.

2.1 Desde una hoja

Un procedimiento de función personalizada es como una función integrada de Excel. Si no conoces el nombre exacto de la función o sus argumentos, puedes utilizar el botón **Fórmula** para ayudarte a introducir la función requerida en la hoja como puedes ver en la **Imagen 2.1**.

1. Desde la ventana de Excel, selecciona cualquier celda.
2. Haz clic en el botón **Insertar función** (fx) que se encuentra a la izquierda de la barra de fórmulas. Excel muestra el cuadro de diálogo **Insertar función**. La parte inferior de la ventana muestra una lista alfabética de la categoría seleccionada.
3. En el cuadro desplegable de las categorías, selecciona **Definido por el usuario**. En el cuadro de funciones aparecerá la función `Suma2` que acabas de crear.
   Cuando seleccionas el nombre de la función, la parte inferior del cuadro de diálogo muestra la sintaxis de la función:

   ```
   Suma2 (m, n)
   ```

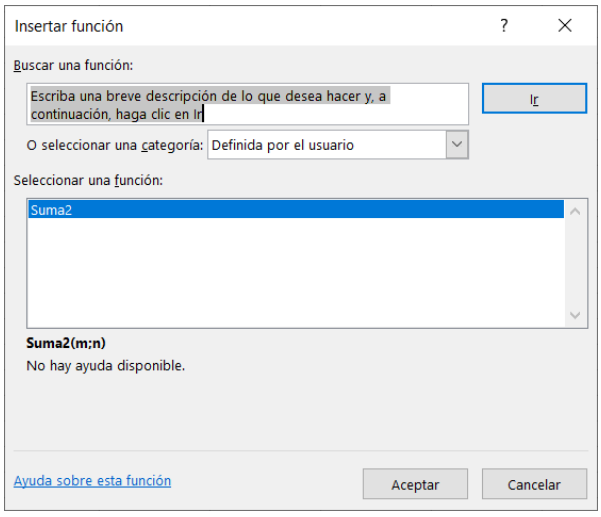


Imagen 2.1 Los procedimientos de la función personalizada se listan en la categoría Definida por el usuario del cuadro de diálogo Insertar función. También aparecen en la lista cuando se selecciona la categoría Todo del desplegable.

4. Haz clic en **Aceptar** para empezar a escribir la fórmula. Aparecerá el cuadro **Argumentos de función**, como se muestra en la **Imagen 2.2**. Este cuadro muestra el nombre de la función y cada uno de sus argumentos `(m y n)`.

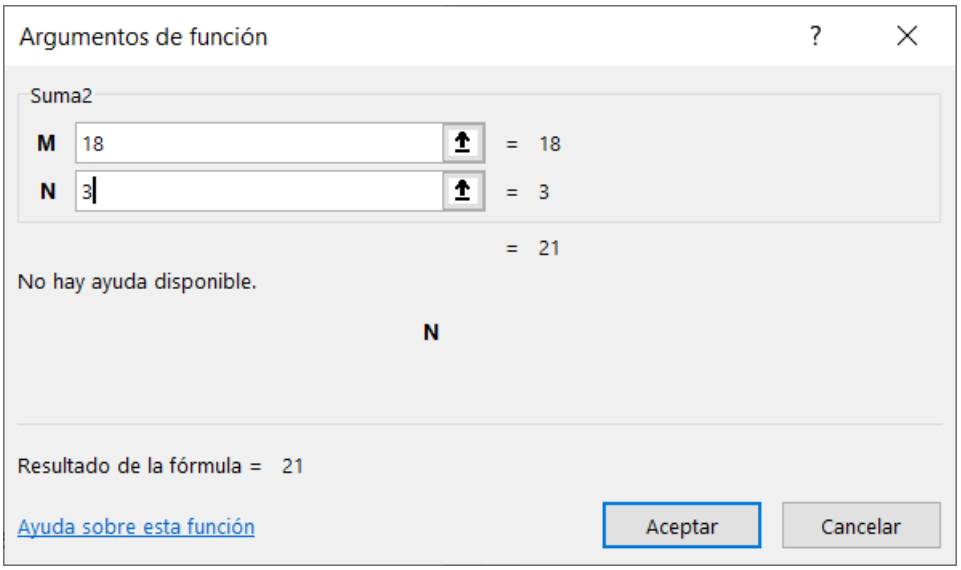


Imagen 2.2 El cuadro Argumentos de función es útil para introducir cualquier función en la hoja, ya sea predefinida o personalizada.

5. Introduce los valores de los argumentos como se muestra en la **Imagen 2.2.** o introduce tus propios valores. Mientras los escribes, Excel los muestra acompañados del resultado actual de la función. Como se necesitan ambos argumentos **(m y n)**, la función devolverá un error si falta alguno de ellos.
6. Haz clic en **Aceptar** para salir del cuadro **Argumentos de función**. Excel introduce la función **Suma2** en la celda seleccionada y muestra el resultado.
7. Para editar la función, selecciona la celda y haz clic en el botón **Insertar función** (fx) para acceder de nuevo al cuadro **Argumentos de función**. Introduce otros valores para los argumentos **m** y **n** y haz clic en **Aceptar**.

Nota

Para editar directamente los valores de los argumentos en la celda, haz doble clic en ella y haz los cambios necesarios.

La función también está configurada para admitir valores que se encuentran en otras celdas. Para introducir valores de este tipo simplemente sitúate en uno de los recuadros de los argumentos que aparecen dentro del cuadro **Argumentos de función** y selecciona la celda que contenga el valor a introducir. Cuando hagas clic en **Aceptar**, Excel calculará el resultado igual que con cualquier otra función predefinida.

2.2 Desde otro procedimiento VBA

Para ejecutar una función personalizada desde un procedimiento, escribe el procedimiento y llama a la función cuando la necesites. El siguiente procedimiento llama a la función **Suma2** y muestra el resultado del cálculo en la ventana **Inmediato**.

1. En el módulo donde introdujiste la función **Suma2**, escribe el siguiente procedimiento:

```
Sub EjecutarSuma2()
Dim m As Single, n As Single
m = 37
n = 3459.77
Debug.Print Suma2(m, n)
MsgBox "Abre la ventana Inmediato para ver el resultado."
End Sub
```

   Fíjate en cómo el procedimiento anterior utiliza la instrucción **Dim** para declarar las variables **m** y **n**. Esas variables se usarán para alimentar los datos e la función.

   Las siguientes dos instrucciones asignan los valores a esas variables. Luego VBA llama a la función **Suma2** y le pasa los valores almacenados en las variables **m** y **n**. Cuando se ejecuta la instrucción **Suma2 = m + n**, VBA regresa al procedimiento **EjecutarSuma2** y utiliza la instrucción **Debug.Print** para mostrar el resultado de la función en la ventana **Inmediato**. Para finalizar, **MsgBox** informa al usuario de dónde se encuentra el resultado. Puedes encontrar más información sobre el uso de **MsgBox** más adelante en este capítulo.

2. Coloca el puntero del ratón en cualquier lugar del procedimiento **EjecutarSuma2** y presiona **F5** para ejecutarlo.

3 Dónde almacenar las funciones personalizadas

La función personalizada solo está disponible mientras el libro donde está almacenada se encuentre abierto. Si cierras el libro, la función ya no está disponible. Para asegurarte de que las funciones personalizadas estén siempre disponibles cada vez que abras Excel, puedes realizar una de las siguientes acciones:

- Almacenarlas en el libro macros personal.
- Guardarla en el propio libro.
- Configurar una referencia al libro que contenga las funciones personalizadas.

4 Paso de argumentos al procedimiento Function

Los procedimientos (tanto los **Sub** como los **Function**) a menudo requieren argumentos. Los argumentos son uno o más valores necesarios para que el procedimiento haga algo. Los argumentos se introducen entre paréntesis. Cuando una función tiene varios argumentos, éstos se separan por comas (o punto y coma en la hoja de cálculo en versiones de Excel en español).

Cómo probar una función personalizada

Tras escribir un procedimiento `Function`, puedes probarlo rápidamente en la ventana **Inmediato**. Para mostrar el valor de una función, abre la ventana **Inmediato** y escribe un signo de interrogación (?) seguido del nombre de la función. Recuerda escribir entre paréntesis los argumentos de la función.

Por ejemplo, escribe `?Suma2(54,23)` y presiona **Intro**. El procedimiento se ejecuta utilizando los valores que introdujiste para los argumentos m y n. El resultado de la función aparece una línea más abajo: 77.

Si has usado Excel durante algún tiempo, ya sabes que las funciones predefinidas de Excel pueden devolver diferentes resultados según los valores que proporciones. Por ejemplo, si las celdas **A4** y **A5** contienen los números 5 y 10 respectivamente, la función **SUMA(A4;A5)** devolverá 15. Al igual que se puede asignar cualquier valor a las funciones de hoja, también puedes asignar cualquier valor a los procedimientos `Function` de VBA.

Veamos cómo puedes pasar algunos valores de un procedimiento `Sub` a la función **SUMA**. Escribiremos un procedimiento que recoja el nombre y los apellidos de un usuario. A continuación, llamaremos a la función `Suma2` para obtener la suma de los caracteres del nombre y los apellidos.

1. Escribe el procedimiento siguiente (`NumeroCaracteres`) en el mismo módulo (Ejemplo1) donde introdujiste `Suma2`.

```
Sub NumeroCaracteres()
    Dim f As Integer
    Dim l As Integer
    f = Len(InputBox("Introduce el nombre:"))
    l = Len(InputBox("Introduce el apellido:"))
    MsgBox Suma2(f, l)
End Sub
```

2. Coloca el puntero del ratón dentro del código del procedimiento `NumeroCaracteres` y presiona **F5**. VBA muestra un cuadro solicitando el nombre. Este cuadro lo genera la función `InputBox("Introduce el nombre:")`. Tienes más información sobre `InputBox` más adelante en este capítulo.
3. Introduce un nombre cualquiera y presiona **Intro** o haz clic en **Aceptar**. VBA toma el texto que has introducido y lo utiliza como argumento para la función `Len`. `Len` calcula el número de caracteres en la cadena de texto especificada. VBA coloca el resultado de la función `Len` en la variable `f`. Después se muestra el siguiente cuadro, esta vez solicitando el apellido.
4. Introduce cualquier apellido y pulsa **Intro** o haz clic en **Aceptar**.
   VBA pasa el apellido a la función `Len` para obtener el número de caracteres. Este número se almacena en la variable `l`. Una vez almacenado, VBA suma los dos números

y devuelve el resultado de la función **Suma2** como argumento a la función **MsgBox**. Ahora aparece un mensaje en la pantalla mostrando el número total de caracteres. Haz clic en **Aceptar** para cerrar el cuadro de mensaje. Puedes ejecutar el procedimiento **NumeroCaracteres** tantas veces como quieras, suministrando diferentes nombres y apellidos. Para pasar un valor específico de una función a una subrutina, asígnale el valor al nombre de la función. Por ejemplo, la siguiente función (**NumeroDias**), pasa el valor 7 al procedimiento **DiasSemana**.

```
Function NumeroDias()
NumeroDias = 7
End Function

Sub DiasSemana()
MsgBox "Hay " & NumeroDias & " días en una semana."
End Sub
```

4.1 Especificar el tipo de argumento

En la sección anterior, aprendiste que las funciones realizan cálculos utilizando los datos recibidos a través de los argumentos. Cuando declaras un procedimiento **Function**, enumeras los nombres de los argumentos entre paréntesis. Los nombres de los argumentos son como variables. Cada uno de ellos se refiere a cualquier valor que se suministre en el momento de llamar a la función. Cuando un procedimiento **Sub** llama a un procedimiento **Function**, le pasa los argumentos necesarios como variables. Una vez que la función realiza su acción, el resultado se asigna al nombre de la función. Observa que el nombre del procedimiento **Function** se utiliza como si fuera una variable.

Como las variables, las funciones pueden devolver diferentes tipos de datos. Pueden ser **String**, **Integer**, **Long**, etc. Para especificar el tipo de datos del resultado de la función, añade la palabra clave **As** y el nombre del tipo de datos deseado al final de la declaración de la función. Por ejemplo:

```
Function Multiplica2(num1, num2) As Integer
```

Veamos un ejemplo de una función que devuelve un número entero, aunque los argumentos que se le pasan se declaran como datos **Single** en el procedimiento **Sub** de llamada.

1. Añade un nuevo módulo al proyecto **ProyectoFunction** y cámbiale el nombre por Ejemplo2.
2. Activa el módulo e introduce el siguiente procedimiento:

```
Sub CuantoEs()
    Dim num1 As Single
    Dim num2 As Single
    Dim resultado As Single
    num1 = 45.33
    num2 = 19.24
    resultado = Multiplica2(num1, num2)
```

```
    MsgBox resultado
End Sub
```

3. Introduce el procedimiento **Multplica2** debajo del anterior.

```
Function Multiplica2(num1, num2) As Integer
    Multiplica2 = num1 * num2
End Function
```

Como los valores almacenados en las variables **num1** y **num2** no son números enteros, es posible que quieras asignar el tipo de datos **Integer** al resultado de la función para asegurarte de que devuelve un número entero. Si no asignas el tipo de datos al resultado de la función **Multiplica2**, el procedimiento **CuantoEs** mostrará el tipo de datos especificados en la línea de declaración de la variable de resultado. En lugar de mostrar 872, el resultado de la multiplicación será 872,1492.

4. Ejecuta el procedimiento **CuantoEs**.
   ¿Qué ocurre si pasamos valores diferentes cada vez que se ejecute el procedimiento? En lugar de introducir los valores manualmente, puedes utilizar la función **InputBox** para pedir al usuario los valores en tiempo de ejecución. Por ejemplo:

```
num1 = InputBox("Introduce un número:")
```

4.2 Pasar argumentos por referencia y por valor

En algunos procedimientos, cuando usas argumentos como variables, VBA puede cambiar de repente el valor. Para asegurarte de que el procedimiento de la función llamada no altera el valor de los argumentos, debes preceder el nombre del argumento en la línea de declaración de la función con la palabra clave **ByVal**. Veamos el siguiente ejemplo:

1. Agrega un nuevo módulo al proyecto **ProyectoFunction** y llámalo Ejemplo3.
2. Actívalo e introduce el siguiente procedimiento:

```
Sub TresNumeros()
    Dim num1 As Integer, num2 As Integer, num3 As Integer
    num1 = 10
    num2 = 20
    num3 = 30
    MsgBox Promedio2(num1, num2, num3)
    MsgBox num1
    MsgBox num2
End Sub
Function Promedio2(ByVal num1, ByVal num2, ByVal num3)
    num1 = num1 + 1
    Promedio2 = (num1 + num2 + num3) / 3
End Function
```

Para evitar que la función altere los valores de los argumentos, utiliza la palabra clave **ByVal** antes de los nombres de los argumentos.

3. Ejecuta el procedimiento **TresNumeros**.
   El procedimiento **TresNumeros** asigna valores a tres variables y luego llama a la función **Promedio2** para calcular y devolver el promedio de los números almacenados en estas variables. Los argumentos de la función son las variables **num1**, **num2** y **num3**. Observa que todos los argumentos de la función están precedidos por la palabra clave **ByVal**.
   Fíjate también en que antes del cálculo del promedio, la función **Promedio2** cambia el valor de la variable **num1**. Dentro del procedimiento de la función, la variable **num1** es igual a 11 (10 + 1). Por lo tanto, cuando la función pasa el promedio calculado al procedimiento **TresNumeros**, la función **MsgBox** muestra el resultado 20,3333333 y no 20 como se esperaba. Las siguientes tres funciones **Msgbox** muestran el contenido de cada una de las variables. Los valores almacenados en estas variables son los mismos que los valores originales asignados a ellas: 10, 20 y 30.
   ¿Qué sucederá si se omite la palabra **ByVal** delante del argumento **num1** en la línea de declaración de la función **Promedio2**? El resultado de la función seguirá siendo el mismo, pero el contenido de la variable **num1** mostrado por **MsgBox**, **sería** 11. La función **Promedio2** no sólo ha devuelto un resultado inesperado (20,3333333 en lugar de 20) sino que también ha modificado los datos originales almacenados en la variable **num1**. Para evitar que VBA cambie permanentemente los valores suministrados a la función, debes utilizar la palabra clave **ByVal**.

Las palabras ByVal y ByRef

Dado que cualquiera de las variables que se pasa a un procedimiento **Function** (o a uno **Sub**) puede ser modificada por el procedimiento que lo recibe, es importante saber cómo proteger el valor original de una variable. VBA tiene dos palabras clave que dan o niegan permiso para cambiar el contenido de una variable: **ByRef** y **ByVal**. Por defecto, VBA pasa la información a un procedimiento **Function** (o a uno **Sub**) por referencia (**ByRef**), refiriéndose a los datos originales especificados en el argumento de la función en el momento en que ésta es llamada. Por lo tanto, si la función altera el valor del argumento, se cambia el valor original. Obtendrás este resultado si omites la palabra clave **ByVal** delante del argumento **num1** en la línea de declaración de la función **Promedio2**.

Si quieres que el procedimiento de la función cambie el valor original, no necesitas insertar explícitamente la palabra **ByRef**, porque las variables son pasadas de esta forma por defecto. Cuando usas la palabra **ByVal** delante delnombre del argumento, VBA usa el argumento como valor. Esto significa que VBA hace una copia de los datos originales y pasa esa copia a la función. Si la función cambia el argumento proporcionado por valor, los datos originales no cambian, solo cambia la copia. Por eso cuando la función **Promedio2** cambió el valor del argumento **num1**, el valor original de la variable **num1** permaneció igual.

4.3 Los argumentos opcionales

En ocasiones, puede que desees aportar un valor adicional a una función. Digamos que tienes una función que calcula el precio de una comida por persona. Pero a veces, sin embargo, te

gustaría que la función realizara el mismo cálculo para dos o más personas. Para indicar que no siempre se requiere un argumento en el procedimiento, debes preceder el nombre del argumento con la palabra clave **Optional**. Los argumentos opcionales aparecen al final de la lista de argumentos, después de los nombres de los argumentos obligatorios. Los opcionales deben ser siempre del tipo **Variant**. Esto significa que no se puede especificar el tipo de argumento utilizando la palabra clave **As**.

En la sección anterior creaste una función para calcular el promedio de tres números. Supongamos que quieres usar esta función para calcular el promedio de dos números. Podrías definir el tercer argumento de la función como **Opcional**. Para no modificar la función **Promedio2** original, vamos a crear la función **PromedioOpcional** para calcular el promedio de dos o tres números:

1. Agrega un nuevo módulo al proyecto **ProyectoFunction** y llámalo Ejemplo4.
2. Haz clic en el módulo recién creado e introduce el siguiente procedimiento:

```
Function PromedioOpcional(num1, num2, Optional num3)
    Dim Numeros As Integer
    Numeros = 3
    If IsMissing(num3) Then
        num3 = 0
        Numeros = Numeros - 1
    End If
    PromedioOpcional = (num1 + num2 + num3) / Numeros
End Function
```

Analicemos en detalle esta función. **PromedioOpcional** puede tener hasta tres argumentos. Los argumentos **num1** y **num2** son obligatorios. **num3** es opcional. Observa como el nombre de **num3** está precedido de la palabra **Optional** y que se encuentra en último lugar en la lista de argumentos.

Como no se declara el tipo de dato de los argumentos, VBA los trata como **Variant**. Dentro del procedimiento de la función, la variable **Numeros** se declara como **Integer** y luego se le asigna un valor inicial de 3. Como la función debe ser capaz de calcular el promedio de dos o tres números, la función **IsMissing** comprueba el número de argumentos suministrados. Si el tercer argumento no se introduce, la función **IsMissing** pone en su lugar el valor cero y al mismo tiempo reduce el valor de la variable **Numeros** en 1. Por lo tanto, si omitimos el argumento **num3**, la variable **Numeros** es 2. La siguiente instrucción calcula el promedio en base a los datos suministrados y el resultado se asigna al nombre de la función.

La función **IsMissing** permite determinar si se ha introducido el argumento opcional. **IsMissing** devuelve el valor lógico **True** si no se ha introducido y **False** cuando detecta que existe el tercer argumento.

Como puedes comprobar, **IsMissing** se ha utilizado dentro de una cláusula **If**. En el Capítulo 5 entraremos en profundidad dentro de las cláusulas de decisión.

Si (`If`) falta el argumento num3, entonces (`Then`) VBA le asigna el número 0 y reduce el valor de `Numeros` en 1.

3. Llama a la función desde la ventana **Inmediato** de esta forma:

```
?PromedioOpcional(2,3)
```

Cuando presiones **Intro**, VBA mostrará el resultado 2,5. Si introduces lo siguiente:

```
?PromedioOpcional(2,3,5)
```

El resultado será 3,33333333333333.

Como has visto, la función `PromedioOpcional` permite calcular el promedio de dos o tres números. Tú decides qué valores y cuántos valores quieres calcular. Cuando comienzas a escribir los valores de los argumentos en la ventana **Inmediato**, VBA muestra el nombre del argumento opcional entre corchetes.

5 ¿Dónde están las funciones predefinidas de VBA?

VBA tiene numerosas funciones predefinidas. Estas funciones pueden ser consultadas en la ayuda de VBA:

https://docs.microsoft.com/es-es/office/vba/Language/Reference/functions-visual-basic-for-applications

Tomemos como referencia las funciones `MsgBox` o `InputBox`. Una de las características de un buen programa es su interacción con el usuario. Cuando se trabaja con Microsoft Excel, se interactúa con la aplicación utilizando varios cuadros de diálogo. Cuando cometes un error, aparece otro cuadro mostrando un mensaje que te informa del error.

Cuando escribes tus propios procedimientos, también puedes informar a los usuarios sobre un error inesperado o el resultado de un cálculo específico. Esto se hace con la ayuda de la función `MsgBox`. Hasta ahora has visto una implementación muy simple de esta función, a continuación verás cómo controlar el aspecto del mensaje. También aprenderás a obtener información del usuario con la función `InputBox`.

6 La función MsgBox

La función `MsgBox` que has utilizado hasta ahora se limitaba a mostrar el mensaje en un simple cuadro de diálogo que contaba con un botón. La única interacción ha sido hacer clic en el botón **Aceptar** para cerrarlo.

Para crear un cuadro de mensaje simple únicamente tienes que escribir la palabra `MsgBox` seguida del texto que quieras mostrar entre comillas. Por ejemplo, la instrucción

```
MsgBox "El procedimiento ha finalizado con éxito."
```

Mostrará el cuadro de diálogo de la Imagen 6.1.

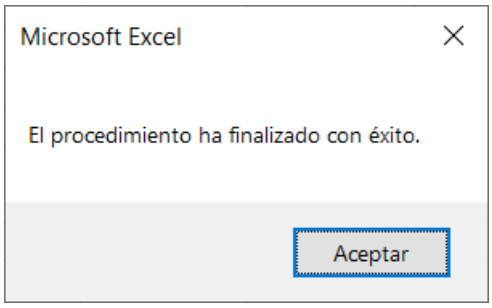


Imagen 6.1 Para mostrar un mensaje al usuario, coloca el texto como argumento de la función MsgBox.

La función **MsgBox** permite utilizar otros argumentos para establecer el número de botones que quieres mostrar en el mensaje o cambiar el título del cuadro ("Microsoft Excel"). También puedes asignar páginas de ayuda. La sintaxis de **MsgBox** es la siguiente:

```
MsgBox (texto [, botones] [, título] [,archivo de ayuda,
contexto])
```

Observa que, aunque la función admite cinco argumentos, únicamente el primero (texto) es obligatorio. Los argumentos que figuran entre corchetes son opcionales.

Cuando se introduce una cadena de texto larga en el argumento **texto**, VBA decide cómo romper el texto para que se ajuste al cuadro del mensaje. Hagamos algunos ejercicios en la ventana **Inmediato** para aprender varias técnicas para formatear el texto.

1. Introduce la siguiente instrucción en la ventana **Inmediato**. Asegúrate de introducirla en una sola línea sin olvidarte de las comillas iniciales y finales.

   ```
   MsgBox "El procedimiento ha finalizado con éxito. Ahora es
   posible conectar la fuente externa al libro de Excel para
   extraer los datos necesarios sobre los cuales crear el informe
   de las ventas del periodo anterior."
   ```

   Al pulsar **Intro**, VBA muestra el siguiente cuadro:

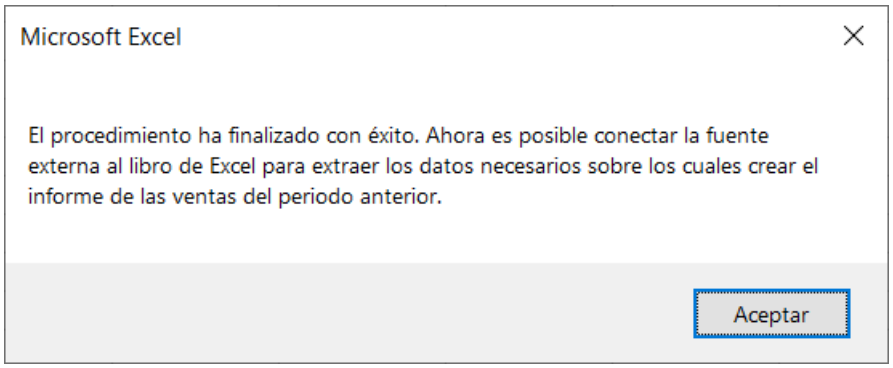


Imagen 6.2 Un mensaje largo se ve mejor cuando se trabaja de forma manual.

Cuando escribes un procedimiento VBA que muestra mensajes largos, puedes dividir el texto en varias líneas con la función **Chr**. La función **Chr** toma un argumento (un número de 0 a 255 de acuerdo al código ASCII), y realiza una acción asignada a ese número. Por ejemplo, **Chr(13)** devuelve un carácter de retorno de carro (que equivale

a la tecla **Intro**), y `Chr(10)` devuelve un carácter de salto de línea (útil para añadir un espacio entre las líneas de texto).

```
Sub MensajeLargo()
    MsgBox "El procedimiento ha finalizado con éxito. Ahora es posible" & Chr(13) _
    & " conectar la fuente externa al libro de Excel para extraer " & Chr(13) _
    & " los datos necesarios sobre los cuales crear el informe " & Chr(13) _
    & " de las ventas del periodo anterior."
End Sub
```

La Imagen 6.3 muestra el cuadro de mensaje tras ejecutar el procedimiento `MensajeLargo`.

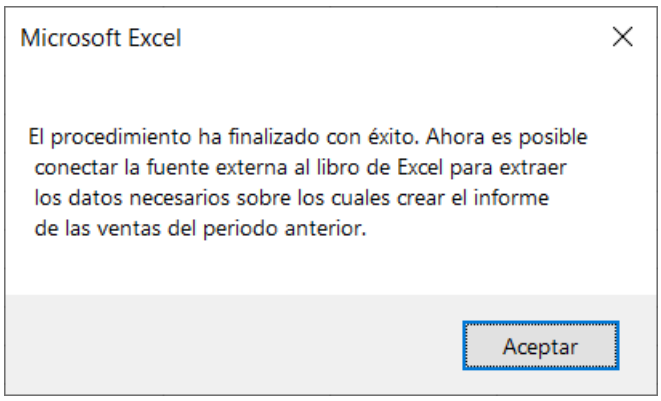


Imagen 6.3 Es posible dividir un mensaje largo en varias líneas utilizando la función Chr(13).

Debes encerrar cada fragmento de texto entre comillas. La función `Chr(13)` indica el lugar donde comienza una nueva línea. El carácter de concatenación de textos (&) se utiliza para crear una cadena de texto concatenada.

Si deseas mostrar en el texto el carácter comillas, necesitarás introducir un par de comillas más, como se muestra a continuación:

```
MsgBox "El procedimiento ""MensajeLargo"" se" & Chr(13) _
& "ha ejecutado con éxito."
```

Cuando se introducen mensajes de texto extremadamente largos, es fácil cometer un error. Como recordarás, VBA tiene un carácter especial de continuación de línea (guion bajo _) que te permite dividir una instrucción VBA larga en varias líneas. Por desgracia, este carácter no se puede usar en la ventana **Inmediato**.

2. Añade un nuevo módulo al proyecto `ProyectoFunction` y llámalo Ejemplo5.
3. Activa el módulo e introduce el procedimiento `MensajeLargo` anterior. Asegúrate de escribir un espacio delante de cada carácter de subrayado.
4. Ejecuta el procedimiento.
   Observa que la instrucción introducida en varias líneas es más legible y el código es más fácil de mantener.
   Para mantener esta legibilidad, quizá desees agregar más espacio entre las líneas de texto incluyendo líneas en blanco. Para ello, puedes usar las funciones `Chr(13)` o `Chr(10)`, como se muestra en el siguiente paso.
5. Introduce el procedimiento `MensajeLargo2` y ejecútalo.

```
Sub MensajeLargo2()
    MsgBox "El procedimiento ha finalizado con éxito. Ahora es posible" & _
    Chr(10) & Chr(10) _
    & " conectar la fuente externa al libro de Excel para extraer " & _
    Chr(13) & Chr(13) _
    & " los datos necesarios sobre los cuales crear el informe " & _
    Chr(10) & Chr(10) _
    & " de las ventas del periodo anterior."
End Sub
```

La Imagen 6.4 muestra el cuadro de mensaje generado por el procedimiento.

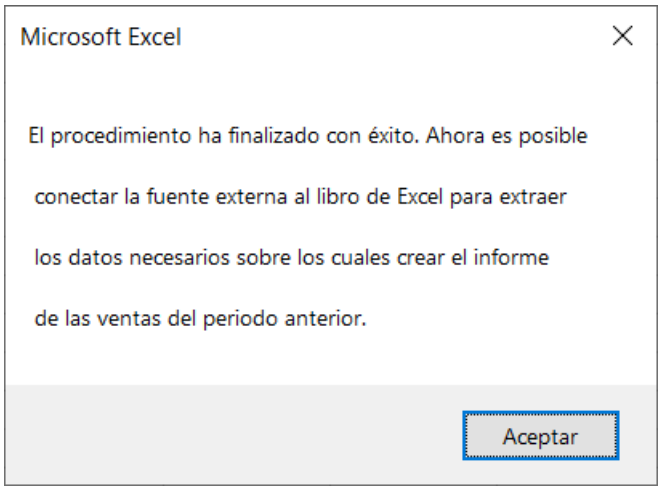


Imagen 6.4 Es posible mejorar la legibilidad del mensaje aumentando el espacio entre las líneas.

Ahora que ya dominas las técnicas de formato de texto, veamos más de cerca el siguiente argumento de la función `MsgBox`. Aunque el argumento de los botones es opcional, se utiliza con frecuencia. Este argumento especifica cuántos y qué tipos de botones quieres que aparezcan en el cuadro de mensaje. Este argumento puede ser una constante o un número como se muestra en la siguiente tabla. Si se omite el argumento el cuadro de mensaje resultante contendrá únicamente el botón **Aceptar**, como se ha visto en los ejemplos anteriores.

Configuración del botón		
vbOKOnly	0	Muestra el botón Aceptar. Opción por defecto.
vbOKCancel	1	Botones Aceptar y Cancelar.
vbAbortRetryIgnore	2	Botones Abortar, Reintentar e Ignorar.
vbYesNoCancel	3	Botones Sí, No y Cancelar.
vbYesNo	4	Botones Sí y No.

vbRetryCancel	5	Botones Reintentar y Cancelar.
Configuración de iconos		
vbCritical	16	Muestra el icono Mensaje crítico.
vbQuestion	32	Muestra el icono Consulta de advertencia.
vbExclamation	48	Muestra el icono Mensaje de advertencia.
vbInformation	64	Muestra el icono Mensaje de información.
Botón predeterminado		
vbDefaultButton1	0	El primer botón es el predeterminado.
vbDefaultButton2	256	El segundo botón es el predeterminado.
vbDefaultButton3	512	El tercer botón es el predeterminado.
vbDefaultButton4	768	El cuarto botón es el predeterminado.
Modalidad del cuadro de mensaje		
vbApplicationModal	0	El usuario debe interactuar con el mensaje antes de continuar trabajando con Excel.
vbSystemModal	4096	Todas las aplicaciones se suspenden hasta que el usuario responde al cuadro de mensaje.
Otras configuraciones		
vbMsgBoxHelpButton	16384	Agrega el botón Ayuda al cuadro de mensaje.
vbMsgBoxSetForeground	65536	Especifica la ventana del cuadro de mensaje como la ventana en primer plano.
vbMsgBoxRight	524288	Se alinea el texto a la derecha.
vbMsgBoxRtlReading	1048576	Especifica que el texto debe mostrarse de derecha a izquierda en sistemas en hebreo y árabe.

¿Cuándo deberías utilizar el argumento `botones`?

Supongamos que quieres que el usuario que utiliza el procedimiento responda a una pregunta con un **Sí** o un **No**. El cuadro de mensajes puede entonces necesitar dos botones. Si un cuadro de mensajes incluye más de un botón, uno de ellos se considera el botón predeterminado.

Cuando el usuario presiona **Intro**, el botón predeterminado se selecciona automáticamente. Dado que puedes mostrar varios tipos de mensajes (críticos, de advertencia o informativos), puedes indicar visualmente su importancia incluyendo en el argumento, un icono para el tipo de mensaje elegido.

Además del tipo de mensaje, el argumento `botones` puede incluir un ajuste para determinar si el cuadro debe cerrarse antes de que el usuario pase a otra aplicación. Es muy probable que el usuario quiera cambiar a otro programa o realizar otra tarea antes de responder a la pregunta planteada en el cuadro de mensaje. Si el cuadro de mensaje es modal (`vbApplicationModal`), el usuario debe cerrar el cuadro antes de seguir utilizando el libro de Excel. Por otro lado, si deseas inhabilitar todas las aplicaciones hasta que el usuario responda al mensaje, debes incluir el parámetro `vbSystemModal` en el argumento `botones`.

Los ajustes del argumento `botones` se dividen en cinco grupos: Configuración de botones, configuración de iconos, botón predeterminado, modalidad del cuadro y otras configuraciones. Sólo podrás configurar un ajuste por grupo.

Para crear el argumento `botones`, puedes agregar los valores de cada configuración que desees incluir. Por ejemplo, para mostrar un cuadro de mensajes con dos botones (**Sí** y **No**), el icono del signo de interrogación y el botón **No** como predeterminado, busca los valores correspondientes en la tabla anterior y súmalos. Debes obtener el número 292 (4 + 32 + 256).

Volvamos a la ventana **Inmediato** para comprobar más a fondo las capacidades de la función `MsgBox`.

1. Para mostrar rápidamente el cuadro de mensaje utilizando el argumento `botones`, introduce la siguiente instrucción en la ventana **Inmediato** y pulsa **Intro**.

   ```
   MsgBox "¿Deseas realizar el cálculo?", 292
   ```

   El resultado es el que se muestra en la Imagen 6.5.

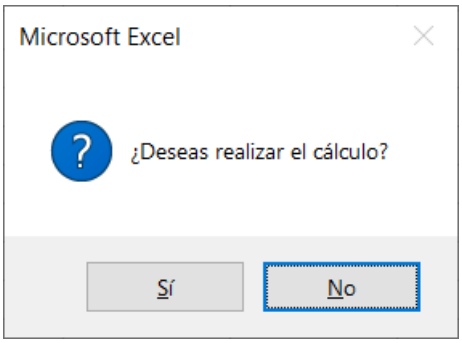


Imagen 6.5 Es posible especificar el número de botones del cuadro utilizando el argumento botones.

   Cuando creas el argumento `botones` sumando los valores de las constantes, el procedimiento deja de ser legible, ya que no existe una tabla donde puedas consultar a qué equivale el resultado 292. Para mejorar la legibilidad de los `MsgBox` es mejor utilizar constantes en lugar de sus valores.

2. Ahora introduce la siguiente instrucción:

```
MsgBox "¿Deseas realizar el cálculo?", vbYesNo + vbQuestion +
vbDefaultButton2
```

Esta declaración (que debe ser introducida en una sola línea) genera el mismo resultado que se muestra en la Imagen 6.5 y es más legible.

El siguiente ejemplo muestra cómo usar el argumento **botones** dentro de un procedimiento de VBA.

1. Agrega un nuevo módulo al proyecto **ProyectoFunction**, al que llamarás Ejemplo6.
2. Selecciona el módulo e introduce el siguiente procedimiento:

```
Sub MensajeSiNo()
    Dim Pregunta As String
    Dim MisBotones As Integer
    Pregunta = "¿Deseas crear un libro nuevo?"
    MisBotones = vbYesNo + vbPregunta + vbDefaultButton2
    MsgBox Pregunta, MisBotones
End Sub
```

En este procedimiento, la variable **Pregunta** almacena el texto del mensaje. La configuración del argumento **botones** se coloca en la variable **MisBotones**.

En lugar de usar los nombres de las constantes, puedes usar sus valores de esta forma:

```
MisBotones = 4 + 32 + 256
```

Sin embargo, al especificar los nombres de las constantes del argumento, hace que el procedimiento sea más fácil de entender tanto para ti como para otros que puedan trabajar con el procedimiento en el futuro.

Las variables **Pregunta** y **MisBotones** se utilizan como argumentos para la función **MsgBox**. Cuando se ejecuta el procedimiento, aparece el cuadro de mensaje de la Imagen 6.5. Observa que el botón **No** está seleccionado. Es el botón por defecto de este cuadro de diálogo. Si presionas **Intro**, Excel elimina el **MsgBox** de la pantalla. No sucede nada más porque el procedimiento no tiene más instrucciones.

Para cambiar el botón predeterminado usa la constante **vbDefaultButton1**.

El tercer argumento de la función **MsgBox** es el título. Aunque también es un argumento opcional, es muy útil, pues te permite crear procedimientos que ocultan que los has programado con Excel. Con este argumento puedes mostrar el texto que desees en la barra de título del cuadro de mensaje.

Supongamos que el procedimiento **MensajeSiNo** muestra en la barra de título el texto “Nuevo libro de trabajo”. El siguiente procedimiento (**MensajeSiNo2**) muestra el uso del argumento título:

```
Sub MensajeSiNo2()
Dim Pregunta As String
    Dim MisBotones As Integer
```

```
    Dim MiTitulo As String
    Pregunta = "¿Deseas crear un libro nuevo?"
    MisBotones = vbYesNo + vbPregunta + vbDefaultButton2
    MiTitulo = "Nuevo libro de trabajo"
    MsgBox Pregunta, MisBotones, MiTitulo
End Sub
```

El texto del argumento **título** se almacena en la variable **MiTitulo**. Si no se especifica el argumento **título**, Excel muestra por defecto el texto "Microsoft Excel".

Observa que los argumentos se enumeran en el orden determinado por la sintaxis de la función **MsgBox**:

```
MsgBox (texto [, botones] [, título] [,archivo de ayuda, contexto])
```

Si deseas establecer tu propio orden, debes preceder el valor de cada argumento con su nombre:

```
MsgBox title:=MiTitulo, prompt:=Pregunta, buttons:=MisBotones
```

Los dos últimos argumentos opcionales, **archivo de ayuda** y **contexto** son utilizados por los programadores con experiencia en el uso de archivos de ayuda en el entorno de Windows. El argumento **archivo de ayuda** indica el nombre de un archivo de ayuda especial que contiene información adicional que puedes mostrar al usuario. Cuando especificas este argumento, el botón de ayuda se agregará al cuadro del mensaje.

6.1 Valores devueltos por la función MsgBox

Cuando se muestra un cuadro de texto con un botón, al hacer clic en **Aceptar** o al pulsar la tecla **Intro**, desaparece de la pantalla.

Sin embargo, cuando un cuadro tiene más de un botón, el procedimiento que estés creando debería detectar qué botón se ha pulsado. Para ello, debes guardar el resultado del cuadro de mensaje en una variable. La siguiente tabla muestra los valores que devuelve la función **MsgBox**.

Aceptar	vbOK	1
Cancelar	vbCancel	2
Abortar	vbAbort	3
Reintentar	vbRetry	4
Ignorar	vbIgnore	5
Sí	vbYes	6
No	vbNo	7

Revisemos el procedimiento **MensajeSiNo2** para averiguar qué botón pulsó el usuario:

1. En el módulo **Ejemplo6**, introduce el procedimiento **MensajeSiNo3**:

```
Sub MensajeSiNo3()
    Dim Pregunta As String
    Dim MisBotones As Integer
    Dim MiTitulo As String
    Dim MiEleccion As Integer
    Pregunta = "¿Deseas crear un libro nuevo?"
    MisBotones = vbYesNo + vbPregunta + vbDefaultButton2
    MiTitulo = "Nuevo libro de trabajo"
    MiEleccion = MsgBox(Pregunta, MisBotones, MiTitulo)
    MsgBox MiEleccion
End Sub
```

   En el procedimiento anterior, asignamos el resultado de la función **MsgBox** a la variable **MiEleccion**. Observa que los argumentos de la función **MsgBox** se encuentran ahora entre paréntesis:

```
MiEleccion = MsgBox(Pregunta, MisBotones, MiTitulo)
```

2. Ejecuta el procedimiento **MensajeSiNo3**.
   Cuando se ejecuta el procedimiento, se muestra un cuadro con dos botones. Cuando haces clic en el botón **Sí**, la declaración MsgBox **MiEleccion** muestra el número 6. Si haces clic en **No**, se muestra el número 7.

MsgBox ¿con paréntesis o sin ellos?

Utiliza paréntesis para encerrar la lista de argumentos de la función MsgBox cuando quieras utilizar el resultado devuelto por la función. Al enumerar los argumentos de la función sin paréntesis, le dices a VBA que quieres ignorar el resultado de la función cuando MsgBox contenga más de un botón

7 La función InputBox

La función **InputBox** muestra un cuadro de diálogo con un mensaje que solicita al usuario que introduzca datos. Este cuadro de diálogo tiene dos botones: **Aceptar** y **Cancelar**. Al hacer clic en **Aceptar**, la función **InputBox** devuelve la información introducida en el cuadro de texto. Cuando seleccionas **Cancelar**, la función devuelve la cadena vacía (“”). La sintaxis de **InputBox** es la siguiente:

```
InputBox (texto[,título][, por defecto][, posx][, posy][,
archivo de ayuda, contexto])
```

El primer argumento (**texto**) es el mensaje de texto que quieres mostrar en el cuadro de diálogo. Se pueden introducir cadenas texto largas en varias líneas utilizando las funciones

Chr(13) o **Chr(10)** (ver los ejemplos de **Msgbox** mostrados anteriormente en este capítulo). El resto de los argumentos de **InputBox** son opcionales.

El segundo argumento (**título**), permite cambiar el título por defecto del cuadro de diálogo. El valor por omisión es "Microsoft Excel".

El tercer argumento de la función **InputBox** (**por defecto**), permite mostrar un valor por defecto en el cuadro de texto. Si se omite este argumento, se muestra el cuadro de edición vacío.

Los dos argumentos siguientes (**posx** y **posy**) permiten especificar la posición exacta en la que debe aparecer el cuadro de diálogo e la pantalla. Si omites estos argumentos, el cuadro aparece en el centro de la ventana actual. El argumento **posx** determina la posición horizontal de la ventana del cuadro, desde el borde izquierdo de la pantalla. Si se omite, el cuadro de diálogo se centra horizontalmente. El argumento **posy** determina la posición vertical desde la parte superior de la pantalla. Si se omite este argumento, el cuadro de diálogo se posiciona verticalmente a un tercio de la parte inferior de la pantalla. Tanto **posx** como **posy** se miden en twips. Un twip equivale aproximadamente a 0,0007 pulgadas.

Los dos últimos argumentos (**archivo de ayuda** y **contexto**), se utilizan de la misma manera que los argumentos correspondientes de la función **MsgBox** que hemos visto anteriormente en este capítulo.

Ahora que conoces el significado de los argumentos de **InputBox**, veamos algunos ejemplos de uso de esta función.

Agrega un nuevo módulo al proyecto ProyectoFunction del libro Capítulo 4 – Funciones.xlsm. Llámalo Ejemplo7.

1. Introduce el siguiente procedimiento en el módulo que acabas de crear:

```
Sub LugarNacimiento()
    InputBox prompt:="Introduce el lugar de nacimiento:" & Chr(13) _
    & " (Por ejemplo, España, México, Colombia, Chile, etc.) "
End Sub
```

   Este procedimiento muestra un cuadro de diálogo con dos botones, como se muestra en la imagen 7.1. El mensaje de entrada se muestra en dos líneas.

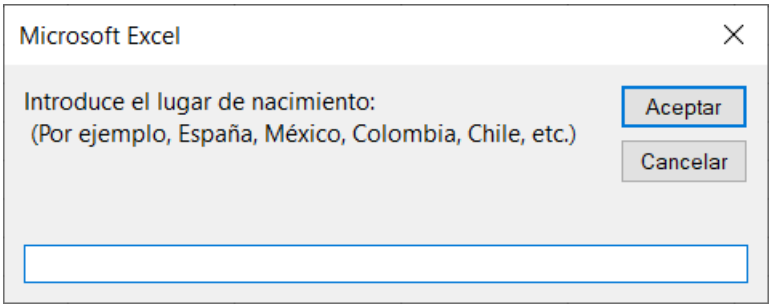


Imagen 7.1 El cuadro de diálogo generado por el procedimiento Lugar Nacimiento.

Al igual que con **MsgBox**, si prevés utilizar los datos introducidos por el usuario en el cuadro de diálogo, debes almacenar el resultado de la función **InputBox** en una variable.

2. Introduce el procedimiento **LugarNacimiento2** donde se asigna el resultado de **InputBox** a la variable **Población**:

```
Sub LugarNacimiento2()
    Dim MiTexto As String
    Dim Poblacion As String
    Const MiTitulo = "Introduce la población"
    MiTexto = "Introduce el lugar de nacimiento:" & Chr(13) _
    & "(Por ejemplo, España, México, Colombia, Chile, etc.)"
    Poblacion = InputBox(MiTexto, MiTitulo)
    MsgBox "Naciste en " & Poblacion & ".", , "Tu respuesta"
End Sub
```

Observa que esta vez los argumentos de la función **InputBox** están entre paréntesis. Los paréntesis son necesarios si deseas utilizar el resultado de la función **Inputbox** más adelante en el procedimiento. El procedimiento **LugarNacimiento2** utiliza una constante para especificar el texto que aparecerá en la barra de título del cuadro de diálogo.

Debido a que el valor de la constante permanece igual a lo largo de la ejecución del procedimiento, puedes declarar el título del cuadro como una constante. Sin embargo, si prefieres usar una variable, también puedes hacerlo. Cuando se ejecuta un procedimiento utilizando la función **InputBox**, el cuadro generado por esta función siempre aparece en el mismo área de la pantalla. Para cambiar la ubicación del cuadro de diálogo, debes proporcionar los argumentos **posx** y **posy**, como se explicó anteriormente.

3. Ejecuta el procedimiento **LugarNacimiento2**.
4. Para mostrar el cuadro de diálogo en la esquina superior izquierda de la pantalla, modifica la función **LugarNacimiento2** de la siguiente forma y luego ejecútala:

```
Poblacion = InputBox(MiTexto, MiTitulo, , 1, 200)
```

Observa que el argumento **MiTitulo** va seguido de dos comas. La segunda coma marca la posición del argumento **por defecto**, que se ha omitido. Los dos siguientes argumentos determinan la posición horizontal y vertical del cuadro de diálogo. Si omites la segunda coma después del argumento **MiTitulo**, VBA usará el número 1 como argumento **por defecto**. Si precedes los valores de los argumentos por sus nombres (por ejemplo, **prompt:=MiTexto, title:=MiTitulo, xpos:=1, ypos:=200**), no tendrás que acordarte de poner una coma en el lugar de cada argumento omitido.

¿Qué ocurre si introduces un número en lugar de una ciudad? Dado que los usuarios suelen introducir datos incorrectos en los cuadros de diálogo, el procedimiento debe verificar que los datos introducidos pueden ser utilizados en posteriores manipulaciones de datos. La función **InputBox** en sí misma no proporciona una característica para la validación de datos. Para ello, debes aprender algunas instrucciones adicionales de VBA que verás en el siguiente capítulo.

7.1 Averiguar y convertir tipos de datos

El resultado devuelto por una función **InputBox** es siempre una cadena de texto. Si el usuario introduce un número, el valor que el usuario introdujo debe convertirse en un valor numérico antes de realizar cálculos matemáticos.

VBA es capaz de convertir valores de un tipo de datos a otros.

Conversión de tipos de datos

Consulta el Capítulo3 para obtener más información sobre el uso de la función **VarType** para determinar el tipo de datos de una variable y las funciones de conversión de tipo de dato más comunes.

Probemos un procedimiento que sugiere qué tipo de datos debe introducir el usuario, proporcionando un valor por defecto en el cuadro **InputBox**.

1. Haz clic en el módulo Ejemplo7 para mostrar su ventana **Código** e introduce el siguiente procedimiento:

```
Sub SumarDosNumeros()
    Dim MiTexto As String
    Dim Valor1 As String
    Dim Valor2 As Integer
    Dim MiSuma As Single
    Const MiTitulo = "Introducción de datos"
    MiTexto = "Introduce un número:"
    Valor1 = InputBox(MiTexto, MiTitulo, 0)
    Valor2 = 2
    MiSuma = Valor1 + Valor2
    MsgBox "El resultado es " & MiSuma & _
    " (" & Valor1 & " + " & CStr(Valor2) + ")", _
    vbInformation, "Total"
End Sub
```

El procedimiento **SumarDosNumeros** muestra el cuadro de diálogo de la Imagen 7.2. Observa que este cuadro de diálogo tiene dos características especiales que se

obtienen al utilizar el título opcional y los argumentos por defecto de la función **InputBox**. En lugar del título por defecto "Microsoft Excel" el cuadro de diálogo muestra una cadena de texto definida por el contenido de la constante **MiTitulo**. El cero introducido como valor por defecto en el argumento **por defecto** sugiere que el usuario introduzca un número en lugar de un texto. Una vez que el usuario proporciona los datos y hace clic en **Aceptar**, a ese dato se asigna la variable **Valor1**.

```
Valor1 = InputBox(MiTexto, MiTitulo, 0)
```

2. Ejecuta el procedimiento **SumarDosNumeros** e introduce un número cuando se muestre el cuadro de diálogo.

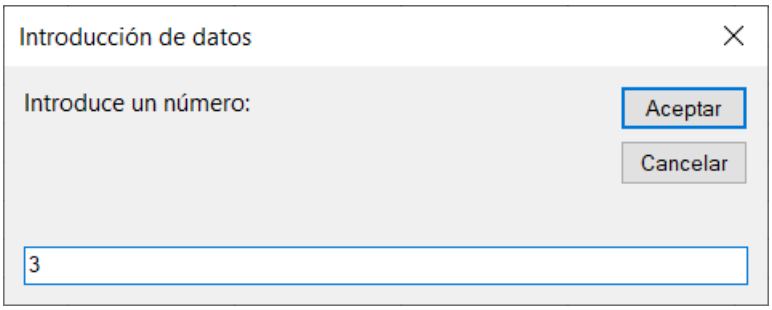


Imagen 7.2 Para sugerir al usuario que introduzca un tipo específico de datos, es posible que desees proporcionar un valor en el argumento "por defecto".

El tipo de dato de la variable Valor1 es **String**.

3. Puedes comprobar fácilmente el tipo de datos si cambias la instrucción anterior por esta declaración:

```
MsgBox VarType(Valor1)
```

Cuando VBA ejecute la línea anterior, mostrará un cuadro de mensaje con el número 8. Recuerda que en el Capítulo 3 vimos que este número representa el tipo de datos **String**.

La línea **MiSuma = Valor1 + Valor2** suma el valor almacenado en la variable **Valor2** al valor que introdujo el usuario en el cuadro y asigna el resultado a la variable **MiSuma**. Como el tipo de datos de la variable **Valor1** es **String**, antes de utilizar los datos de esta variable en el cálculo, VBA se pone a trabajar en el backstage para realizar la conversión de tipo de datos. VBA entiende la necesidad de la conversión. Sin ella, los dos tipos de datos incompatibles (String e Integer) generarían un error de tipos. El procedimiento finaliza con la función **MsgBox,** que muestra el resultado del cálculo además de cómo se calculó el total. Observa que la variable **Valor2** tiene que ser convertida del tipo de datos Integer a String utilizando la función **CStr** para poder mostrarla en el cuadro de mensaje.

```
MsgBox "El resultado es " & MiSuma & _
    " (" & Valor1 & " + " & CStr(Valor2) + ")", _
    vbInformation, "Total"
```

Definir una constante

Para asegurarte de que todas las barras de título en un procedimiento VBA en particular muestren el mismo texto, asigna el texto del título a una constante. Haciendo esto ahorrarás tiempo al no tener que escribir el texto del título más de una vez.

8 Uso del método InputBox

Además de la función **InputBox**, también existe el método **InputBox**. Si activas el **Examinador de objetos** y escribes "InputBox" en el cuadro de búsqueda, VBA mostrará dos resultados de **Inputbox**. Uno en la biblioteca de Excel y otro en la biblioteca de VBA, como se muestra en la Imagen 8.1.

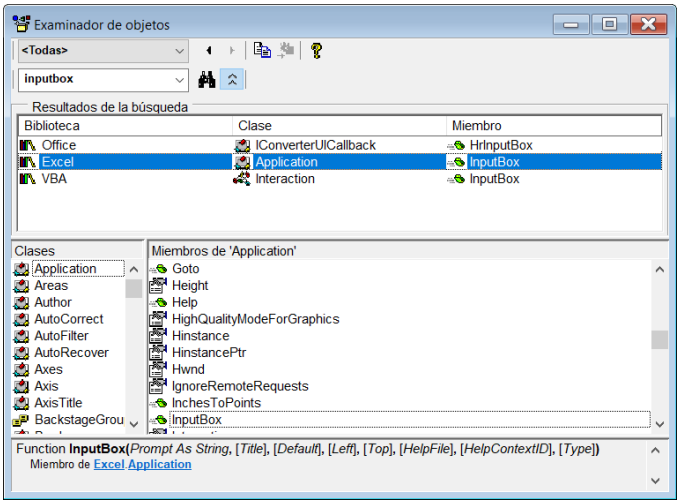


Imagen 8.1 No olvides utilizar el Examinador de objetos cuando investigues las funciones y métodos de VBA.

El método **InputBox** disponible en la biblioteca de Excel tiene una sintaxis ligeramente diferente de la función **InputBox**. Su sintaxis es:

```
Expresión.InputBox (texto[,título][, por defecto][, posx][,
posy][, archivo de ayuda, contexto][, tipo)
```

Todos los argumentos entre corchetes son opcionales. El argumento **texto** es el mensaje que se mostrará en el cuadro de diálogo. **Título** hace referencia al título del cuadro de diálogo. **Por defecto** es el valor que aparecerá seleccionado en el cuadro de texto cuando éste se inicie.

Los argumentos **posx** y **posy** especifican la posición del cuadro en la pantalla. Como vimos anteriormente, los valores de estos argumentos se introducen en puntos o Twips. Un punto equivale a 1/72 pulgadas. Los argumentos **archivo de ayuda** y **contexto** muestran el botón de ayuda y el tema específico cuando el usuario haga clic en él.

El último argumento, **tipo**, especifica el tipo de datos devueltos. Si omites este argumento, el método **InputBox** devolverá el tipo Texto. Los valores del argumento **tipo** se muestran en la siguiente tabla.

0	Una fórmula.
1	Un número.
2	Una cadena (texto).
4	Un valor lógico (True o False).
8	Una referencia de celda como objeto **Range**.
16	Un valor de error.
64	Una matriz de valores.

Puedes permitir que el usuario introduzca un número o un texto en el cuadro de texto si utilizas 3 para el argumento **tipo**. Este valor se obtiene sumando los valores de un número (1) y una cadena (2), como se muestra en la tabla anterior. El método **InputBox** es bastante útil para los procedimientos VBA que requieren que el usuario seleccione un rango de celdas en una hoja de cálculo. Veamos un ejemplo de procedimiento que utiliza el método **InputBox** de Excel:

1. Cierra el **Examinador de objetos** si lo tienes abierto.
2. En el módulo Ejemplo7, introduce el siguiente procedimiento:

```
Sub MiRango()
    Dim NuevoRango As Range
    Dim Mensaje As String
    Mensaje = "Utiliza el ratón para seleccionar un rango:"
    Set NuevoRango = Application.InputBox(prompt:=Mensaje, _
    Title:="Rango a formatear", _
    Type:=8)
    NuevoRango.NumberFormat = "0.00"
    NuevoRango.Select
End Sub
```

El procedimiento **MiRango** comienza con una declaración de una variable de objeto (**NuevoRango**). Como recordarás del Capítulo 3, las variables de objeto apuntan a la ubicación de los datos. El rango de celdas que el usuario selecciona se asigna a la variable de objeto **NuevoRango**. Fíjate en la palabra clave **Set** antes del nombre de la variable.

```
Set NuevoRango = Application.InputBox(prompt:=Mensaje, _
    Title:="Rango a formatear", _
    Type:=8)
```

El argumento `Type` (`Type:=8`) permite al usuario seleccionar cualquier rango de celdas.

Cuando el usuario selecciona las celdas, la siguiente instrucción `NuevoRango.NumberFormat = "0.00"` cambia el formato de las celdas seleccionadas. La última instrucción selecciona el rango de las celdas que resaltó el usuario.

3. Presiona **Alt + F11** para activar la ventana de la aplicación de Microsoft Excel y luego presiona **Alt + F8** para mostrar el cuadro de diálogo **Macro**. Selecciona el procedimiento `MiRango` y ejecútalo.
   VBA muestra un cuadro de diálogo que pide al usuario que seleccione un rango de celdas en la hoja.
4. Usa el ratón para seleccionar las celdas que desees. La Imagen 8.2 muestra cómo VBA introduce la referencia del rango seleccionado en el cuadro de texto mientras arrastras el ratón seleccionando las celdas.

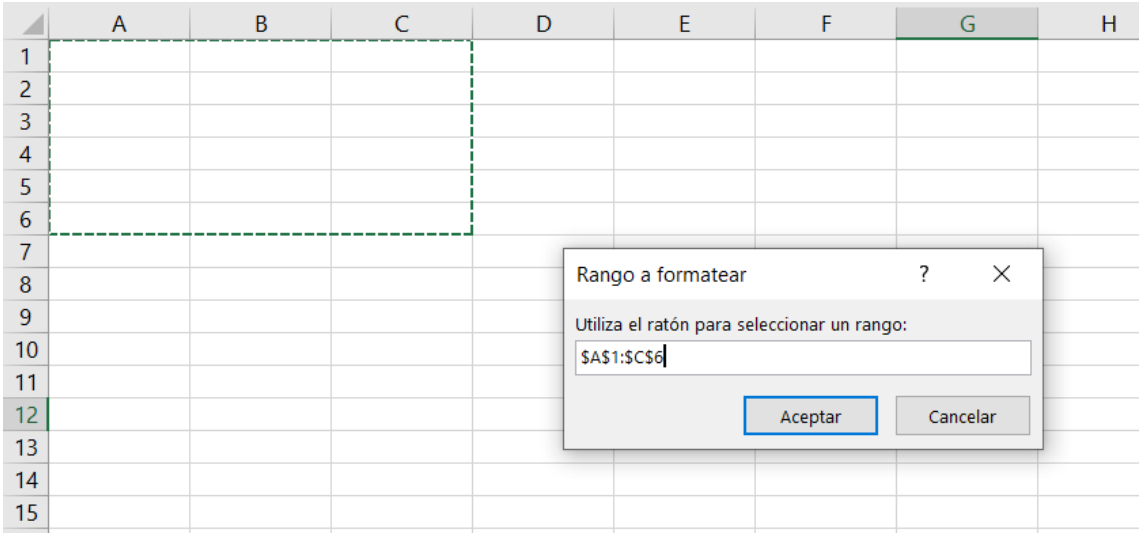


Imagen 8.2 Con el método InputBox podemos obtener un rango de celdas seleccionándolas con el ratón.

5. Cuando termines de seleccionar las celdas, haz clic en **Aceptar** en el cuadro de diálogo. El rango seleccionado ahora contiene otro formato. Para comprobarlo, introduce un número entero en cualquiera de las celdas seleccionadas. El número debería aparecer formateado con dos decimales.
6. Vuelve a ejecutar el procedimiento y, cuando aparezca el cuadro de diálogo, haz clic en **Cancelar**.
   Cuando haces clic en el botón **Cancelar** o presionas la tecla **Esc**, VBA muestra el mensaje de error "Se requiere un objeto". Al hacer clic en el botón **Depurar** del mensaje de error, VBA resaltará la línea de código que causó el error. Si has presionado el botón **Cancelar** es porque no quieres seleccionar ninguna celda, por lo que debes encontrar una forma de ignorar el error que se muestra. Usando la declaración especial `On Error GoTo` puedes tomar un desvío cuando se produce un error. Esta instrucción tiene la siguiente sintaxis:

```
On Error GoTo etiqueta
```

Esta instrucción debe colocarse justo debajo de las líneas de declaración de las variables. El nombre de la etiqueta puede ser cualquier palabra que desees, excepto alguna de las palabras clave de VBA. Si ocurre un error, VBA saltará a la etiqueta especificada, como se muestra en el paso 8.

7. Haz clic en **Ejecutar – Reiniciar** para cancelar el procedimiento que se estaba ejecutando.
8. Modifica el procedimiento `MiRango` para que se parezca el procedimiento `MiRango2` que se muestra a continuación:

```
Sub MiRango2()
    Dim NuevoRango As Range
    Dim Mensaje As String
    On Error GoTo IrAlFinal
    Mensaje = "Utiliza el ratón para seleccionar un rango:"
    Set NuevoRango = Application.InputBox(prompt:=Mensaje, _
    Title:="Rango a formatear", _
    Type:=8)
    NuevoRango.NumberFormat = "0.00"
    NuevoRango.Select
IrAlFinal:
End Sub
```

9. Ejecuta el nuevo procedimiento y haz clic en **Cancelar** en cuanto aparezca el cuadro de diálogo. Observa que esta vez el procedimiento no genera el error al cancelar el cuadro. Cuando VBA encuentra el error, salta a la etiqueta colocada al final del procedimiento. Las declaraciones colocadas entre `On Error GoTo IrAlFinal` y la etiqueta `IrAlFinal` se ignoran. En el Capítulo 9, encontrarás otros ejemplos de errores.

9 Resumen

En este capítulo has aprendido la diferencia entre los procedimientos `Sub`, que realizan acciones, y los procedimientos `Function`, que devuelven valores. Los procedimientos `Sub` pueden grabarse con la grabadora de macros o escribirse a mano, pero los `Function` solo pueden escribirse a mano, porque pueden necesitar argumentos.

Has aprendido a pasar argumentos a las funciones y a determinar el tipo de datos del resultado de una función. También has aumentado tu vocabulario de palabras clave VBA con palabras como `ByVal`, `ByRef` u `Optional`. También has aprendido cómo con la ayuda de los parámetros, los subprocedimientos pueden pasar valores a los procedimientos que los llaman. Después de leer este capítulo y realizar sus prácticas, deberías ser capaz de crear algunas funciones personalizadas que se adapten a tus necesidades específicas. También deberías ser capaz de interactuar con los usuarios de los procedimientos que escribas utilizando las funciones `MsgBox` e `InputBox`, así como el método `InputBox` de Excel.

El Capítulo 5 te introducirá en la toma de decisiones. Aprenderás cómo cambiar el rumbo de un procedimiento VBA en base a los resultados de las condiciones que le proporciones.

Sub o Function, ¿cuál debería utilizar?

Crea un procedimiento Sub cuando…	Crea un procedimiento Function cuando…
Quieras realizar algunas acciones.	Quieras realizar un cálculo más de una vez.
Quieras que el usuario introduzca datos.	Necesites un cálculo que no puedas realizar con funciones de hoja.
Quieras mostrar un mensaje en la pantalla.	Tengas que llamar al mismo bloque de instrucciones más de una vez.
	Quieras comprobar si una expresión es verdadera o falsa.

Capítulo 5
Las estructuras condicionales

VBA, como otros lenguajes de programación, ofrece declaraciones especiales que te permiten incluir puntos de decisión en tus procedimientos.

Pero ¿qué es un punto de decisión? Supongamos que alguien se acerca a ti y te pregunta: "¿Te gusta el color verde?". Después de pensarlo responderás "sí" o "no", y si no estás seguro, puedes responder "tal vez". En la programación debes ser decisivo.

Solo se permiten respuestas de "sí" y "no". En programación, todas las decisiones se basan en las respuestas proporcionadas. Si la respuesta es positiva, el procedimiento ejecuta un bloque específico de instrucciones. Si la respuesta es negativa, el procedimiento ejecuta otro bloque de instrucciones o simplemente no hace nada. En este capítulo aprenderás a utilizar las declaraciones condicionales de VBA para alterar el flujo del programa. Las declaraciones condicionales se denominan a menudo "estructuras de control", ya que te dan la capacidad de controlar el flujo del procedimiento VBA saltándote ciertas declaraciones y "desviándose" a otra parte del procedimiento.

1 Los operadores lógicos y relacionales

A través de expresiones condicionales, tomarás decisiones en los procedimientos de VBA. Una expresión condicional es una expresión que utiliza uno de los operadores relacionales de la Tabla 1, uno de los operadores lógicos de la Tabla 2, o una combinación de ambos. Cuando VBA encuentra una expresión condicional en el procedimiento, evalúa la expresión para determinar si es verdadera o falsa.

=	Igual a
<>	Distinto a
>	Mayor que

<	Menor que
>=	Mayor o igual a
<=	Menor o igual a

Tabla 1 Operadores relacionales

AND	Todas las condiciones deben ser verdaderas para evaluar la función como verdadera.
OR	Al menos una de las condiciones debe ser verdadera para evaluar la condición como verdadera.
NOT	Para negar una condición. Si la evaluación es verdadera, NOT la transforma en falsa.

Tabla 2 Operadores lógicos

2 La estructura IF … THEN

La forma más sencilla de conseguir que se tome una decisión en un procedimiento VBA es utilizar la declaración **`If … Then`**. Supongamos que quieres elegir una acción que depende de una condición. Puedes utilizar la siguiente estructura:

`If` condición **`Then`** acción.

Por ejemplo, para eliminar una fila en blanco de una hoja, primero comprueba si la celda activa está en blanco. Si el resultado de la prueba es verdadero, se elimina toda la fila que contiene esa celda:

```
If ActiveCell = "" Then Selection.EntireRow.Delete
```

Si la celda activa no está en blanco, VBA ignorará la declaración que sigue a la palabra **`Then`**.

Puede que algunas veces quieras realizar varias acciones cuando la condición sea verdadera. Aunque podrías agregar otras declaraciones en la misma línea separándolas con dos puntos, el código se verá más claro si utilizas la versión multilínea de la declaración **`If … Then`**, como se muestra a continuación:

```
If Condición Then
    Instrucción1
    Instrucción2
    InstrucciónN
End If
```

Por ejemplo, para realizar varias acciones cuando el valor de la celda activa es superior a 50, puedes escribir el siguiente bloque de instrucciones:

```
If ActiveCell.Value > 50 Then
    MsgBox "El valor exacto es " & ActiveCell.Value
    Debug.Print ActiveCell.Address & ": " & ActiveCell.Value
End If
```

En este ejemplo, las instrucciones entre las palabras clave `Then` y `End If` no se ejecutan si el valor de la celda activa es menor o igual a 50. Observa que el bloque `If … Then` debe terminar siempre con las palabras clave `End If`.

¿Cómo trata VBA una declaración de este tipo? Cuando el flujo del procedimiento llega a una declaración `If … Then`, evalúa la condición que se encuentra entre estas palabras.Vamos a hacer una prueba. Evaluemos la siguiente condición

```
ActiveCell.Value > 50
```

1. Crea un nuevo libro de Excel.
2. Selecciona cualquier celda y escribe el número 50
3. Presiona **Alt + F11** para acceder al editor de VBA.
4. Activa la ventana **Inmediato**.
5. Escribe la siguiente instrucción y presiona **Intro** cuando hayas finalizado:

   ```
   ? ActiveCell.Value > 50
   ```

   Cuando pulsas **Intro**, VBA muestra el resultado de la prueba (`False`). Cuando el resultado de la prueba es falso, VBA no se molestará en leer la declaración que sigue a la palabra clave `Then` en el código. Simplemente pasará a leer la siguiente línea del procedimiento, si es que hay alguna. Si no existen más líneas, el procedimiento finalizará.
6. Ahora cambia el operador a menor o igual (<=) y vuelve a presionar **Intro** para que VBA evalúe la condición.

   ```
   ? ActiveCell.Value <= 50
   ```

   Esta vez la prueba es verdadera y VBA saltará a cualquier declaración que encuentre después de la palabra `Then`.
7. Cierra la ventana **Inmediato**.

Ahora que sabes cómo se evalúan las condiciones, probemos algo más complejo.

1. Crea un nuevo libro y guárdalo en la carpeta **Archivos Manual VBA** que creaste en C:\ con el nombre “Capítulo 5 – Estructuras decisión.xlsm”.
2. Dirígete al editor de VBA y renombra el proyecto como “Decisiones”.
3. Inserta un nuevo módulo y llámalo `IfThen`.
4. Dentro del módulo `IfThen`, introduce el siguiente procedimiento:

```
Sub IfThen_simple()
    Dim semanas As String
    semanas = InputBox("¿Cuántas semanas hay en un año?", "Test")
    If semanas <> 52 Then MsgBox "No es correcto. Prueba de nuevo."
End Sub
```

Este procedimiento almacena la respuesta del usuario en la variable semanas. El valor de la variable se compara con el número 52. Si el resultado de la comparación es verdadero (es decir, si el valor almacenado en la variable no es igual a 52), VBA mostrará el mensaje "No es correcto. Prueba de nuevo".

5. Ejecuta el procedimiento e introduce un número que no sea 52.
6. Vuelve a ejecutar el procedimiento e introduce el número 52.
   Cuando introduces el número correcto de semanas, VBA no hace nada. El procedimiento simplemente termina. Sería bueno mostrar un mensaje cuando el usuario adivine la respuesta a la pregunta, ¿verdad?
7. Escribe la siguiente instrucción en una línea aparte justo antes de la palabra clave **End Sub**:

   ```
   If semanas = 52 Then MsgBox "¡Correcto!"
   ```

8. Vuelve a ejecutar el procedimiento e introduce el número 52.
   Al introducir la respuesta correcta, VBA no ejecuta la instrucción **If semanas <> 52 Then MsgBox "No es correcto. Prueba de nuevo."** Cuando se ejecuta el procedimiento, la parte derecha de la declaración se ignora si el resultado de evaluar la condición es falso.
   Como recordarás, un procedimiento puede llamar a otro, e incluso a sí mismo. Veamos si es cierto:
9. Modifica la primera declaración **If** en el procedimiento de la siguiente manera:

   ```
   If semanas <> 52 Then MsgBox "No es correcto. Prueba de nuevo.": IfThen_simple
   ```

   Se añaden dos puntos y el nombre del procedimiento **IfThen_simple**. Si el usuario introduce una respuesta incorrecta verá un mensaje y, al hacer clic en el botón **Aceptar** del cuadro de diálogo, el **InputBox** aparecerá de nuevo y tendrá otra oportunidad de introducir otra respuesta. El usuario podrá seguir adivinando durante mucho tiempo. De hecho, no será capaz de salir del programa hasta que introduzca la respuesta correcta. Si hace clic en **Cancelar**, tendrá que vérselas con el desagradable mensaje de error "No coinciden los tipos". Ya viste en el capítulo anterior cómo usar la declaración **On Error GoTo** para evitar este error, al menos temporalmente, hasta que aprendas más sobre el manejo de errores en el Capítulo 9. Por ahora, tras introducir la estructura para manejar el error, el código quedaría de la siguiente forma:

   ```
   Sub IfThen_simple()
       Dim semanas As String
       On Error GoTo Error
       semanas = InputBox("¿Cuántas semanas hay en un año?", "Test")
       If semanas <> 52 Then MsgBox _
          "No es correcto. Pruerba de nuevo.": IfThen_simple
       If semanas = 52 Then MsgBox "¡Correcto!"
   Error:
   End Sub
   ```

   Ejecuta el procedimiento varias veces introduciendo respuestas incorrectas.

El truco de `On Error Goto` permite al usuario hacer clic en el botón **Cancelar** sin tener que visualizar el mensaje de error.

Dos estructuras para la declaración If … Then

La declaración `If … Then` tiene dos formatos: en una sola línea y en varias líneas. El formato en una sola línea es bueno para declaraciones cortas como:

```
If codigosecreto <> "esk·#e~AAskw310=" Then MsgBox _
"El código secreto no es correcto."
```

O

```
If codigosecreto = "esk·#e~AAskw310=" Then alpha = _
True: beta = False
```

En este procedimiento, `codigosecreto`, `alpha` y `beta` son los nombres de las variables. En el primer ejemplo, VBA muestra el mensaje "El código secreto no es correcto" si el valor de `codigosecreto` no es igual al valor establecido en la condición. En el segundo ejemplo, VBA establece el valor de `alpha` en `True` y `beta` en `False` cuando la variable `codigosecreto` es igual al establecido. Fíjate que la segunda instrucción a ejecutar está separada de la primera por dos puntos. Cuando existen dos o más instrucciones tras la palabra `Then`, o se trata de una línea muy larga, se hace más conveniente utilizar varias líneas, como en el siguiente ejemplo:

```
If ActiveSheet.Name = "Hoja1" Then
    ActiveSheet.Move after:=Sheets(Worksheets.Count)
End If
```

Aquí, VBA examina el nombre de la hoja activa. Si es Hoja1, la condición es verdadera y se procede a ejecutar la línea que sigue a la palabra `Then`. Como resultado, la hoja activa se mueve a la derecha de todo el conjunto de hojas.

3 La estructura IF … THEN … ELSE

Ahora ya sabes cómo mostrar un mensaje o realizar una acción cuando una o más condiciones son verdaderas o falsas. Sin embargo, ¿qué debes hacer si el procedimiento necesita tomar una decisión cuando la condición es verdadera y otra decisión diferente cuando es falsa? Añadiéndole la cláusula `Else`, puedes dirigir el procedimiento a las instrucciones apropiadas dependiendo de la evaluación de la condición.

La declaración `If … Then … Else` se puede escribir en una o varias líneas.

```
If condición Then instrucciones1 Else instrucciones2
```

Las instrucciones que siguen a la palabra clave `Then` se ejecutan si la condición es verdadera, y las instrucciones que siguen a la cláusula `Else` se ejecutan si la condición es falsa. Por ejemplo:

```
If Ventas > 5000 Then Comision = Ventas * 0.05 Else MsgBox _
```

```
"No hay comisiones."
```

Si el valor almacenado en la variable **Ventas** es superior a 5000, VBA calculará la comisión mediante la fórmula **Ventas * 0,05**. Sin embargo, si la variable **Ventas** no es mayor de 5000, VBA mostrará el mensaje "No hay comisiones".

La declaración **If … Then … Else** debe ser usada para decidir cuál de las dos acciones realizar.

La forma más habitual de utilizar esta estructura es en varias líneas, de la siguiente forma:

```
If condicion Then
    instrucciones a ejecutar si la condicion es verdadera (True)
Else
    instrucciones a ejecutar si la condicion es falsa (False"
End If
```

Observa cómo la estructura **If … Then** siempre finaliza con la línea **End If**. Acostúmbrate a utilizar sangrías en las líneas para facilitar la lectura. Aquí tienes otro ejemplo.

```
If ActiveSheet.Name = "Hoja1" Then
    ActiveSheet.Name = "Mi hoja"
    MsgBox "Has renombrado esta hoja."
Else
    MsgBox "El nombre no es el habitual."
End If
```

Si la condición (**ActiveSheet.Name = "Hoja1"**) es verdadera, VBA ejecutará las instrucciones entre **Then** y **Else** e ignorará la instrucción entre **Else** y **End If**. Si la condición es falsa, VBA omitirá las instrucciones **Then** y **Else** y ejecutará la instrucción entre **Else** y **End If**.

Veamos el ejemplo del procedimiento completo:

1. Inserta un nuevo módulo en el proyecto Decisiones y llámalo IfThenElse.
2. Introduce el siguiente código:

```
Sub TipoDia()
    Dim Respuesta As String
    Dim Pregunta As String
    Dim Mensaje1 As String, Mensaje2 As String
    Dim MiFecha As Date

    Pregunta = "Introduce una fecha en el formato dd/mm/aaaa:" _
    & Chr(13) & " (p.e. 23/02/2020)"
    Mensaje1 = "Día laborable"
    Mensaje2 = "Fin de semana"
    Respuesta = InputBox(Pregunta)
```

```
        MiFecha = Weekday(CDate(Respuesta))
        If MiFecha >= 2 And MiFecha <= 6 Then
            MsgBox Mensaje1
        Else
            MsgBox Mensaje2
        End If
    End Sub
```

vbSunday	1
vbMonday	2
vbTuesday	3
vbWensday	4
vbThursday	5
vbFriday	6
vbSaturday	7

Tabla 3 Valores devueltos por la función Weekday.

El procedimiento anterior pide al usuario que introduzca cualquier fecha. La cadena suministrada se convierte al tipo de datos **Date** con la función **CDate**. A continuación la función **Weekday** convierte la fecha en un número entero que indica el día de la semana. Las constantes del día de la semana se muestran en la Tabla 3. El número entero se almacena en la variable **MiFecha**. La prueba condicional se realiza para comprobar si el valor de la variable **MiFecha** es igual o mayor a 2 (>=2) y menor o igual a 6 (<=6). Si el resultado de la evaluación de la condición es verdadero, se muestra al usuario un mensaje indicando que se trata de un día laborable. En caso contrario, se muestra un mensaje indicando que es fin de semana.

3. Ejecuta el procedimiento **TipoDia** desde la ventana de VBA. Ejecútalo varias veces, introduciendo fechas diferentes. Comprueba las respuestas obtenidas con un calendario para comprobar que sean correctas.

4 La estructura IF … THEN … ELSEIF

Muy a menudo tendrás que comprobar los resultados de varias condiciones diferentes. Para unir un conjunto de condiciones **If**, puedes usar la cláusula **ElseIf**. Usando la estructura **If … Then … ElseIf**, puedes introducir más condiciones en la misma declaración. Esta es su sintaxis:

```
If condicion1 Then
    Instrucciones a ejecutar si condicion1 es True
```

```
ElseIf condicion2 Then
    Instrucciones a ejecutar si condicion2 es True
ElseIf condicion3 Then
    Instrucciones a ejecutar si condicion3 es True
ElseIf condicionN Then
    Instrucciones a ejecutar si condicionN es True
Else
    Instrucciones a ejecutar si todas las condiciones son False
End If
```

La cláusula **Else** es opcional; puedes omitirla si no hay acciones a ejecutar cuando todas las condiciones son falsas. El procedimiento puede incluir cualquier número de declaraciones y condiciones **ElseIf**. La cláusula **ElseIf** siempre se escribe antes de la cláusula **Else**. Las declaraciones en la cláusula **ElseIf** se ejecutan solo sin la condición de esta cláusula es verdadera.

Veamos el siguiente ejemplo de código:

```
If ActiveCell.Value = 0 Then
    ActiveCell.Offset(0, 1).Value = "Cero"
ElseIf ActiveCell.Value > 0 Then
    ActiveCell.Offset(0, 1).Value = "Positivo"
ElseIf ActiveCell.Value < 0 Then
    ActiveCell.Offset(0, 1).Value = "Negativo"
End If
```

El código comprueba el valor de la celda activa e introduce la etiqueta apropiada (“Cero”, “Positivo” o “Negativo”) en la columna de la derecha. Observa que no se utiliza la cláusula **Else**. Si el resultado de la primera condición (**ActiveCell.Value = 0**) es falso, VBA salta a la siguiente declaración **ElseIf** y evalúa la condición (**ActiveCell.Value > 0**). Si el valor no es mayor que cero, VBA salta al siguiente **ElseIf** y evalúa la condición **ActiveCell < 0**.

Veamos ahora cómo funciona la estructura **If … Then … ElseIf** en un procedimiento completo:

1. Inserta un nuevo módulo en el proyecto.
2. Llámalo **IfThenElseIf**.
3. Introduce el siguiente procedimiento:

```
Sub TipoNumero()
    Range("A1").Select
    If ActiveCell.Value = 0 Then
        ActiveCell.Offset(0, 1).Value = "Cero"
    ElseIf ActiveCell.Value > 0 Then
        ActiveCell.Offset(0, 1).Value = "Positivo"
    ElseIf ActiveCell.Value < 0 Then
```

```
            ActiveCell.Offset(0, 1).Value = "Negativo"
        End If
    End Sub
```

Dado que es necesario ejecutar el procedimiento `TipoNumero` varias veces para probar cada condición, hagamos que VBA asigne un atajo de teclado temporal a este procedimiento:

4. Abre la ventana **Inmediato** y escribe la siguiente declaración:

   ```
   Application.OnKey "^+y", "TipoNumero"
   ```

   Al presionar **Intro**, VBA ejecuta el método `OnKey` que asigna el procedimiento `TipoNumero` a la combinación de teclas **Ctrl+Mayús+Y**. Este atajo de teclado es sólo temporal, no funcionará cuando reinicies Excel.

   Para asignar la tecla de acceso directo a un procedimiento, utiliza el botón **Opciones** del cuadro de diálogo **Macro**, al que puedes acceder desde la ficha **Programador > Macros** en la ventana de Excel.

5. Ahora dirígete a la ventana de Excel y activa la Hoja2.
6. Escribe un cero (0) en la celda A1 y presiona **Intro**. Luego presiona **Ctrl+Mayús+Y**.
7. VBA llama al procedimiento `TipoNumero` y muestra la palabra “Cero” en la celda B1.
8. Introduce cualquier número mayor que cero en la celda A1 y presiona **Ctrl+Mayús+Y** de nuevo. VBA vuelve a llamar al procedimiento. Al ejecutarlo se evalúa la primera condición, y debido a que el resultado es falso, pasa a la siguiente condición. La segunda condición es verdadera, así que VBA ejecuta la instrucción situada debajo de `Then` y el resto de las condiciones se las salta. Como ya no existen más instrucciones después de `End If`, el procedimiento finaliza. La celda B1 ahora muestra la palabra “Positivo”.
9. Introduce cualquier número menor que cero en la celda A1 y presiona **Ctrl+Mayús+Y**. En este momento, las dos primeras condiciones vuelven a ser falsas, así que VBA va a examinar la tercera condición. Como esta evaluación devuelve “Verdadero”, VBA introduce la palabra “Negativo” en la celda B1.
10. Ahora introduce algo de texto en la celda A1 y presiona **Ctrl+Mayús+Y**.
    La respuesta de VBA es “Positivo”. Sin embargo, no es una respuesta correcta.
    Quizá quieras mostrar la palabra “Texto” cuando se trate de un texto. Para hacer el procedimiento `TipoNumero` más inteligente, necesitas aprender a tomar decisiones complejas utilizando declaraciones anidadas `If … Then`.

5 Estructuras If … Then anidadas

Es posible tomar decisiones más complejas en los procedimientos de VBA colocando una declaración `If … Then` dentro de otra.

Cuando una estructura (`If … Then` o cualquier otra) se encuentra contenida dentro de otra, se dice que están anidadas. El siguiente procedimiento es una versión mejorada de

TipoNumero, creado en la sección anterior. El procedimiento ha sido modificado para mostrar cómo funcionan las sentencias **If … Then** anidadas:

```
Sub IfThenAnidadas()
    Range("A1").Select
    If IsEmpty(ActiveCell) Then
        MsgBox "La celda está vacía."
    Else
        If IsNumeric(ActiveCell.Value) Then
            If ActiveCell.Value = 0 Then
                ActiveCell.Offset(0, 1).Value = "Cero"
            ElseIf ActiveCell.Value > 0 Then
                ActiveCell.Offset(0, 1).Value = "Positivo"
            ElseIf ActiveCell.Value < 0 Then
                ActiveCell.Offset(0, 1).Value = "Negativo"
            End If
        Else
            ActiveCell.Offset(0, 1).Value = "Texto"
        End If
    End If
End Sub
```

Para que el procedimiento sea más fácil de entender, cada declaración **If … Then** se muestra en un color diferente. Se puede ver claramente que el procedimiento utiliza tres bloques **If … Then**.

6 La estructura Select Case

Para evitar estructuras tan complejas como la anterior, que son difíciles de interpretar, puedes utilizar la estructura **Select Case** en su lugar. La sintaxis es la siguiente:

```
Select Case expresión
    Case condición1
        instrucciones si condición1 coincide con la evaluación de expresión
    Case condición2
        instrucciones si condición2 coincide con la evaluación de expresión
    Case condiciónN
        instrucciones si condiciónN coincide con la evaluación de expresión
    Case Else
        instrucciones si no hay valores coincidentes con expresión
End Select
```

Puedes colocar cualquier número de cláusulas **Case** entre **Select Case** y **End Case**. La cláusula **Case Else** es opcional. Úsala cuando consideres que la evaluación de la condición no obtenga ninguno de los otros resultados. En la declaración **Selec Case**, VBA compara cada condición con el valor de la expresión.

Cuando VBA encuentra la expresión **Select Case**, guarda el valor de la expresión.

Luego procede a probar la expresión con todas las condiciones propuestas en cada cláusula **Case** hasta que encuentra una coincidencia. Si ninguna de las cláusulas **Case** contienen la expresión, VBA salta a la línea **Case Else** y ejecuta el resto del código hasta que se encuentre con **End Case**.

Recuerda que **Case Else** es opcional. Si tu procedimiento no contiene esta cláusula y ninguna de las cláusulas **Case** coincide con el valor evaluado, VBA continuará ejecutando las instrucciones siguientes a la línea **End Case**.

Veamos un ejemplo de un procedimiento que utiliza **Select Case**. En el Capítulo 4 aprendiste bastantes detalles sobre la función **MsgBox**, que te permite mostrar un mensaje con uno o más botones. También aprendiste que el resultado de la función **MsgBox** puede asignarse a una variable. Con la función **Select Case**, puedes decidir qué acción tomar según el botón que pulsó el usuario en el cuadro de mensajes.

1. Inserta un módulo nuevo en el proyecto.
2. Llámalo SelectCase.
3. Introduce el siguiente procedimiento:

```
Sub PruebaBotones()
    Dim Pregunta As String
    Dim Boton As Integer
    Dim Titulo As String
    Dim BotonElegido As Integer

    Pregunta = "¿Deseas crear un libro nuevo?"
    Boton = vbYesNoCancel + vbPregunta + vbDefaultButton1
    Titulo = "Libro nuevo"
    BotonElegido = MsgBox(prompt:=Pregunta, _
    Buttons:=Boton, _
    Title:=Titulo)

    Select Case BotonElegido
        Case 6
            Workbooks.Add
        Case 7
            MsgBox "Puedes crear un libro nuevo más tarde."
        Case Else
            MsgBox "Has presionado el botón Cancelar."
    End Select
End Sub
```

La primera parte del procedimiento muestra un mensaje con tres botones (Sí, No y Cancelar). El valor del botón seleccionado por el usuario se asigna a la variable **MiBoton**. Si el usuario hace clic en Sí, a la variable **MiBoton** se le asigna la constante **vbYes** o su correspondiente valor (**6**). Si el usuario selecciona No, a la variable se le asigna **vbNo** o su valor (**7**). Por último, si se pulsa el botón Cancelar, el contenido de la variable será igual a **vbCancel** o al valor **2**. La declaración **Select Case** compara los valores suministrados después de la cláusula **Case** con el valor almacenado en la variable **MiBoton**. Cuando encuentra una coincidencia, se ejecuta la instrucción **Case** apropiada.

El procedimiento **PruebaBotones** funcionará de la misma forma si utilizas las constantes en lugar de los valores de los botones.

```
Select Case BotonElegido
      Case vbYes
          Workbooks.Add
      Case vbNo
          MsgBox "Puedes crear un libro nuevo más tarde."
      Case Else
          MsgBox "Has presionado el botón Cancelar."
End Select
```

Puedes omitir la cláusula **Case Else** introduciendo la estructura de la siguiente forma:

```
Select Case BotonElegido
      Case vbYes
          Workbooks.Add
      Case vbNo
          MsgBox "Puedes crear un libro nuevo más tarde."
      Case vbCancel
          MsgBox "Has presionado el botón Cancelar."
  End Select
```

4. Ejecuta el procedimiento **PruebaBotones** haciendo clic en un botón diferente cada vez.

6.1 Utilizar Is con la cláusula Case

En ocasiones una decisión se toma según un operador relacional (como los que se muestran en la Tabla 1 Operadores relacionales), como, por ejemplo, si la expresión de la prueba es mayor, menor o igual. La palabra clave **Is** permite utilizar una expresión condicional en una cláusula **Case**. Aquí se muestra la sintaxis de la estructura **Select Case** donde se utiliza la palabra **Is**:

```
Select Case expresión
    Case Is condición1
        instrucciones si condición1 coincide con la evaluación de expresión
    Case Is condición2
```

```
            instrucciones si condición2 coincide con la evaluación de expresión
        Case Is condiciónN
            instrucciones si condiciónN coincide con la evaluación de expresión
    End Select
```

Aunque no es necesario utilizar `Case Else` en la estructura `Select Case`, siempre es una buena idea incluirlo, por si acaso la variable que estás evaluando contiene un valor inesperado. La cláusula `Case Else` es el lugar ideal para poner un mensaje de error.

Vamos a comparar algunos números en el siguiente ejemplo:

```
Select Case MiNumero
    Case Is <= 10
        MsgBox "El número es menor o igual que 10."
    Case 11
        MsgBox "Has introducido once."
    Case Is >= 100
        MsgBox "El número es mayor o igual que 100."
    Case Else
        MsgBox "El número se encuentra entre 12 y 99."
End Select
```

Suponiendo que la variable `MiNumero` contenga el valor 120, la tercera cláusula `Case` sería la verdadera y la única instrucción que se ejecuta es `MsgBox "El número es mayor o igual que 100."`.

6.2 Especificar un rango de valores en la cláusula Case

En el ejemplo anterior vimos una declaración `Select Case` sencilla, que utilizaba una expresión en cada cláusula `Case`. Sin embargo, muchas veces puedes querer especificar un rango de valores. Para ello, utiliza la palabra clave `To` entre los valores de las expresiones, como en el siguiente ejemplo:

```
Select Case UnidadesVendidas
    Case 1 To 100
        Descuento = 0.05
    Case Is <= 500
        Descuento = 0.1
    Case 501 To 1000
        Descuento = 0.15
    Case Is > 1000
        Descuento = 0.2
End Select
```

Analicemos el bloque anterior suponiendo que la variable `UnidadesVendidas` tenga el valor 99.

VBA compara el valor de `UnidadesVendidas` con la expresión condicional de las cláusulas `Case`.

La primera y la tercera cláusula `Case` muestran cómo usar un rango de valores en una expresión condicional, usando la palabra clave `To`. Como `UnidadesVendidas` es igual a 99, la condición de la cláusula del primer `Case` es verdadera; por lo tanto, VBA asigna el valor 0,05 a la variable `Descuento`.

¿Qué ocurre con la segunda cláusula `Case`, que también es verdadera? Aunque es obvio que 99 es menor o igual a 500, VBA no ejecuta la instrucción asociada. La razón de esto es que una vez que VBA encuentra una cláusula `Case` con una condición verdadera, no se molesta en mirar el resto de las cláusulas. Salta por encima de ellas ejecutando el procedimiento con las instrucciones que se encuentran por debajo de `End Select`.

Varias condiciones en la cláusula Case

Las comas utilizadas para separar las condiciones dentro de una cláusula `Case` tienen el mismo significado que el operador `Or` que se utiliza en la declaración `If`. La cláusula `Case` es verdadera si al menos una de las condiciones es verdadera.

Anidar significa colocar una estructura dentro de otra. Veremos más ejemplos de anidación con las estructuras de bucle en el Capítulo 7.

6.3 Utilizar varias expresiones en una cláusula Case

Es posible especificar varias condiciones dentro de una cláusula `Case` separando cada condición con una coma, como se muestra en el siguiente ejemplo:

```
Select Case MiMes
    Case "enero", "febrero", "Marzo"
        Debug.Print MiMes & ": Primer trimestre."
    Case "Abril", "Mayo", "Junio"
        Debug.Print MiMes & ": Segundo trimestre."
    Case "Julio", "Agosto", "Septiembre"
        Debug.Print MiMes & ": Tercer trimestre."
    Case "Octubre", "Noviembre", "Diciembre"
        Debug.Print MiMes & ": Cuarto trimestre."
End Select
```

7 Procedimientos VBA con varias condiciones

El procedimiento `IfThen_simple` con el que trabajaste al principio del capítulo, evaluaba solo una condición. Esta declaración, sin embargo, puede tener más de una condición. Para especificar varias condiciones en una instrucción `If … Then`, utiliza los operadores lógicos

And y **Or**, enumerados en la **Tabla 2.** Aquí está la sintaxis de la estructura utilizando el operador **AND**.

```
If condición1 And condición2 Then instrucciones a ejecutar
```

En la sintaxis anterior, tanto **condición1** como **condición2** deben ser verdaderas para que VBA ejecute las instrucciones de la derecha. Por ejemplo:

```
If Ventas = 10000 And Salario < 45000 Then Com = Ventas * 0.07
```

En este ejemplo:

```
Condición1: Ventas = 10000
Condición2: Salario < 45000
```

Cuando se usa **And** en la expresión condicional, ambas condiciones deben ser verdaderas para que VBA pueda calcular la comisión de ventas. Si alguna de estas condiciones es falsa (o ambas) VBA ignora la declaración después de **Then**.

Si solo es suficiente que se cumpla una de las condiciones, puedes utilizar el operador **Or**. Esta es su sintaxis:

```
If condición1 Or condición2 Then instrucciones a ejecutar
```

Si se cumplen la **condición1** o la **condición2**, se ejecutan las instrucciones. El operador **Or** es más flexible, pues solo una de las condiciones tiene que ser verdadera para que VBA ejecute la instrucción que sigue a la palabra clave **Then**.

Veamos un ejemplo:

```
If dep = "S" OR dep = "M" Then comision = 500
```

En este ejemplo, si al menos una condición es cierta, VBA asigna 500 a la variable comisión. Si ambas condiciones son falsas, VBA ignora el resto de la línea. Ahora veamos un ejemplo de un procedimiento completo. Supongamos que al comprar 50 unidades de un producto, se obtiene un 10% de descuento en cada uno si el precio es superior a 30 €. El siguiente procedimiento muestra cómo hacerlo:

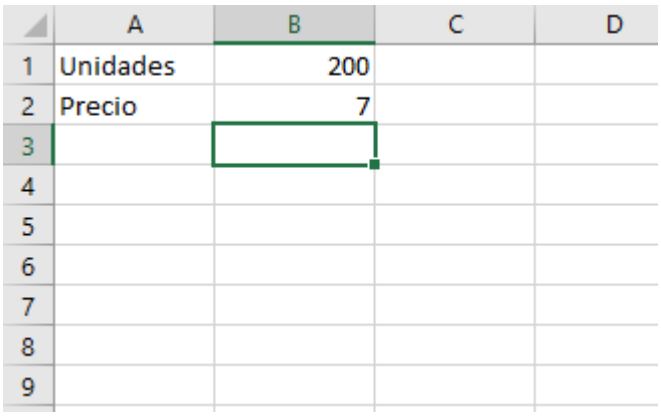
1. Crea un nuevo módulo en el proyecto **Decisiones** y llámalo **IfThenAnd**. A continuación, introduce el siguiente procedimiento.

```
Sub Descuentos()
    Dim precio As Single
    Dim unidades As Integer
    Dim descuento As Single

    Const mens1 = "Para obtener el descuento debes comprar al menos "
    Const mens2 = "El precio deben ser 30 €"
    unidades = Range("B1").Value
    precio = Range("B2").Value

    If precio = 30 And unidades >= 50 Then
```

```
        descuento = (precio * unidades) * 0.1
     Range("A4").Value = "El descuento es: " & descuento
    End If
    If precio = 30 And unidades < 50 Then
        Range("A4").Value = mens1 & 50 - unidades & " unidades más."
    End If
        If precio <> 30 And unidades >= 50 Then
        Range("A4").Value = mens2
    End If
    If precio <> 30 And unidades < 50 Then
        Range("A4").Value = "No coincide ningún criterio."
    End If
End Sub
```

El procedimiento **Descuentos** tiene cuatro bloques **If … Then** que se utilizan para evaluar el contenido de dos variables: precio y unidades. El operador **And** entre las palabras **If** y **Then** permite probar más de una condición. Con el operador **And**, todas las condiciones deben ser verdaderas para que VBA ejecute las instrucciones entre las palabras **Then** y **End If**. Como el procedimiento **Descuentos** se basa en los datos introducidos en la hoja, es más conveniente ejecutarlo desde la ventana de Excel.

2. Cambia a la ventana de Excel y haz clic en la ficha **Programador > Macros**.
3. En el cuadro de diálogo **Macro**, selecciona la macro **Descuentos** y haz clic en el botón **Opciones**.
4. Selecciona el cuadro de texto **Tecla de método abreviado** y presiona las teclas **Mayús + I** para asignar el atajo **Ctrl + Mayús + I** a la macro y haz clic en **Aceptar** para salir del cuadro.
5. Haz clic en **Cancelar** para cerrar el cuadro de diálogo **Macro**.
6. Introduce los datos que se muestran en la imagen Imagen 7.1 en una hoja de Excel.



	A	B	C	D
1	Unidades	200		
2	Precio	7		
3				
4				
5				
6				
7				
8				
9				

Imagen 7.1 Datos de ejemplo en una hoja.

7. Presiona **Ctrl + Mayús + I** para ejecutar el procedimiento **Descuentos**.
8. Modifica los valores de las celdas B1 y B2 para que cada vez que ejecutes el procedimiento se obtenga un resultado diferente.

8 Uso de la lógica condicional en procedimientos Function

Para practicar más con la estructura **Select Case**, usémosla en un procedimiento **Function**. Como recordarás del Capítulo 4, los procedimientos **Function** permiten devolver un resultado a un procedimiento **Sub**. Supongamos que una subrutina debe mostrar un descuento en función del número de unidades vendidas. Puedes obtener el número de unidades vendidas a través del usuario y luego ejecutar una función para averiguar qué descuento debe aplicarse.

1. Introduce el siguiente procedimiento en el módulo **SelectCase**:

```
Sub MostrarDescuento()
    Dim UdVendidas As Integer
    Dim MiDescuento As Single

    UdVendidas = InputBox("Introduce el número de unidades vendidas:")
    MiDescuento = ObtenerDescuento(UdVendidas)
    MsgBox MiDescuento
End Sub
```

2. A continuación, introduce el siguiente procedimiento **Function**:

```
Function ObtenerDescuento(UdVendidas As Integer)
    Select Case unitsSold
        Case 1 To 200
            ObtenerDescuento = 0.05
        Case Is <= 500
            ObtenerDescuento = 0.1
        Case 501 To 1000
            ObtenerDescuento = 0.15
        Case Is > 1000
            ObtenerDescuento = 0.2
    End Select
End Function
```

3. Sitúa el cursor en cualquier lugar del procedimiento **MostrarDescuento** y presiona F5 para ejecutarlo. Ejecútalo varias veces introduciendo diferentes valores para probar cada una de las cláusulas **Case**.
   El procedimiento **MostrarDescuento** pasa el valor almacenado en la variable **UdVendidas** a la función **ObtenerDescuento**. Cuando VBA encuentra la instrucción **Select Case**, comprueba si el valor de la expresión de la cláusula del primer **Case** coincide con el valor almacenado en la variable **UdVendidas**. Si hay coincidencia, VBA asigna un 5% de descuento (0,05) al nombre de la función y luego salta a la función **End Select**. Como no hay más instrucciones que ejecutar dentro del procedimiento de la función, VBA vuelve al procedimiento de llamada (**MostrarDescuento**). Aquí asigna el resultado de la función a la variable **MiDescuento**. La última instrucción muestra el valor del descuento obtenido en un cuadro de mensaje.

9 Resumen

Las declaraciones condicionales de este capítulo permiten controlar el flujo del procedimiento. Al comprobar que una condición es cierta (o no) puedes decidir qué instrucciones deben ejecutarse y cuáles deben saltarse. En otras palabras, en lugar de ejecutar el procedimiento de arriba abajo, línea por línea, puedes hacerlo solo con ciertas líneas de código según cumplan o no las condiciones estipuladas.

Si te preguntas qué tipo de estructura condicional debes utilizar en la toma de decisiones, aquí tienes unas pautas:

- Para introducir solo una condición, **`If`** es la mejor opción.
- Si necesitas decidir cuál de dos acciones realizar, utiliza **`If … Then …Else`**.
- Si el procedimiento requiere dos o más condiciones, utiliza **`If … Then … ElseIf`** o **`Select Case`**.
- Si el procedimiento tiene un gran número de condiciones utiliza la opción **`Select Case`**. Esta estructura es más flexible y fácil de entender que las anteriores.

Algunas decisiones deben ejecutarse varias veces seguidas. Por ejemplo, puedes querer repetir una declaración en cada celda de un rango de una hoja o de varias hojas. En el Capítulo 5 aprenderás cómo realizar uno o varios pasos de forma repetida.

Capítulo 6
Las estructuras de repetición

Ahora que has aprendido cómo dotar a los procedimientos de capacidad de decisión con las estructuras condicionales, es hora de ir un paso más allá.

No todas las decisiones son fáciles. A veces tendrás que realizar varias declaraciones varias veces para llegar a una determinada condición. En otras ocasiones, sin embargo, después de llegar a la decisión, es posible que tengas que ejecutar las declaraciones especificadas cuando una condición sea verdadera o hasta que una condición se convierta en verdadera. En la programación, la realización de tareas repetitivas se llama bucle. VBA tiene varias estructuras de bucle que permiten repetir una secuencia de declaraciones varias veces. En este capítulo aprenderás a hacer un bucle en tu código.

Un bucle es una estructura de programación que hace que una sección del código del programa se ejecute repetidamente. VBA proporciona varias estructuras para implementar bucles en los procedimientos: **`Do … While`**, **`Do … Until`**, **`For … Next`**, **`For … Each`** y **`While … Wend`**.

1 Los bucles Do … While y Do … Until

VBA tiene dos tipos de declaraciones de bucle **`Do`** que repiten una secuencia de declaraciones "mientras que" o "hasta que" una condición sea verdadera. El bucle **`Do … While`** te permite repetir una acción siempre y cuando una condición sea verdadera. Este bucle tiene la siguiente sintaxis.

```
Do While condición
    Declaración1
    Declaración2
    DeclaraciónN
Loop
```

Cuando VBA se encuentra con este tipo de bucle, primero comprueba el valor si la condición es verdadera. Si la condición es falsa, las declaraciones de dentro del bucle no se ejecutan. VBA

continuará ejecutando el procedimiento desde la línea siguiente a la palabra `Loop`. Si la condición es verdadera, las declaraciones dentro del bucle se ejecutan una a una hasta que se encuentra a palabra `Loop`. Esta palabra le dice a VBA que repita todo el proceso de nuevo, siempre y cuando la condición establecida en la línea `Do … While` sea verdadera. Veamos ahora cómo utilizar el bucle `Do … While` en Excel.

En el Capítulo 5 aprendiste a tomar decisiones en función del contenido de una celda. Demos un paso más viendo cómo puedes repetir la misma decisión en varias celdas. La tarea es aplicar el formato Negrita a cualquier celda de una columna, siempre que no esté vacía.

1. Crea un libro nuevo y llámalo "Capítulo 6 – Estructuras repetición.xlsm". Guárdalo en la carpeta Archivos Manual VBA que creaste en el C:\.
2. Abre el editor de VBA y crea un nuevo proyecto. Llámalo Repeticiones.
3. Inserta un módulo nuevo al cual vas a llamar `DoLoop`.
4. Introduce el siguiente procedimiento en el módulo recién creado:

```
Sub AplicaNegrita()
    Do While ActiveCell.Value <> ""
        ActiveCell.Font.Bold = True
        ActiveCell.Offset(1, 0).Select
    Loop
End Sub
```

5. Presiona **Alt + F11** para cambiar a la ventana de Excel y activa la Hoja1. A continuación introduce cualquier dato (texto o números en las celdas **A1:A7**.
6. Cuando hayas terminado de introducir los datos, selecciona la celda A1.
7. Haz clic en la ficha **Programador > Macros**.
   En el cuadro de diálogo, haz doble clic en el procedimiento `AplicaNegrita` (o márcalo y haz clic en **Ejecutar**).
   Al ejecutar el procedimiento, VBA evalúa primero la condición en la línea `Do While ActiveCell.Value <> ""`. La condición dice: Realiza las siguientes declaraciones "mientras" el valor de la celda activa no sea una cadena vacía (`""`). Como has introducido datos en la celda A1 y la has activado (en los pasos 5 y 6), la primera vez que se evalúa la condición, el resultado es verdadero. Por lo tanto, VBA ejecuta la línea `ActiveCell.Font.Bold = True`, que aplica el formato Negrita a la celda activa.
   A continuación, VBA selecciona la celda de la siguiente fila (recuerda la propiedad `Offset` tratada en el Capítulo3). Como la declaración siguiente es la palabra clave `Loop`, VBA vuelve a la primera línea del bucle y comprueba de nuevo la condición. Si la celda activa recién seleccionada no está vacía, VBA repite las sentencias dentro del bucle. Este proceso continúa hasta que se examina el contenido de la celda A8. Como A8 está vacía, la condición es falsa, por lo que VBA se salta las instrucciones que hay dentro del bucle. Como no hay más declaraciones que ejecutar después de la palabra `Loop`, el procedimiento finaliza.

Veamos otra sintaxis del bucle `Do … While`, que permite probar la condición en la parte inferior del bucle de la siguiente manera:

```
Do
    Declaración1
    Declaración2
    DeclaraciónN
Loop While condición
```

Cuando se prueba la condición en la parte inferior del bucle, las declaraciones de éste se ejecutan al menos una vez. Veamos un ejemplo:

```
Sub ClaveSecreta()
    Dim Codigo As String

    Do
        Codigo = InputBox("Introduce el código secreto:")
        If Codigo = "ayudaexcel.com" Then Exit Do
    Loop While Codigo <> "ayudaexcel.com"
End Sub
```

Observa que cuando se evalúa la condición, VBA ya ha ejecutado las declaraciones una vez. Además de colocar la condición al final del bucle, el procedimiento `ClaveSecreta` muestra cómo salir del bucle cuando se cumple la condición. Cuando VBA encuentra la instrucción `Exit Do`, el bucle termina inmediatamente.

Otro tipo de bucle, `Do … Until`, permite repetir una o más declaraciones hasta que una condición se cumple. En otras palabras, `Do … Until` repite un bloque de código siempre que no se cumpla la condición. Esta es la sintaxis:

```
Do Until condición
    Declaración1
    Declaración2
    DeclaraciónN
Loop
```

Utilizando esta sintaxis, ahora puedes reescribir el anterior procedimiento `AplicaNegrita` de la siguiente manera:

```
Sub AplicaNegrita2()
    Do Until IsEmpty(ActiveCell)
        ActiveCell.Font.Bold = True
        ActiveCell.Offset(1, 0).Select
    Loop
End Sub
```

La primera línea del procedimiento dice que se deben realizar las siguientes declaraciones hasta que se alcanza la primera celda vacía. Como resultado, si la celda activa no está vacía, VBA ejecuta las dos declaraciones de dentro del bucle. Este proceso continúa mientras la

condición `IsEmpty(ActiveCell)` sea falsa. Como la condición a evaluar se aplica al principio del bucle, las declaraciones de dentro no se ejecutarán si la primera celda se encuentra vacía. Tendrás la oportunidad de probar este procedimiento en la siguiente sección.

`Do … Until` tiene también una segunda sintaxis, que permite probar la condición al final del bucle:

```
Do
    Declaración1
    Declaración2
    DeclaraciónN
Loop Until condición
```

Si quieres que las declaraciones se ejecuten al menos una vez, coloca la condición en la línea de la declaración `Loop`.

Probemos un procedimiento que borra las hojas vacías de un libro:

1. Introduce el siguiente procedimiento en el Módulo `DoLoop`:

```
Sub EliminarHojasVacias()
    Dim MiRango As Range
    Dim CuentaHojas As Integer

    CuentaHojas = Worksheets.Count

    Do
        Worksheets(CuentaHojas).Select
        Set MiRango = ActiveSheet.UsedRange
        If MiRango.Address = "$A$1" And _
            Range("A1").Value = "" Then
            Application.DisplayAlerts = False
            Worksheets(CuentaHojas).Delete
            Application.DisplayAlerts = True
        End If
        CuentaHojas = CuentaHojas - 1
    Loop Until CuentaHojas = 1
End Sub
```

2. Presiona **Alt + F11** para cambiar a la ventana de Excel e insertar manualmente tres hojas en el libro. En una de las hojas, introduce un texto o número en la celda A1. En otra hoja, introduce algunos datos en las celdas B2:C10. No introduzcas ningún dato en la tercera hoja.
3. Ejecuta el procedimiento `EliminarHojasVacias`. Cuando lo haces, VBA elimina la hoja seleccionada siempre que se cumplan dos condiciones: que la dirección de la propiedad `UsedRange` devuelva la celda A1 y que A1 se encuentre vacía. La propiedad

`UsedRange` se aplica al objeto `Worksheet` y contiene todas las celdas de la hoja de cálculo, tanto vacías como no vacías. Por ejemplo, si introduces algo en las celdas B2 y C10, el "Rango actual" (así es como se denomina) será $A$1:$C$10. El rango actual está limitado por la celda no vacía más alejada de la parte superior izquierda y la más alejada de la parte inferior derecha en una hoja.
Debido a que el libro debe contener al menos una hoja, el código se ejecuta hasta que la variable `CuentaHojas` sea igual a 1. La línea `CuentaHojas = CuentaHojas – 1` asegura que la variable `CuentaHojas` se reduce en uno cada vez que el flujo del código pasa por ella. El valor de `CuentaHojas` se inicia al principio del procedimiento con la siguiente declaración:

```
Worksheets.Count
```

Cuando se elimina una hoja, Excel normalmente muestra un cuadro de diálogo para confirmar la acción. Si deseas que se elimine directamente sin pedir confirmación, utiliza la siguiente declaración:

```
Application.DisplayAlerts = False
```

Cuando finaliza el procedimiento, vuelve a conectar el sistema de mensajes con la siguiente declaración:

```
Application.DisplayAlerts = True
```

Contadores

Un contador es una variable numérica que lleva la cuenta del número de elementos que han sido procesados. En el procedimiento `EliminarHojasVacías` que acabamos de crear, se declara la variable `CuentaHojas` para llevar la cuenta de las hojas que han sido procesadas. Una variable contador debe ser iniciada (cuando se asigna el valor) al principio de la estructura. Esto te asegura conocer siempre el valor exacto antes de empezar a usarla. Un contador puede ser aumentado o disminuido en un número especificado. Más adelante en el capítulo, veremos más ejemplos de uso de contadores.

2 Cómo evitar bucles infinitos

Si no diseñas tu bucle correctamente, obtendrás un bucle infinito, es decir, un bucle que nunca termina. No podrás detener el procedimiento usando la tecla **Esc**. El siguiente procedimiento hace que el bucle se ejecute sin fin porque el programador olvidó incluir una condición:

```
Sub Saludo()
    Do
        MsgBox "¡Hola!"
    Loop
End Sub
```

Para detener la ejecución del bucle infinito debes pulsar la combinación de teclas **Ctrl + Inter**. VBA mostrará un cuadro de diálogo que indica que "La ejecución del código se ha

interrumpido" (como se muestra en la Imagen 2.1). Haz clic en el botón **Finalizar** para cerrar el cuadro.

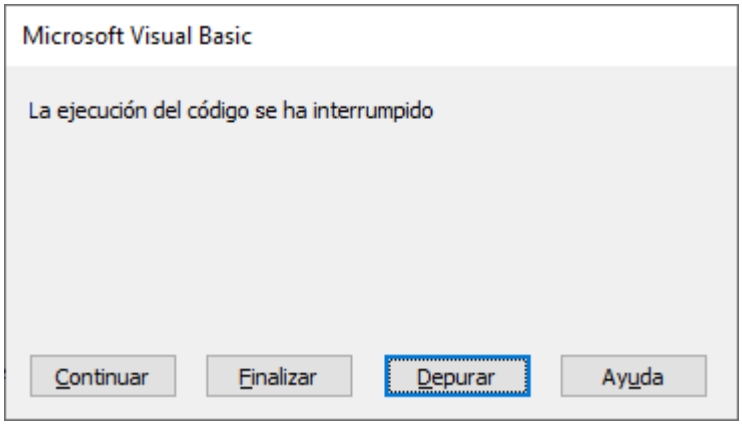


Imagen 2.1 Una forma de detener una macro es presionar Ctrl + Inter.

3 Ejecutar un procedimiento línea a línea

Cuando se ejecutan procedimientos que utilizan estructuras de bucle, a veces es difícil ver si el procedimiento funciona como debería. A veces es muy útil ejecutar el procedimiento "a cámara lenta" para poder comprobar la lógica del programa. Examinemos cómo VBA te permite ejecutar un procedimiento línea a línea.

1. Inserta una nueva hoja en el libro e introduce cualquier dato en las celdas A1:A5.
2. Selecciona la celda A1 y haz clic en la ficha **Programador > Macros**.
3. En el cuadro de diálogo, selecciona la macro **`AplicaNegrita`** y haz clic en el botón **Paso a paso**.
   El editor de VBA resaltará la primera línea del procedimiento **`AplicaNegrita`**, como se muestra en la Imagen 3.1. Observa la flecha indicadora en el margen de la ventana **Código**.

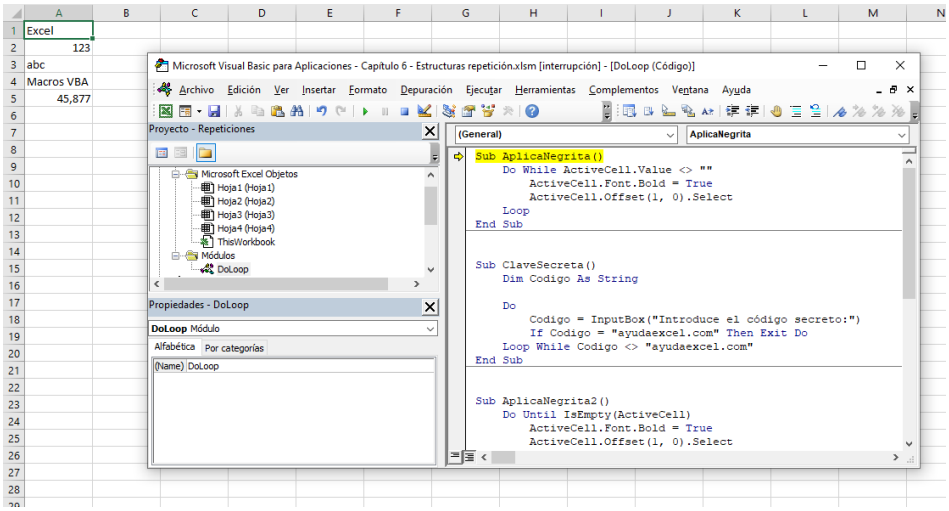


Imagen 3.1 Ejecutando el código del procedimiento línea a línea.

4. Coloca las ventanas una al lado de la otra como se muestra en la Imagen 3.1.
5. Antes de comenzar, asegúrate de que la celda A1 esté seleccionada y que contenga datos.
6. Haz clic en la ventana del título del editor de VBA para mover el foco a esa ventana y luego presiona F8. El resaltado amarillo de la ventana **Código** salta a la línea `Do While ActiveCell.Value <> ""`.
7. Continúa presionando la tecla **F8** mientras te fijas en la ventana **Código** y en la ventana de Excel.

> **Nota**
>
> Encontrarás más información relacionada con la ejecución paso por paso en el Capítulo 9.

4 El bucle While … Wend

El bucle `While … Wend` equivale funcionalmente al bucle `Do … While`. Esta estructura se ha heredado de versiones anteriores de VBA y se incluye únicamente por compatibilidad. Esta es su sintaxis:

```
While condición
    declaración1
    declaración2
    declaración3
Wend
```

La condición se prueba en la parte superior del bucle. Las declaraciones se ejecutan cuando la condición sea verdadera. Una vez que la condición sea falsa, VBA sale del bucle.

Veamos un ejemplo de un procedimiento que utiliza la estructura `While … Wend`. Cambiaremos la altura de todas las celdas no vacías en una hoja.

1. Inserta un módulo nuevo en el proyecto y llámalo `WhileWend`.
2. Introduce el siguiente procedimiento en el módulo:

```
Sub CambiarAltura()
    While ActiveCell <> ""
        ActiveCell.RowHeight = 28
        ActiveCell.Offset(1, 0).Select
    Wend
End Sub
```

3. Haz que se muestre la ventana de Excel e introduce algunos datos en las celdas B1:B4 de cualquiera de las hojas que contiene el libro.
4. Selecciona la celda B1 y haz clic en la ficha **Programador > Macros**.
5. En el cuadro de diálogo selecciona `CambiarAltura` y haz clic en **Ejecutar**.

El procedimiento establece la altura de la fila en 28 si la celda activa no está vacía. La celda de debajo se selecciona utilizando la propiedad **Offset** del objeto **Range**. La instrucción **ActiveCell.Offset(1, 0).Select** le dice a VBA que seleccione la celda que se encuentra una fila hacia abajo (**1**) de la celda activa y en la misma columna (**0**).

5 El bucle For … Next

El bucle **For … Next** se utiliza cuando se sabe de antemano cuántas veces se va a repetir un grupo de declaraciones. Su sintaxis es la siguiente:

```
For contador = comienzo To final [Step incremento]
    Declaración1
    Declaración2
    DeclarraciónN
Next [contador]
```

El código entre corchetes es opcional. **Contador** es una variable numérica que almacena el número de interacciones. **Comienzo** es el número con el que se quiere comenzar a contar y **Final** indica hasta qué número debe llegar la variable **Contador**. Por ejemplo, si quieres repetir cinco veces las declaraciones de dentro del bucle, utiliza el siguiente procedimiento:

```
For contador = 1 To 5
    Instrucciones aquí
Next
```

Cuando VBA se encuentra con la palabra clave **Next**, volverá al principio del bucle y ejecutará de nuevo las instrucciones, siempre que el contador no haya alcanzado el valor final. Tan pronto como el valor del contador sea mayor que el número introducido después de la palabra clave **To**, VBA sale del bucle. Como la variable **Contador** cambia automáticamente después de cada ejecución del bucle, tarde o temprano el valor almacenado excederá el valor especificado. Por defecto, cada vez que VBA ejecuta las sentencias dentro del bucle, el valor de la variable se incrementa en uno. Puedes cambiar esta configuración predeterminada mediante la cláusula **Step**. Por ejemplo, para aumentar el contador de la variable de tres en tres, utiliza el siguiente procedimiento:

```
For contador = 1 To 5 Step 3
    Instrucciones aquí
Next
```

Cuando VBA se encuentra con la instrucción anterior, ejecuta las declaraciones dentro del bucle dos veces. La primera vez en el bucle, el contador es igual a 1. La segunda vez en el bucle, el contador es igual a 4 (3+1). La tercera vez dentro del bucle, el contador es igual a 7 (4+3). Esto hace que VBA se salga de la estructura. Ten en cuenta que el incremento **Step** es opcional y no se especifica a menos que sea un valor distinto a 1. También puedes colocar un número negativo en la cláusula **Step**. VBA disminuirá este valor de la variable **Contador** cada vez que se encuentre con la palabra **Next**.

El nombre de la variable tras la palabra `Next` también es opcional. Sin embargo, es una buena práctica incluirlo para hacer más legible el procedimiento.

¿Cómo puedes usar el bucle `For … Next` en una hoja de Excel? Supongamos que en un informe de ventas quieres incluir únicamente los productos que se vendieron en un mes determinado. Al importar datos de una tabla de Microsoft Access (o de otro software), también obtuviste filas con cantidades iguales a 0. ¿Cómo puedes eliminar rápidamente esas filas con ceros? Aunque hay muchas maneras de resolver este problema, veamos cómo puedes gestionarlo con un bucle `For … Next`:

1. Inserta un nuevo módulo y llámalo **ForNext**.
2. Introduce el siguiente procedimiento en el módulo:

```
Sub EliminaFilasCero()
    Dim totalF As Integer
    Dim r As Integer

    Range("A1").CurrentRegion.Select
    totalF = Selection.Rows.Count
    Range("B2").Select
    For r = 1 To totalF - 1
        If ActiveCell = 0 Then
            Selection.EntireRow.Delete
            totalF = totalF - 1
        Else
            ActiveCell.Offset(1, 0).Select
        End If
    Next r
End Sub
```

   Examinemos el código anterior línea a línea. La dos primeras calculan el número total de filas en el rango actual y almacenan este número en la variable `totalF`. A continuación, VBA selecciona la celda B2 y se encuentra con la palabra clave `For`. Como la primera fila de la hoja contiene los encabezados de las columnas, se disminuye el número total de filas (`totalF - 1`).

   VBA necesitará ejecutar las instrucciones dentro del bucle seis veces. La estructura condicional `If … Then … Else`, que se encuentra anidada dentro del bucle, le dice a VBA que tome una decisión en función del valor de la celda activa. Si el valor es igual a cero, VBA borra la fila y reduce el valor de `totalF` en uno. De lo contrario, la condición es falsa y VBA selecciona la siguiente celda. Cada vez que VBA completa el bucle, salta a la palabra clave `For` para comparar el valor de `r` con el valor de `totalF - 1`.

3. Cambia a la ventana de Excel e inserta una nueva hoja. Introduce los datos que se muestran en la Imagen 5.1.

	A	B
1	Producto	Ventas
2	Manzanas	120
3	Peras	0
4	Plátanos	100
5	Cerezas	0
6	Arándanos	0
7	Fresas	160
8		

Imagen 5.1 Datos de ejemplo.

4. Haz clic en la ficha **Programador > Macros**.
5. En el cuadro de diálogo **Macro**, selecciona el procedimiento `EliminaFilasCero` y haz clic en **Ejecutar**.
   Cuando el procedimiento finaliza, los productos que no tienen ventas han desaparecido.

En parejas

`For` y `Next` siempre deben ir en pareja. Si falta uno de ellos, VBA mostrará el mensaje de error "For sin Next".

6 El bucle For Each … Next

Cuando un procedimiento necesita hacer un bucle a través de todos los objetos de una colección o de todos los elementos de una matriz (las matrices se tratan en el capítulo 7), se debe utilizar el bucle `For Each … Next`. Este bucle no requiere una variable contador. VBA puede calcular por sí mismo cuántas veces debe ejecutarse.

Pensemos, por ejemplo, en una colección de hojas de trabajo de un libro. Para eliminar una hoja de un libro, primero debes seleccionarla y luego hacer clic en la ficha **Inicio > Celdas > Eliminar > Eliminar hoja**. Para dejar solo una hoja de trabajo, debes utilizar el mismo comando varias veces, dependiendo del número total de hojas.

Como cada hoja es un objeto de la colección de hojas del libro, puedes acelerar el proceso de eliminación de hojas utilizando el bucle `For Each … Next`. Este bucle tiene el siguiente aspecto:

```
For Each elemento In grupo
    Declaración1
    Declaración2
    DeclaraciónN
Next [elemento]
```

En la sintaxis anterior, `elemento` es una variable a la que se le asignarán todos los elementos de un conjunto o colección. Esta variable debe ser del tipo Variant para una matriz y del tipo Object para una colección. `Grupo` es el nombre de la colección o matriz.

Veamos ahora cómo usar el bucle `For Each … Next` en un caso real. El siguiente ejemplo elimina algunas hojas del libro:

1. Inserta un nuevo módulo en el proyecto y llámalo **ForEachNext**.
2. Escribe el siguiente procedimiento en el módulo recién creado:

```
Sub EliminaHojas()
    Dim mySheet As Worksheet

    Application.DisplayAlerts = False
    Workbooks.Add
    Sheets.Add After:=ActiveSheet, Count:=3
    For Each MiHoja In Worksheets
        If mySheet.Name <> "Hoja1" Then
            ActiveWindow.SelectedSheets.Delete
        End If
    Next MiHoja
    Application.DisplayAlerts = True
End Sub
```

   VBA creará un nuevo libro de trabajo y agregará tres hojas nuevas después de la hoja activa. A continuación, procederá a eliminar todas las hojas excepto la Hoja1. Observa que la variable `MiHoja` representa un objeto en una colección de hojas. Por lo tanto, esta variable ha sido declarada como objeto (Object). La primera instrucción, `Application.DisplayAlerts = False`, hace que Excel no muestre alertas y mensajes mientras el procedimiento se está ejecutando. El bucle `For Each … Next` pasa por cada hoja de trabajo cuando no sea la Hoja1. Cuando el procedimiento finaliza, el libro solo cuenta con una hoja, la Hoja1.

3. Posiciona el cursor en cualquier parte del procedimiento y presiona **F5** para ejecutarlo.

7 Finalizar un bucle antes de tiempo

A veces puede que no quieras esperar hasta que el bucle termine por sí solo. Es posible que un usuario haya introducido datos erróneos en la hoja y el procedimiento encuentre un error, o tal vez la tarea se ha completado y no hay necesidad de hacer bucles adicionales.

Puedes salir del bucle antes de tiempo sin llegar a la condición que lo finaliza. VBA tiene dos tipos de declaraciones de salida:

- La declaración `Exit For` se usa para terminar un bucle `For … Next` o un `For Each … Next`.
- La declaración `Exit Do` sirve para salir inmediatamente de los bucles `Do`.

El siguiente procedimiento muestra cómo usar la declaración **Exit For** para dejar el bucle **For Each**:

1. Introduce el siguiente procedimiento en el módulo **ForEachNext**:

```
Sub SalidaBucle()
    Dim MiCelda As Variant
    Dim MiRango As Range

    Set MiRango = Range("A1:H10")
    For Each MiCelda In MiRango
        If MiCelda.Value = "" Then
            MiCelda.Value = "Vacía"
        Else
            Exit For
        End If
    Next MiCelda
End Sub
```

   El procedimiento **SalidaBucle** examina el contenido de cada celda del rango especificado (A1:H10). Si la celda activa está vacía, VBA introduce el texto "Vacía". Cuando VBA encuentra la primera celda no vacía, sale del bucle.

2. Crea un nuevo libro de Excel e introduce un valor en cualquier celda del rango A1:H10.
3. Haz clic en la ficha **Programador > Macros**.
4. En el cuadro de diálogo **Macros**, selecciona el procedimiento **SalidaBucle** y haz clic en **Ejecutar**.

8 El bucle Do … While en acción

1. Introduce el siguiente procedimiento en el módulo **DoLoop**:

```
Sub DiezSegundos()
    Dim Parar

    Parar = Now + TimeValue("00:00:10")
    Do While Now < Parar
        Application.DisplayStatusBar = True
        Application.StatusBar = Now
    Loop
        Application.StatusBar = False
End Sub
```

   En este procedimiento, las declaraciones dentro del bucle **Do … While** se ejecutarán siempre que el tiempo devuelto por la función **Now** sea menor que la variable **Parar**.

La variable `Parar` almacena la hora actual más 10 segundos (puedes ver la ayuda online para otros ejemplos de uso de la función `TimeValue`.)

La declaración `Application.DisplayStatusBar` le dice a VBA que encienda la barra de estado. La siguiente instrucción coloca la fecha y hora actuales en la barra de estado. Mientras la hora se muestra durante 10 segundos, el usuario no puede trabajar con el sistema (el puntero del ratón se convierte en un reloj de arena). Una vez que transcurren los 10 segundos (es decir, cuando la condición `Now < Parar` se evalúa como verdadera), VBA abandona el bucle y ejecuta la declaración después de la palabra clave `Loop`. Esta declaración devuelve el mensaje de la barra de estado por defecto ("Listo").

2. Presiona **Alt + F11** para cambiar a la ventana de Excel.
3. Haz clic en **Programador > Macros**. En el cuadro de diálogo **Macro**, haz doble clic en el nombre de la macro `DiezSegundos` (o selecciona el nombre y haz clic en ejecutar). Observa cómo aparecen la fecha y la hora en la barra de estado. Cuando finalizan los 10 segundos, la barra de estado debería volver a "Listo".

9 Bucles y condicionales

Combinemos ahora las estructuras de bucle con algo de lógica condicional para escribir un procedimiento que comprueba si una determinada hoja forma parte de un libro de trabajo.

1. Introduce el siguiente procedimiento en un módulo nuevo:

```
Sub HojaLibro(NombreHoja As String)
    Dim MiHoja As Worksheet
    Dim contador As Integer

    contador = 0
    Workbooks.Add
    Sheets.Add After:=ActiveSheet, Count:=3
    For Each MiHoja In Worksheets
        If MiHoja.Name = NombreHoja Then
            contador = contador + 1
        Exit For
        End If
    Next MiHoja
    If contador = 1 Then
        MsgBox NombreHoja & " existe."
    Else
        MsgBox NombreHoja & " no se ha encontrado."
    End If
End Sub
```

```
Sub EncuentraHoja()
    Call HojaLibro("Hoja4")
End Sub
```

El procedimiento **HojaLibro** utiliza la declaración **Exit For** para asegurar que salimos de bucle tan pronto como el nombre de la hoja pasada en el argumento del procedimiento se encuentre en el libro de trabajo. El procedimiento **EncuentraHoja** se usa para mostrarte cómo llamar a un procedimiento desde otro.

2. Para ejecutar **HojaLibro**, ejecuta el procedimiento **EncuentraHoja**.

10 Resumen

En este capítulo has aprendido a repetir ciertos grupos de declaraciones utilizando bucles en los procedimientos. Mientras trabajabas con varios tipos de declaraciones de bucle, viste cómo cada bucle realiza repeticiones de una manera ligeramente diferente. A medida que vayas ganando experiencia en la programación te resultará más fácil elegir la estructura de control de flujo apropiada para cualquier tarea.

En siguiente capítulo aprenderás qué son las matrices y cómo se usan para trabajar con conjuntos de datos más grandes.

Capítulo 7
Las matrices

En capítulos anteriores, trabajamos con muchos procedimientos de VBA que usaban variables para almacenar información específica sobre un objeto, propiedad o valor. Para cada valor individual que queremos manipular con el procedimiento, declaramos una variable. ¿Pero qué ocurre si tenemos multitud de valores? Si tuviésemos que escribir un procedimiento VBA que fuese capaz de trabajar con decenas (o cientos) de datos, tendríamos que crear suficientes variables para manejarlos todos. ¿Imaginas la pesadilla de almacenar en un procedimiento las tasas de cambio de moneda de todos los países del mundo? Para crear una tabla que contenga los datos necesarios, necesitaríamos al menos tres variables para cada país: Nombre del país, nombre de la moneda y tipo de cambio.

Por suerte, VBA tiene una forma de evitar este problema. Al agrupar las variables relacionadas, los proyectos VBA pueden manejar una gran cantidad de datos con facilidad. En este capítulo aprenderás a manipular listas y tablas de datos con matrices.

1 Qué es una matriz

Una matriz, también llamada *arreglo* o *array* (en inglés) es un tipo especial de variable que representa un grupo de valores similares del mismo tipo de datos (String, Integer, Currency, Date, etc.). Los dos tipos más comunes de matrices son las unidimensionales (también llamadas *listas* o *vectores*) y las matrices bidimiensionales (*tablas*).

Ejemplos de matriz unidimensional son, por ejemplo, una lista de la compra, una lista de los días de la semana o una lista de los empleados de una empresa. Cada elemento de la lista tiene un valor de índice que permite acceder a ese elemento. Por ejemplo, en la siguiente ilustración tenemos una matriz unidimensional de seis elementos indexados de 0 a 5.

(0)	(1)	(2)	(3)	(4)	(5)

Se puede acceder al tercer elemento de esta matriz especificando el índice (2). Por defecto, el primer elemento de un conjunto es el índice cero (0). Puedes cambiar este comportamiento utilizando la declaración `Option Base 1` o codificando explícitamente el límite inferior de la matriz como se explica más adelante en este capítulo.

Todos los elementos de la matriz deben ser del mismo tipo de datos. En otras palabras, una matriz no puede almacenar, por ejemplo, datos String e Integer. A continuación, se muestran dos ejemplos de matriz unidimensional: una que se llama `Ciudades` y es del tipo String, y otra llamada `Lotería`, que almacena números enteros (Integer).

Matriz unidimensional: Ciudades$

Ciudades (0)	Madrid
Ciudades (1)	Barcelona
Ciudades (2)	Bilbao
Ciudades (3)	Sevilla
Ciudades (4)	Valencia
Ciudades (5)	A Coruña

Matriz unidimiensional: Loteria%

Loteria (0)	10
Loteria (1)	15
Loteria (2)	42
Loteria (3)	33
Loteria (4)	18
Loteria (5)	6

Imagen 1.1 Diferentes matrices unidimensionales. Los caracteres $ y % indican el tipo de datos.

Como puedes ver, los contenidos asignados a cada elemento de la matriz coinciden con el tipo de datos de la matriz. Si quieres almacenar diferentes tipos de datos en la misma matriz, debes declarar el conjunto como Variant. Aprenderás a declarar las matrices en la siguiente sección.

Puedes imaginar una matriz bidireccional como una tabla. La posición de cada elemento está determinada por su número de fila y su número de columna. Por ejemplo, una matriz de este tipo es la que contiene las ventas anuales de diferentes productos (las dimensiones serían el nombre del producto y el año). Este es un ejemplo de matriz bidimensional vacía:

(0,0)	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)
(1,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)
(2,0)	(2,1)	(2,2)	(2,3)	(2,4)	(2,5)
(3,0)	(3,1)	(3,2)	(3,3)	(3,4)	(3,5)
(4,0)	(4,1)	(4,2)	(4,3)	(4,4)	(4,5)
(5,0)	(5,1)	(5,2)	(5,3)	(5,4)	(5,5)

Imagen 1.2 Un ejemplo de matriz bidireccional vacía con sus números de índice).

Puedes acceder al primer elemento de la segunda fila de esta matriz bidimensional especificando los índices (1,0). A continuación, se presentan dos ejemplos de una matriz bidimensional: `VentasAnuales`, la cual almacena las ventas anuales de productos, y `CambioMoneda`, que almacena el nombre el país, su moneda y el tipo de cambio.

Matriz bidimensional: VentasAnuales (@)

Producto 1 (0,0)	124,51 € (0,1)
Producto 2 (1,0)	114,25 € (1,1)
Producto 3 (2,0)	25,54 € (2,1)
Producto 4 (3,0)	221,112 € (3,1)

Matriz bidimensional: CambioMoneda

Japón (0,0)	Yen japonés (0,1)	0,008407 (0,2)
Australia (1,0)	Dólar australiano (1,1)	0,55001 (1,2)
Canadá (2,0)	Dólar canadiense (2,1)	0,6418 (2,2)
Noruega (3,0)	Corona (3,1)	0,08768 (3,2)
Estados unidos (4,0)	Dólar estadounidense (4,1)	0,890153 (4,2)

Imagen 1.3 Dos ejemplos de matriz bidimensional.

En estos ejemplos, la matriz **`VentasAnuales`** puede contener un máximo de ocho elementos (cuatro filas por dos columnas) y la matriz **`CambioMoneda`** permitirá un máximo de 15 elementos (cinco filas por tres columnas).

Aunque las matrices de VBA pueden tener hasta 60 dimensiones, a la mayoría de la gente le resulta difícil imaginar dimensiones más allá de tres. Una matriz tridimensional es una matriz de matrices bidimensionales (tablas) donde cada tabla tiene el mismo número de filas y columnas. Una matriz tridimensional se identifica por tres índices: tabla, fila y columna. El primer elemento de una matriz de tres dimensiones tiene el índice (0,0,0).

1.1 Declaración de matrices

Debido a que una matriz es una variable, debes declararla de forma similar a como declaras otras variables (utilizando la palabra clave **`Dim`**, **`Private`** o **`Public`**). Para las matrices de longitud fija, los límites de la matriz se enumeran entre paréntesis detrás del nombre de la variable. Si se declara una matriz de longitud indefinida o dinámica, el nombre de la variable va seguido de un par de paréntesis vacíos.

La última parte de la declaración de una matriz es la definición del tipo de datos que contendrá. Una matriz puede contener cualquiera de los siguientes tipos de datos: Long, Single, Double, Variant, Currency, String, Boolean, Byte o Date. Veamos algunos ejemplos:

`Dim ciudades(5) as String`	Declara una matriz de 6 elementos con índices de 0 a 5.
`Dim loteria(1 to 6) as String`	Declara una matriz de 6 elementos con índices de 1 a 6.
`Dim productos(2 to 11)`	Declara una matriz de 10 elementos con índices de 2 a 11.
`Dim enteros(-3 to 6)`	Declara una matriz de 10 elementos con índices de -3 a 6 (el límite inferior de una

	matriz puede ser 0, 1 o un número negativo).
`Dim MatrizDinamica() as Integer`	Declara una matriz de longitud variable cuyos límites se determinarán en tiempo de ejecución (véanse ejemplos más adelante en este capítulo).

`Dim CambioMoneda(4,2) as Variant`	Declara una matriz con cinco filas y tres columnas.
`Dim VentasAnuales(3, 1) as Currency`	Declara una matriz con cuatro filas y dos columnas.
`Dim Matriz2D(1 to 3, 1 to7) as Single`	Declara una matriz con tres filas indexadas de 1 a 3 y siete columnas indexadas de 1 a 7.

`Dim Cambio(2, 1 to 6, 4) as Variant`	Declara una matriz de tres dimensiones (la primera tiene tres elementos, la segunda tiene seis elementos con índices de 1 a 6 y la tercera tiene cinco elementos).

Cuando se declara una matriz, VBA reserva automáticamente suficiente espacio de memoria. La cantidad de memoria asignada depende del tamaño de la matriz y del tipo de datos. Cuando se declara la matriz unidimensional `Loteria` con seis elementos, VBA reserva 12 bytes (2 bytes para cada elemento del array). Recuerda que el tamaño del tipo de datos Integer es de 2 bytes y, por lo tanto, 2 * 6 = 12. Cuanto más grande sea la matriz, más espacio de memoria se requiere para almacenar los datos. Dado que las matrices pueden consumir mucha memoria e impactar en el rendimiento de tu ordenador, se recomienda que declares matrices con los elementos máximos que crees que utilizarás.

¿Qué es una matriz?

Una matriz es un conjunto de variables que tienen un nombre común. Mientras que una variable típica sólo puede contener un valor, una variable de matriz puede almacenar muchos valores individuales. Se hace referencia a un valor específico de la matriz utilizando el nombre de la matriz y un número de índice.

1.2 Límites superior e inferior de una matriz

Por defecto, VBA asigna cero (0) al primer elemento de la matriz. Por lo tanto, el número 1 representa el segundo elemento, el número 2 representa el tercero, y así sucesivamente. Si la indexación numérica comienza en cero, la matriz unidimensional `Ciudades(5)` contiene seis elementos numerados del 0 al 5. Si prefieres comenzar a contar los elementos de la matriz desde 1, puedes especificar explícitamente un límite inferior utilizando la declaración `Option Base 1`. Esta instrucción debe ser colocada en la parte superior de un módulo VBA antes de cualquier procedimiento Sub. Si no especificas la declaración `Option Base 1` en un procedimiento que utiliza matrices, VBA asume que debe utilizar la instrucción `Option Base 0` y comienza a indexar los elementos desde 0. Si prefieres no utilizar la instrucción `Option Base 1` y aun así hacer que la indexación de la matriz comienza en un número distinto a 0, debes especificar los límites de la matriz al declararla. Se denominan límites sus índices más bajos y más altos. Veamos el siguiente ejemplo:

```
Dim Ciudades(3 To 6) As Integer
```

La instrucción anterior declara una matriz unidimensional con cuatro elementos. Los números encerrados entre paréntesis después del nombre de la matriz especifican los límites inferiores (3) y superiores (6) de la misma. El primer elemento de esta matriz tiene el índice 3, el segundo el 4, el tercero el 5 y el cuarto el 6. Observa la palabra `To` entre los dos números.

1.3 Iniciar y rellenar una matriz

Tras declarar una matriz, debes asignar valores a sus elementos. A menudo esto se denomina “iniciar una matriz”, “inicializar una matriz” o “llenar una matriz”. Los tres métodos que se pueden usar para cargar datos en una matriz se tratan en esta sección.

1.4 Rellenar una matriz con instrucciones individuales

Supongamos que deseas almacenar los nombres de tus seis ciudades favoritas en una matriz unidimensional llamada `Ciudades`. El primer paso es declarar la matriz con la instrucción `Dim`:

```
Dim Ciudades(5) As String
```

o

```
Dim Ciudades$(5)
```

Posteriormente, puedes asignar valores a la variable de la matriz de esta manera:

```
Ciudades(0) = "Madrid"
Ciudades(1) = "Barcelona"
Ciudades(2) = "Bilbao"
```

```
Ciudades(3) = "Sevilla"
Ciudades(4) = "Valencia"
Ciudades(5) = "A Coruña"
```

1.5 Rellenar una matriz con la función Array

La función integrada Array devuelve un conjunto de datos Variant. Como Variant es el tipo de datos por defecto, la cláusula `As Variant` es opcional en la declaración:

```
Dim Ciudades() As Variant
```

Observa que no se especifica el número de elementos.

A continuación, se utiliza la función `Array` para asignar valores a los elementos de la matriz:

```
Ciudades = Array("Madrid", "Barcelona", "Bilbao", _
"Sevilla", "Valencia", "A Coruña")
```

Cuando se utiliza la función `Array` para rellenar la matriz, el límite inferior es 0 o 1 y el límite superior es 5 o 6, dependiendo de la configuración de `Option Base` (ver la sección anterior).

1.6 Rellenar una matriz con un bucle For … Next

La forma más sencilla de aprender a utilizar los bucles para rellenar una matriz es escribiendo un procedimiento que lo haga con un número específico de valores enteros. Veamos el ejemplo:

```
Sub RellenaMatrizNumeros()
    Dim MiMatriz(1 To 10) As Integer
    Dim i As Integer

    'Inicia el generador de números aleatorios
    Randomize
    'Rellena la matriz con 10 números aleatorios entre 1 y 100
    For i = 1 To 10
        MiMatriz(i) = Int((100 * Rnd) + 1)
    Next
    'Imprime los valores de la matriz en la ventana Inmediato
    For i = 1 To 10
        Debug.Print MiMatriz(i)
    Next
End Sub
```

El procedimiento anterior utiliza un bucle `For … Next` para rellenar `MiMatriz` con 10 números aleatorios entre el 1 y el 100. Observa que el procedimiento utiliza la función `Rnd` para generar un número aleatorio. Esta función devuelve un valor entre 0 y 1. Puedes probarlo en la ventana **Inmediato** introduciendo:

```
x=Rnd
?x
```

Antes de llamar a la función **Rnd**, el procedimiento **RellenaMatrizNumeros** utiliza la palabra **Randomize** para iniciar el generador de números aleatorios. Para familiarizarte tanto con la declaración **Randomize** como con la función **Rnd**, te recomiendo consultar la ayuda online de VBA.

2 Práctica con una matriz unidimensional

Tras aprender algunos conceptos básicos de las matrices, escribamos un par de procedimientos VBA para que comiencen a formar parte de tu nuevo conjunto de habilidades. El siguiente procedimiento usa una matriz unidimensional para mostrar mediante programación una lista de seis regiones de España:

1. Crea un nuevo libro de Excel y llámalo "Capítulo 7 – Matrices.xlsm". Guárdalo en la carpeta **Archivos Manual VBA** que creaste en C:\.
2. Cambia el nombre del proyecto VBA a **Matrices**.
3. Inserta un nuevo módulo en el proyecto y llámalo **MatricesEstaticas**.
4. Introduce el siguiente procedimiento en el módulo:

```
' Los índices de las matrices comienzan desde el 1
Option Base 1
Sub CiudadesFavoritas()
    ' Se declara la matriz
    Dim Ciudades(6) As String

    ' Se asignan valores a los elementos de la matriz
    Ciudades(1) = "Madrid"
    Ciudades(2) = "Barcelona"
    Ciudades(3) = "Bilbao"
    Ciudades(4) = "Sevilla"
    Ciudades(5) = "Valencia"
    Ciudades(6) = "A Coruña"
    ' Se muestra la lista de las ciudades
    MsgBox Ciudades(1) & Chr(13) & Ciudades(2) & Chr(13) _
    & Ciudades(3) & Chr(13) & Ciudades(4) & Chr(13) _
    & Ciudades(5) & Chr(13) & Ciudades(6)
End Sub
```

Antes de que comience el procedimiento, se cambia la indexación predeterminada de la matriz. Observa que la declaración **Option Base 1** se encuentra en la parte superior de la ventana del módulo antes de la declaración **Sub**. Esta instrucción le dice a VBA que asigne el número 1 en lugar del 0.

La matriz **Ciudades ()**, del tipo String, se declara con seis elementos. A cada elemento de la matriz se le asigna un valor. La última declaración utiliza la función **MsgBox** para mostrar la lista de ciudades. Cuando se ejecuta esta instrucción en cinco

pasos, los nombres de las ciudades aparecen en líneas separadas en el cuadro de mensajes, como se muestra en la Imagen 2.1. Puedes modificar el orden de los datos visualizados cambiando los valores de índice.

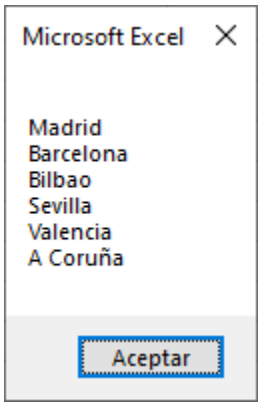


Imagen 2.1 Es posible mostrar los elementos de una matriz unidimensional con la función MsgBox.

5. Posiciona el cursor en cualquier punto dentro del código del procedimiento y presiona **F5** para ejecutarlo.
6. Sigue probando modificaciones en el procedimiento para que muestre, por ejemplo, los nombres de las ciudades en orden inverso (de 6 a 1).

3 Práctica con una matriz de dos dimensiones

Ahora que sabes cómo crear una lista desde código VBA (una dimensión) es hora de mirar más de cerca cómo puedes trabajar con tablas de datos. El siguiente procedimiento crea una matriz bidimensional que contendrá el nombre del país, el nombre de la moneda y el tipo de cambio de tres países:

1. En el módulo **MatricesEstaticas** introduce el siguiente procedimiento:

```
Sub CambioMoneda()
    Dim t As String
    Dim r As String
    Dim Cambio(3, 3) As Variant

    t = Chr(9) ' Tabulador
    r = Chr(13) ' Intro
    Cambio(1, 1) = "Japón"
    Cambio(1, 2) = "Yen"
    Cambio(1, 3) = 0.008407
    Cambio(2, 1) = "México"
    Cambio(2, 2) = "Peso"
    Cambio(2, 3) = 0.03947
    Cambio(3, 1) = "Canadá"
    Cambio(3, 2) = "Dólar"
```

```
    Cambio(3, 3) = 0.6418
    MsgBox "País " & t & t & "Moneda" & t & t & "Cambio €" _
    & r & r _
    & Cambio(1, 1) & t & t & Cambio(1, 2) & t & t _
      & Cambio(1, 3) & r _
    & Cambio(2, 1) & t & t & Cambio(2, 2) & t & t _
      & Cambio(2, 3) & r _
    & Cambio(3, 1) & t & t & Cambio(3, 2) & t & t _
      & Cambio(3, 3) & r & r _
    & "* Cambios de muestra. Sin validez.", , _
    "Cambio de moneda"
End Sub
```

2. Ejecuta el procedimiento.
   Cuando lo hagas, verás un cuadro de mensaje con la información del cambio de moneda, como se muestra en la Imagen 3.1.

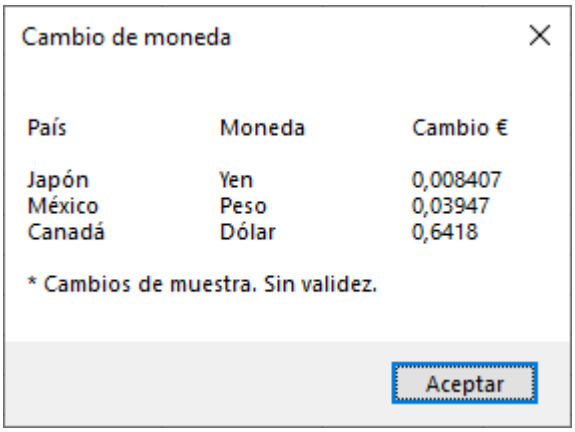


Imagen 3.1 El texto que se muestra en el cuadro de mensaje puede tener un formato personalizado.

4 Práctica con una matriz dinámica

Las matrices introducidas hasta ahora en este capítulo eran estáticas. Una matriz estática de un tamaño especifico. Utiliza una matriz estática cuando sepas de antemano la magnitud que debe tener (el número de elementos). El tamaño de la matriz estática se especifica cuando se declara. Por ejemplo, la declaración `Dim Frutas (9) As String`, declara una matriz estática llamada `Frutas`, que está compuesta por 10 elementos (suponiendo que no se haya cambiado el tipo de indexación a 1). ¿Pero qué ocurre si no estás seguro de cuántos elementos contendrá la matriz? Si tu procedimiento depende de una decisión del usuario, el número de elementos suministrado podría variar cada vez que se ejecute.

¿Cómo puedes asegurarte de que la matriz que declaras no está desperdiciando memoria? Después de declarar una matriz, VBA reserva suficiente memoria como para alojar todos los valores. Si declaras una matriz para que contenga más elementos de los que necesitas, terminarás desperdiciando recursos. La solución a este problema es convertir la matriz en dinámica.

Una matriz dinámica es una matriz cuyo tamaño puede cambiar. Se utiliza cuando el tamaño se determina cada vez que se ejecuta el procedimiento. Se declara colocando paréntesis vacíos después del nombre de la matriz.

```
Dim Frutas () As String
```

Antes de usar una matriz dinámica, debes usar la instrucción **ReDim** para establecer dinámicamente los límites inferiores y superiores de la matriz. Por ejemplo, al principio puedes necesitar almacenar cinco frutas en la matriz:

```
ReDim Frutas(1 To 5)
```

La declaración **ReDim** redimensiona las matrices, informando a VBA del nuevo tamaño de estas. Esta declaración puede ser usada varias veces en el mismo procedimiento. El procedimiento del siguiente ejemplo, cargará dinámicamente en una matriz unidimensional los datos introducidos en una hoja:

1. Inserta un nuevo módulo y llámalo **MatricesDinamicas**.
2. Introduce el siguiente procedimiento en el módulo:

```
Sub CargarMatrizHoja()
    Dim RangoDatos As Range
    Dim MiMatriz() As Variant
    Dim cnt As Integer
    Dim i As Integer
    Dim celda As Variant
    Dim r As Integer
    Dim ultimo As Integer

    Set RangoDatos = ActiveSheet.UsedRange
    ' Obtiene la cuenta de las celdas no vacías (solo texto y _
      números)
    ultimo = RangoDatos.SpecialCells(xlCellTypeConstants, _
      3).Count
    If IsEmpty(RangoDatos) Then
        MsgBox "La hoja está vacía."
        Exit Sub
    End If
    ReDim MiMatriz(1 To ultimo)
    i = 1
    'Rellena la matriz con los datos de la hoja
    'Formatea todos los valores numéricos
    For Each celda In RangoDatos
        If celda.Value <> "" Then
            If IsNumeric(celda.Value) Then
```

```
                MiMatriz(i) = Format(celda.Value, "#,#00.00 €")
            Else
                MiMatriz(i) = celda.Value
            End If
            i = i + 1
        End If
    Next
    'Muestra los valores de la matriz en la ventana Inmediato
    For i = 1 To ultimo
        Debug.Print MiMatriz(i)
    Next
    Debug.Print "Elementos en la matriz: " & UBound(MiMatriz)
End Sub
```

3. Cambia a la ventana de Excel e introduce algunos datos en la Hoja2. Si no existe la hoja, créala. Por ejemplo, introduce tus frutas favoritas en las celdas A1:B6 y algunos números en las celdas D1:D9.
4. Haz clic en la ficha **Programador > Macros**. En el cuadro de diálogo, selecciona **Cargar MatrizHoja** y haz clic en **Ejecutar**.
   Cuando el procedimiento se complete, comprueba los datos en la ventana **Inmediato**. Deberías visualizar los datos que has escrito en la hoja. Los datos numéricos deberían aparecer en formato de moneda.

5 Práctica con funciones de matriz

VBA cuenta con cinco funciones predefinidas: **Array**, **IsArray**, **Erase**, **LBound** y **UBound**. En las siguientes secciones veremos el uso de cada una de ellas en los procedimientos de ejemplo:

5.1 La función Array

La función **Array** permite crear una matriz durante la ejecución del código sin tener que dimensionarlo primero. Esta función siempre devuelve un conjunto de datos Variant. Utilizando la función **Array** puedes introducir rápidamente una serie de valores en una lista.

El procedimiento **InfoCoche** crea una matriz unidimensional de tres elementos de tamaño fijo.

1. Inserta un nuevo módulo en el proyecto actual y llámalo **FuncionArray**
2. Introduce el siguiente procedimiento:

```
Option Base 1
Sub InfoCoche()
    Dim auto As Variant
    auto = Array("Ford", "Sierra", "1999")
    MsgBox auto(2) & " " & auto(1) & ", " & auto(3)
    auto(2) = "4 puertas"
```

```
    MsgBox auto(2) & " " & auto(1) & ", " & auto(3)
End Sub
```

3. Ejecuta el procedimiento.

5.2 La función IsArray

Con la función **IsArray** puedes probar si una variable es una matriz. El resultado será verdadero (True) cuando se trate de una matriz y falso (False) cuando no lo sea. Aquí hay un ejemplo:

1. Inserta un módulo nuevo y llámalo, **FuncionIsArray**.
2. Introduce el siguiente procedimiento en el módulo:

```
Sub EsMatriz()

    ' Se declaran la matriz y otras dos variables
    Dim NombresHojas() As String
    Dim NumeroHojas As Integer
    Dim cnt As Integer

    ' Cuenta las hojas del libro activo
    NumeroHojas = ActiveWorkbook.Sheets.Count
    ' Especifica el tamaño de la matriz
    ReDim NombresHojas(1 To NumeroHojas)
    ' Introduce y muestra los nombres de las hojas
    For cnt = 1 To NumeroHojas
        NombresHojas(cnt) = _
        ActiveWorkbook.Sheets(cnt).Name
        MsgBox NombresHojas(cnt)
    Next cnt
    ' Comprueba si se trata de una matrizy
    If IsArray(NombresHojas) Then
        MsgBox "La variable NombresHojas es una matriz."
    End If
End Sub
```

5.3 La función Erase

Cuando quieras eliminar los datos de una matriz, debes utilizar la función **Erase**. Esta función borra todos los datos que se encuentran en matrices estáticas o dinámicas. Además, libera toda la memoria que se asignó a una matriz dinámica. Si un procedimiento tiene que volver a utilizar la matriz dinámica, debes utilizar la instrucción **ReDim** para especificar el tamaño de la matriz. El siguiente ejemplo muestra cómo borrar los datos de la matriz **Ciudades**.

1. Inserta un módulo nuevo y llámalo **FuncionErase**.
2. Introduce el siguiente código en el módulo:

```
' El índice de las matrices comienzan por 1
Option Base 1
Sub BorraCiudades()
    ' Declara la matriz
    Dim ciudades(1 To 5) As String

    ' Asigna los valores a los elementos de la matriz
    ciudades(1) = "Madrid"
    ciudades(2) = "Barcelona"
    ciudades(3) = "Bilbao"
    ciudades(4) = "Sevilla"
    ciudades(5) = "A Coruña"
    ' Muestra la lista de las ciudades contenidas en la matriz
    MsgBox ciudades(1) & Chr(13) & ciudades(2) & Chr(13) _
    & ciudades(3) & Chr(13) & ciudades(4) & Chr(13) _
    & ciudades(5)
    Erase ciudades
    ' Vuelve a mostrar el contenido de la matriz
    MsgBox ciudades(1) & Chr(13) & ciudades(2) & Chr(13) _
    & ciudades(3) & Chr(13) & ciudades(4) & Chr(13) _
    & ciudades(5)
End Sub
```

Tras borrar la matriz con la función **Erase**, la función **MsgBox** muestra el cuadro de mensaje vacío.

3. Ejecuta el procedimiento **BorraCiudades**.

5.4 Las funciones LBound y UBound

Las funciones **LBound** y **UBound** devuelven números enteros que indican el límite inferior y superior de una matriz.

1. Inserta un nuevo módulo en el proyecto y llámalo **FuncionesLBoundUBound**.
2. Introduce el siguiente procedimiento en el módulo:

```
' El índice de las matrices comienzan por 1
Option Base 1
Sub BorraCiudades()
    ' Declara la matriz
    Dim ciudades(1 To 5) As String

    ' Asigna los valores a los elementos de la matriz
    ciudades(1) = "Madrid"
    ciudades(2) = "Barcelona"
```

```
        ciudades(3) = "Bilbao"
        ciudades(4) = "Sevilla"
        ciudades(5) = "A Coruña"
        ' Muestra la lista de las ciudades contenidas en la matriz
        MsgBox ciudades(1) & Chr(13) & ciudades(2) & Chr(13) _
        & ciudades(3) & Chr(13) & ciudades(4) & Chr(13) _
        & ciudades(5)
        ' Muestra los límites de la matriz
        MsgBox "El límite inferior es: " & LBound(ciudades) _
        & Chr(13) & "El límite superior es: " & UBound(ciudades)
    End Sub
```

3. Ejecuta el procedimiento.

6 Problemas con matrices

Cuando trabajas con matrices, cometer un error es más fácil de lo que piensas. Si intentas asignar más valores de los que hay declarados, VBA mostrará el error "Subíndice fuera del intervalo", como se muestra en la Imagen 6.1.

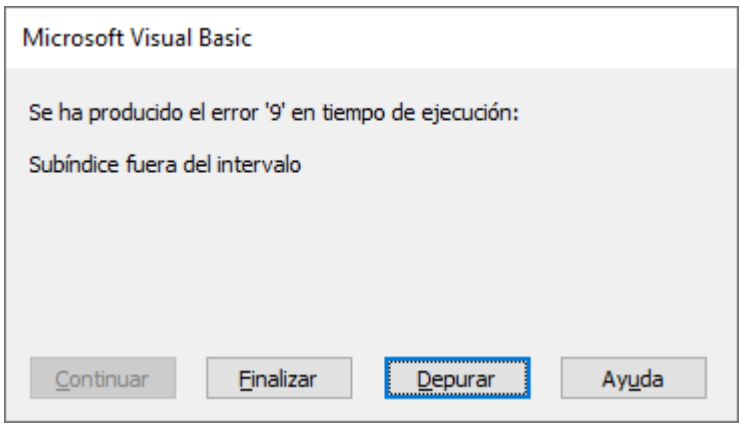


Imagen 6.1 Este error lo produce un intento de introducir un elemento que no cabe en la matriz.

Supongamos que declaras una matriz unidimensional de seis elementos y estás tratando de asignar un valor al elemento número siete. Cuando ejecutas el procedimiento, VBA no puede encontrar el elemento, así que muestra este mensaje de error.

Al hacer clic en el botón **Depurar**, VBA resaltará la línea de código que causó el error. Para arreglar este tipo de error, deberías comenzar por mirar la declaración de la matriz. Una vez que sabes cuántos elementos debe contener la matriz, es fácil averiguar que el culpable es el número de índice que aparece entre paréntesis en la línea de código resaltada. En el ejemplo que se muestra en la Imagen 6.2, una vez reemplazamos la línea de código `ciudades(7) = "A Coruña"` por `ciudades(6) = "A Coruña"` y presionamos **F5** para reanudar el procedimiento, éste se ejecutará según lo previsto.

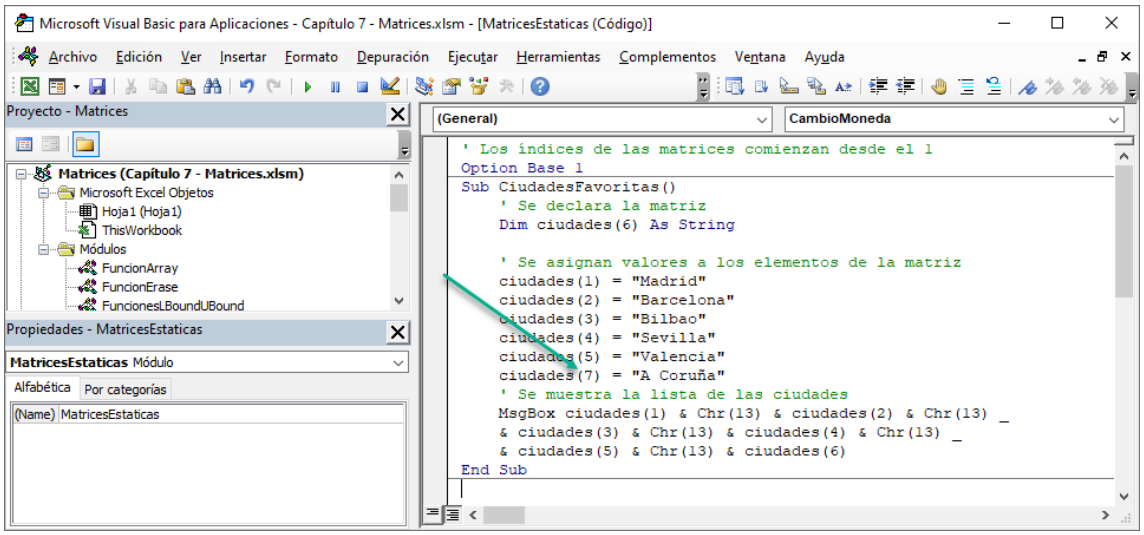


Imagen 6.2 Al hacer clic en el botón Depurar, VBA resalta la instrucción que provocó el error.

Otro error frecuente que puedes encontrar al trabajar con matrices es la "falta de coincidencia de tipos". Para evitar este error, ten en cuenta que todos los elementos de la matriz deben ser del mismo tipo de datos. Si intentas asignar a un elemento de la matriz un valor que entre en conflicto con el tipo de datos declarado con **`Dim`**, obtendrás el error **Type mismatch** durante la ejecución del código. Para almacenar valores de diferentes tipos de datos, declara la matriz como Variant.

7 La palabra ParamArray

Los valores pueden pasarse entre subrutinas y funciones como argumentos obligatorios u opcionales. Si el argumento que se pasa no es absolutamente necesario para que el procedimiento funcione, el nombre del argumento va precedido de la palabra clave **`Optional`**. Sin embargo, a veces no se sabe de antemano cuántos argumentos se necesitarán. Un ejemplo clásico es la suma. Puede que ahora quieras sumar dos números, pero más adelante podrías necesitar 3, 10 o 15 números.

Con la palabra clave **`ParamArray`**, puedes pasar al procedimiento una matriz de cualquier número de elementos.

La siguiente función sumará tantos números como necesites. La función comienza con la declaración de la matriz **`MisNumeros`**. Observa el uso de la palabra **`ParamArray`**. La matriz debe declararse como Variant, y debe ser el último argumento en la definición del procedimiento.

1. Inserta un módulo nuevo en el proyecto y llámalo **`Param_Arrays`**.
2. En el módulo, introduce el siguiente procedimiento:

```
Function SumaArgumentos(ParamArray MisNumeros() As Variant)
    Dim MiSuma As Single
    Dim MiValor As Variant
    For Each MiValor In MisNumeros
```

```
        MiSuma = MiSuma + MiValor
        Next
        SumaArgumentos = MiSuma
    End Function
```

Para probar la función, activa la ventana **Inmediato** y escribe la siguiente instrucción:

```
?SumaArgumentos(1, 23.24, 3, 24, 8, 34)
```

Al presionar **Intro**, VBA devuelve el total de la suma de todos los elementos entre paréntesis: 93,24. Puedes introducir un número ilimitado de argumentos. Para agregar más valores, introdúcelos dentro de los paréntesis y presiona **Intro**. Observa que cada argumento de la función está separado por una coma.

8 Introducción de datos con una matriz

Anteriormente en este capítulo, vimos cómo utilizar varias funciones VBA de matriz. El siguiente procedimiento muestra cómo la función `Array` puede acelerar la introducción de datos:

1. Inserta un nuevo módulo y llámalo `IntroduccionMatriz`.
2. Introduce el siguiente código en el módulo:

```
Sub EncabezadoColumnas()
    Dim encabezado As Variant
    Dim celda As Range
    Dim i As Integer

    i = 0
    encabezado = Array("Nombre", "Apellido", _
    "Categoría", "Salario")
    Workbooks.Add
    For Each celda In Range("A1:D1")
        celda.Formula = encabezado(i)
        i = i + 1
    Next
    Columns("A:D").Select
    Selection.Columns.AutoFit
    Range("A1").Select
End Sub
```

9 Ordenar una matriz con Excel

A todos nos resulta más fácil trabajar con datos ordenados. Algunas operaciones con matrices, como encontrar valores máximos o mínimos, requieren que la matriz esté ordenada. Una vez

ordenada, se puede encontrar el valor máximo asignando el índice superior a la matriz ordenada. Observa el ejemplo:

```
X = MiMatriz(UBound(MiMatriz))
```

El valor mínimo puede obtenerse leyendo el primer valor de la matriz ordenada:

```
x = MiMatriz(1)
```

Pero ¿cómo se puede ordenar una matriz? Esta sección muestra cómo utilizar Excel para ordenar matrices. Una forma fácil de hacerlo es copiando los valores de la matriz en una hoja de trabajo y luego utilizar la herramienta **Ordenar** de Excel. Tras completar esta acción, puedes cargar de nuevo los valores ordenados a la matriz de VBA. Esta técnica es la más simple, pues se puede utilizar la grabadora de macros. Incluso con una matriz que contenga muchos elementos, también es más rápido que el clásico procedimiento de "ordenación por burbuja" que se utiliza habitualmente con matrices.

1. Inserta un nuevo módulo y llámalo `OrdenarMatriz`.
2. Introduce el procedimiento del mismo nombre que se muestra a continuación:

```
Sub OrdenarMatriz()
    Dim MiMatriz() As Integer
    Dim i As Integer
    Dim x As Integer
    Dim y As Integer
    Dim r As Integer
    Dim MiRango As Range

    'Inicia el generador de números aleatorios
    Randomize
    ReDim MiMatriz(1 To 10)
    ' Rellena la matriz con 10 números aleatorios entre 1 y 100
    For i = 1 To 10
        MiMatriz(i) = Int((100 * Rnd) + 1)
        Debug.Print " Valor " & i & ":" & vbTab & MiMatriz(i)
    Next
    ' Vuelca el contenido de la matriz en una hoja
    Worksheets.Add
    r = 1 ' Contador de filas
    With ActiveSheet
        For i = 1 To 10
            Cells(r, 1).Value = MiMatriz(i)
            r = r + 1
        Next i
    End With
```

```
    ' Usa la función Ordenar de Excel con los valores de la hoja
    Set MiRango = ActiveSheet.UsedRange
    With ActiveSheet.Sort
        .SortFields.Clear
        .SortFields.Add Key:=Range("A1"), _
        SortOn:=xlSortOnValues, Order:=xlAscending, _
        DataOption:=xlSortNormal
        .SetRange MiRango
        .Header = xlNo
        .MatchCase = False
        .Apply
    End With
    'Libera la memoria utilizada, borrando la matriz
    Erase MiMatriz
    ReDim MiMatriz(1 To 10)
    ' Carga de nuevo los valores de la hoja a la matriz
    For i = 1 To 10
        MiMatriz(i) = ActiveSheet.Cells(i, 1).Value
    Next
    'Muestra los valores ordenados en la ventana Inmediato
    i = 1
    For i = 1 To 10
        Debug.Print "Valor ordenado: " & MiMatriz(i)
    Next
    'find minimum and maximum values stored in the array
    x = MiMatriz(1)
    y = MiMatriz(UBound(MiMatriz))
    Debug.Print "Valor mínimo=" & x & vbTab; "Valor máximo=" & y
End Sub
```

El procedimiento rellena una matriz dinámica con 10 valores aleatorios e imprime esta matriz en la ventana **Inmediato** y en una hoja nueva. A continuación, los valores introducidos en la hoja se ordenan de forma ascendente utilizando la herramienta **Ordenar** de Excel. Las declaraciones de ordenación se han grabado con la grabadora de macros y luego se han modificado para las necesidades de este procedimiento. Una vez ordenadas, la instrucción `Erase` se utiliza para liberar la memoria utilizada por la matriz dinámica. Antes de recargar la matriz con los valores ordenados, el procedimiento redeclara la variable usando la sentencia `ReDim`. Las últimas declaraciones del procedimiento muestran cómo recuperar los valores mínimo y máximo de la variable de la matriz.

3. Cambia a la ventana de Excel y ejecuta el procedimiento.

10 Resumen

En este capítulo has aprendido a utilizar las matrices en procedimientos complejos que requieren muchas variables. Has trabajado con ejemplos de procedimientos que mostraban cómo declarar y utilizar matrices unidimensionales (listas) y bidimensionales (tablas). Has aprendido la diferencia entre matrices estáticas y dinámicas y has practicado utilizando las cinco funciones de VBA que se usan frecuentemente con matrices: **`Array`**, **`IsArray`**, **`Erase`**, **`LBound`** y **`UBound`**. También has aprendido una nueva palabra clave (**`ParamArray`**) y a ordenar una matriz con Excel.

En el siguiente capítulo aprenderás a usar colecciones en lugar de matrices para manipular grandes cantidades de datos.

Capítulo 8
Las colecciones

Microsoft Excel ofrece un gran número de objetos predefinidos a los que se puede acceder desde los procedimientos VBA para automatizar muchos aspectos de la hoja de cálculo. No estamos limitados de ninguna manera a usar estos objetos. Pero además, VBA nos permite crear nuestros propios objetos y colecciones de objetos, con sus propios métodos y propiedades.

Mientras escribes tus procedimientos, puedes encontrarte con alguna situación en la que no existe una colección predefinida para manejar esa tarea. La solución es crear un objeto de colección personalizado. Ya sabes, desde el capítulo anterior, cómo trabajar con varios elementos de datos mediante el uso de matrices dinámicas y estáticas. Dado que las colecciones tienen propiedades y métodos predefinidos que permiten añadir, eliminar y contar sus elementos, es mucho más fácil trabajar con ellas que con las matrices. En este capítulo aprenderás a trabajar con colecciones de objetos, tanto su declaración como el uso de módulos de clase.

Antes de sumergirnos en la teoría y los ejemplos prácticos del capítulo, deberías familiarizarte con algunos términos:

- **Colección**: Objeto que contiene un conjunto de objetos relacionados entre sí.

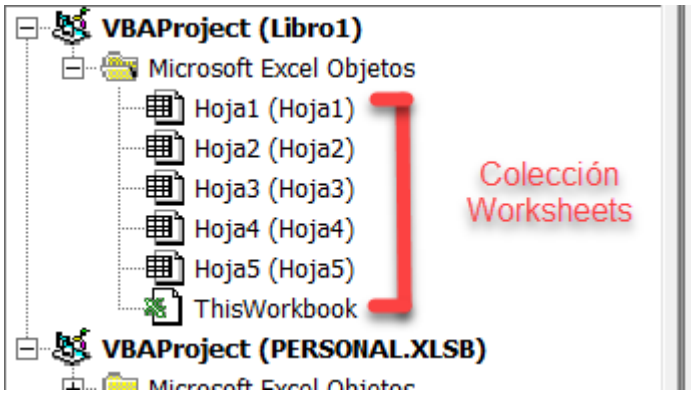


Imagen 10.1 Ejemplo de una colección.

- **Clase**: Se define como un objeto que tiene sus propios métodos y propiedades. La clase actúa como una especie de plantilla de objeto, a partir de la cual se crea una instancia de objeto en tiempo de ejecución.

Imagen 10.2 Una clase de objeto.

- **Instancia**: Un objeto específico que pertenece a una clase se denomina "instancia de la clase". Cuando se crea una instancia, se crea un nuevo objeto que tiene las propiedades y métodos definidos por la clase.
- **Módulo de clase**: Un módulo que contiene la definición de una clase, y que incluye sus propiedades y métodos definidos.
- **Módulo de formulario**: Un módulo que contiene el código VBA para todos los procedimientos de eventos desencadenados en formularios o sus controles. Un módulo de formulario es un tipo de módulo de clase.
- **Evento**: Una acción reconocida por un objeto (por ejemplo, un clic de ratón o una pulsación de tecla), para la cual se puede definir una respuesta. Los eventos pueden ser causados por una acción del usuario,por una declaración VBA o pueden ser activados por el sistema.
- **Procedimiento de eventos**: Un procedimiento que se ejecuta automáticamente en respuesta a un evento iniciado por usuario, el código del programa o el disparo del sistema.

1 Las colecciones de objetos

Una colección es un conjunto de objetos similares. En Microsoft Excel, por ejemplo, todos los libros abiertos pertenecen a la colección **`Workbooks`** y todas las hojas de cálculo (no hojas de gráfico ni hojas de macro 5.0) de un libro son miembros de la colección **`WorkSheets`**. Las colecciones son objetos que contienen otros objetos. Sin importar con qué colección quieras trabajar, las colecciones te permiten hacer lo siguiente:

- Hacer referencia a un objeto específico de la colección usando un valor de índice. Por ejemplo, para hacer referencia al segundo objeto de la colección de hojas de trabajo, utiliza cualquiera de las siguientes declaraciones:

```
Worksheets(2).Select
```

```
Worksheets("Hoja2").Select
```

- Determinar el número de elementos de la colección utilizando la propiedad `Count`. Por ejemplo, al introducir en la ventana **Inmediato** la siguiente declaración, VBA devolverá el número total de hojas del libro actual:

```
?Worksheets.Count
```

.

- Insertar nuevos elementos en la colección utilizando el método `Add`. Por ejemplo, cuando se introduce en la ventana **Inmediato** la declaración:

```
Worksheets.Add
```

VBA insertará una nueva hoja en el libro actual. La colección `Worksheets` ahora contiene un elemento más.

- Recorrer cada objeto de la colección usando el bucle `For Each … Next`. Supongamos que abres un libro de Excel que contiene cinco hojas: "Salario diario", "Salario semanal", "Salario mensual", "Salario anual" y "Comisiones". Para borrar las hojas cuyo nombre contiene la palabra "salario", puedes escribir el siguiente procedimiento:

```
Sub EliminaHojas()
    Dim ws As Worksheet
    Application.DisplayAlerts = False
    For Each ws In Worksheets
        If InStr(ws.Name, "salario") Then
            ws.Delete
        End If
    Next
    Application.DisplayAlerts = True
End Sub
```

La instrucción `Application.DisplayAlerts = False` se utiliza para suprimir algunos avisos y mensajes que Excel muestra mientras el código se está ejecutando. En este caso, queremos suprimir el mensaje de confirmación que Excel muestra cuando se eliminan hojas. La función `InStr` es muy útil para las comparaciones de cadenas, pues permite encontrar un texto dentro de otro. La instrucción `InStr(ws.Name, "salario")` le dice a Excel que determine si el nombre de la hoja (almacenado en la variable de objeto `ws`) contiene la cadena de caracteres "salario".

1.1 Cómo declarar y usar una colección personalizada

Para crear una colección definida por el usuario se debe comenzar declarando una variable de objeto del tipo colección:

```
Dim Nombre_colección as Collection
Set Nombre_colección = New Collection
```

o

```
Dim Nombre_colección As New Collection
```

1.2 Cómo agregar objetos a la colección

Después de declarar el objeto de la colección con la palabra clave **Dim**, puedes insertar nuevos elementos en la colección mediante el método **Add**. La sintaxis de **Add** tiene el siguiente aspecto:

```
Objeto.Add elemento [, clave, antes, después]
```

Únicamente es obligatorio el **objeto** y el **elemento**. El **objeto** es el nombre de la colección, que es el mismo que utilizaste al declararlo. El **elemento** es el objeto que quieres agregar a la colección.

Aunque los otros argumentos son opcionales, son bastante útiles. Es importante entender que a los elementos de una colección se les asignan números automáticamente empezando por 1. Sin embargo, también se les puede asignar un valor clave único. En lugar de acceder a un elemento específico con un índice (1, 2, 3, etc.), puedes asignar una clave en el momento en que se agrega el objeto a la colección. Por ejemplo, si estás creando una colección de hojas personalizadas, podrías usar un nombre de hoja como clave. Para identificar un elemento en una colección de empleados o estudiantes, podrías utilizar sus números de ID como clave.

Para especificar la posición del objeto en la colección, debes usar el argumento **before** o **after** (no ambos). El argumento **before** es el objeto ubicado delante del nuevo objeto añadido. El argumento **after** es el objeto tras el que se añade el objeto.

Los objetos con los que se rellenan las colecciones no tienen que ser del mismo tipo de dato.

El siguiente procedimiento crea una colección personalizada llamada **ColecNotas**. Usaremos esta colección para guardar los comentarios que desees guardar en la hoja de trabajo.

1. Crea un nuevo libro de Excel y guárdalo en la carpeta **Archivos Manual VBA** que creaste en C:\ con el nombre "**Capítulo 8 – Colecciones.xlsm**".
2. Haz clic con el botón derecho del ratón en cualquier celda de la Hoja1 y selecciona **Insertar nota** en el menú contextual.
   Escribe cualquier texto y, a continuación, haz clic fuera del marco de la nota.
   Agrega dos nuevas hojas al libro y utiliza la misma técnica para agregar dos notas en la **Hoja2**. Introduce un texto diferente en cada comentario. Agrega otra nota en la **Hoja3**. Ahora deberías tener cuatro notas en tres hojas del libro.
3. Haz clic en la ficha **Archivo** y selecciona **Opciones**. En el cuadro **Opciones de Excel**, en la sección general y en el área llamada **Personalizar la copia de Microsoft Office**, deberías ver un cuadro te texto con tu nombre. Elimínalo e introduce el nombre "Juan Romero". A continuación, haz clic en **Aceptar** para cerrar el cuadro **Opciones**. Ahora, introduce un comentario en cualquier parte de la **Hoja2** y otro en la **Hoja3**. En las últimas notas debe aparecer como autor "Juan Romero". Cuando finalices de introducir las notas, vuelve a la ventana **Opciones de Excel** y cambia el cuadro de texto con el nombre de usuario. Introduce de nuevo tu nombre.
4. Abre el editor de VBA y renombra el proyecto a **ColClasObj**.

5. Agrega un nuevo módulo y llámalo **MiColección**.
6. En el módulo, introduce el siguiente procedimiento:

```
Sub ObtenerComentarios()
    Dim sht As Worksheet
    Dim Notas As New Collection
    Dim MiNota As Comment
    Dim i As Integer
    Dim t As Integer
    Dim strNombre As String

    strNombre = InputBox("Introduce el nombre del autor:")
    For Each sht In ThisWorkbook.Worksheets
        sht.Select
        i = ActiveSheet.Comments.Count
        For Each MiNota In ActiveSheet.Comments
            If MiNota.Author = strNombre Then
                MsgBox MiNota.Text
                If Notas.Count = 0 Then
                    Notas.Add Item:=MiNota, Key:="first"
                Else
                    Notas.Add Item:=MiNota, Before:=1
                End If
            End If
        Next
        t = t + i
    Next
    If Notas.Count <> 0 Then MsgBox Notas("first").Text
    MsgBox "Comentarios en el libro: " & t & Chr(13) & _
    "Comentarios en la colección: " & Notas.Count
    Debug.Print "Comentarios de " & strNombre
    For Each MiNota In Notas
        Debug.Print Mid(MiNota.Text, Len(MiNota.Author) + 2, _
        Len(MiNota.Text))
    Next
End Sub
```

El procedimiento anterior comienza declarando el objeto de colección personalizado llamado **Notas**. A continuación, se solicita el nombre de un autor para recorrer después todas las hojas del libro intentando localizar los comentarios del nombre introducido. Sólo se añaden a la colección personalizada los comentarios introducidos por el autor especificados.

El procedimiento asigna una clave al primer comentario y luego añade los comentarios restantes a la colección colocándolos antes del último comentario (observa el uso del argumento `before`). Si la colección tiene al menos un comentario, el procedimiento muestra un cuadro de mensaje con el texto del comentario que se identificó con el argumento `clave`. Observa cómo se utiliza el argumento `clave` al hacer referencia a un elemento de la colección. El procedimiento imprime entonces en la ventana **Inmediato** todos los comentarios incluidos en la colección.

Las funciones de texto `Mid` y `Len`, se utilizan para obtener solo el texto del comentario sin el nombre del autor. A continuación, se muestra un cuadro de texto con el número total de comentarios del libro y el número de comentarios del autor especificado.

7. Ejecuta el procedimiento `Obtener comentarios` otras dos veces, introduciendo nombres diferentes en el `InputBox`. Ten en cuenta que VBA diferencia entre mayúsculas y minúsculas.
8. Comprueba los resultados de los procedimientos en la ventana **Inmediato**.

1.3 Cómo eliminar objetos de una colección personalizada

Eliminar un elemento de una colección personalizada es tan fácil como agregarlo. Para quitarlo, utiliza el método `Remove` con la siguiente sintaxis:

```
Objeto.Remove elemento
```

`Objeto` es el nombre de la colección, y `elemento` es el objeto que quieres quitar de la colección.

Para mostrar el proceso de eliminación de un elemento de una colección, modifiquemos el procedimiento `ObtenerComentarios` de la sección anterior. Al final del procedimiento mostraremos uno por uno el contenido de los elementos que se encuentran en ese momento en la colección `Notas` y preguntaremos al usuario si el elemento debe ser eliminado de la colección.

1. Agrega las siguientes líneas en la parte superior, donde declaraste las variables:

   ```
   Dim Respuesta as Integer
   Dim miID As Integer
   ```

   La primera línea declara la variable `Respuesta`, que se usará para guardar el resultado de la función `MsgBox`. La segunda línea declara la variable `miId`, utilizada para almacenar el número de índice del objeto `Colección`.

2. Localiza la siguiente línea en el procedimiento:

   ```
   For Each MiNota In Notas
   ```

   Escribe encima de ella la siguiente línea:

   ```
   MiId = 1
   ```

3. Localiza la siguiente línea en el procedimiento:

   ```
   Debug.Print Mid(MiNota.Text, Len(MiNota.Author) + 2, _
   ```

```
Len(MiNota.Text))
```

e introduce debajo de ella las siguientes instrucciones:

```
Respuesta = MsgBox("¿Eliminar este comentario?" & Chr(13) _
& Chr(13) & MiNota.Text, vbYesNo + vbQuestion)
If Respuesta = 6 Then
Notas.Remove Index:=miID
Else
miID = miID + 1
End If
```

4. Introduce las siguientes instrucciones justo antes del final del procedimiento (en la fila anterior a `End Sub`).
5. Ejecuta el procedimiento `ObtenerComentarios2` y elimina uno de los comentarios cuando aparezca el cuadro de mensaje. Ten en cuenta que este procedimiento solo manipula la colección personalizada de comentarios y no los comentarios reales que introdujiste el libro. Por tanto, tras eliminar los comentarios con el código anterior, los comentarios reales seguirán presentes en el libro. Para eliminar todos los comentarios del libro, ejecuta el siguiente código:

```
Sub EliminaComentariosLibro()
Dim miComentario As Comment
Dim sht As Worksheet

    For Each sht In ThisWorkbook.Worksheets
        For Each miComentario In sht.Comments
            miComentario.Delete
        Next
    Next
End Sub
```

Reubicar índices de colecciones

Las colecciones se reindexan automáticamente cuando se elimina un objeto de ellas. Por lo tanto, para eliminar todos los objetos de una colección personalizada, puedes utilizar 1 para el argumento clave, como en el siguiente ejemplo:

```
Do While MiColeccion.Count > 0
    MiColeccion.Remove Index:=1
Loop
```

2 Creación y uso de objetos personalizados

El menú **Insertar** del editor de VBA tiene dos opciones de módulos: **Módulo** y **Módulo de clase**. Hasta ahora has utilizado módulos estándar para crear procedimientos de subrutinas y funciones. En este capítulo comenzarás a utilizar los módulos de clase para crear una clase personalizada llamada `ClsComponentes` y aprenderás a definir sus propiedades y métodos.

Antes de poder crear objetos personalizados, necesitas conocimientos básicos de lo que es una clase. Si recuerdas el principio de este capítulo, verás que se describió una clase como una "plantilla de objetos". Una analogía frecuentemente utilizada es comparar una clase de objeto con un cuchillo para cortar pan. Al igual que el cuchillo define el aspecto que tendrá el pan después de cortarlo, la definición de clase determina cómo debe verse y comportarse un objeto. Antes de poder usar una clase de objeto, primero debes crear una nueva instancia de esa clase. Las instancias de objetos son las barras de pan. Cada instancia de objeto tiene las características (propiedades y métodos) definidas por su clase. De la misma forma que puedes cortar muchas barras de pan utilizando el mismo cuchillo, puedes crear múltiples instancias de una clase. También puedes cambiar las propiedades de cada instancia de una clase de forma independiente a cualquier otra instancia de la misma clase.

Un módulo de clase te permite definir tus propias clases personalizadas, con propiedades y métodos personalizados. Recuerda que una propiedad es un atributo de un objeto que define una de sus características, como la forma, la posición, el color, el título, etc. Puedes crear las propiedades de tus objetos personalizados escribiendo procedimientos de propiedades en un módulo de clase. Hay tres tipos de procedimientos de propiedad: `Property Get`, `Property Let` y `Property Set`.

Aprenderás a trabajar con procedimientos de propiedades en el siguiente ejercicio.

Un método es una acción que el objeto puede realizar. Los métodos de objeto también se crean en un módulo de clase escribiendo subrutinas o procedimientos de función. El trabajo con módulos de clase es un tema avanzado y se tratará en este capítulo.

El siguiente ejercicio te introducirá en el proceso de creación de un objeto personalizado llamado `ClsComponentes`. Este objeto contendrá información sobre un componente de un ordenador. Contendrá cuatro propiedades que almacenarán la información sobre el componente: tipo de componente, fabricante, modelo y precio. También tendrá un método que te permitirá modificar el precio. La información sobre el componente que utilizarás en el proyecto está incluida en el archivo de texto "Capítulo 8 – Componentes.txt".

Como puedes ver en la Imagen 2.1, el archivo de texto contiene varias líneas (registros).

Los datos entre comillas se tratan como un solo campo. Los campos están delimitados por comas. Este tipo de archivos de texto a menudo se llaman "archivos delimitados" o "archivos planos". Para completar correctamente este ejercicio necesitas saber que en los archivos de texto plano los datos se recuperan en el mismo orden en que se almacenan.

Los archivos planos pueden abrirse en modo de entrada, salida o anexión. En este proyecto utilizarás el modo de entrada, que te permitirá leer los datos del archivo en las propiedades del objeto personalizado. Como el archivo contiene datos sobre varios componentes, también

aumentarás tu comprensión sobre las colecciones al leer los datos del archivo de texto en la colección de objetos `ClsComponentes` y al manipular estos objetos. Así que, empecemos.

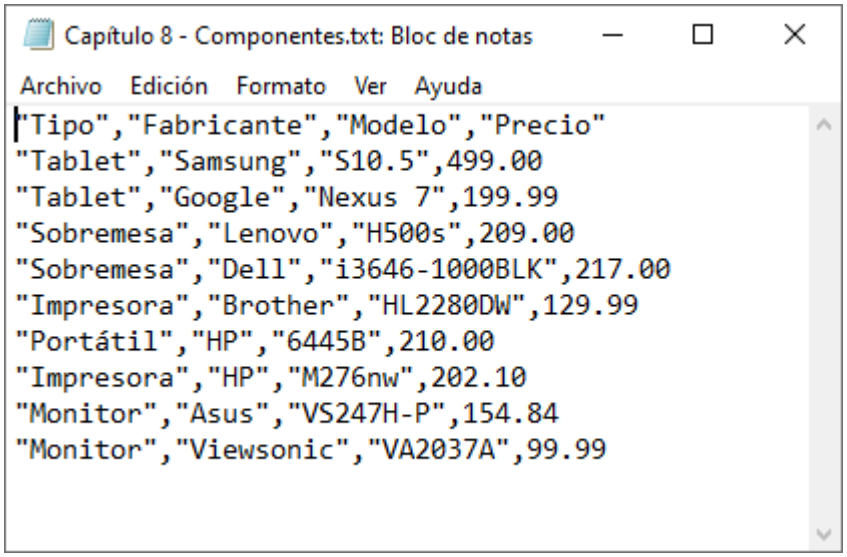


Imagen 2.1 El archivo de texto contiene los datos para la clase de objeto clsComponentes personalizada.

3 Ejercicio Clases de objeto

1. Abre el libro "Capítulo 8 – Colecciones.xlsm" e inserta un módulo de clase haciendo clic en el menú **Insertar – Módulo de clase**.
2. Haz clic en el módulo y renómbralo de la misma forma que los módulos anteriores. Llámalo `ClsComponentes`.

Nomenclatura de los módulos de clase

Cada vez que crees un módulo de clase, dale un nombre significativo. El nombre que elijas debe ser fácil de recordar e identificar. Por regla general, el nombre suele tener como primera letra una "C" mayúscula o comenzar por "cls".

3.1 Declarando las variables

Después de agregar y renombrar el módulo de clase, el siguiente paso es declarar las variables que contendrán los datos que quieres almacenar en el objeto. A cada elemento de datos que quieres almacenar en un objeto, se le debe asignar una variable. Las variables en una clase se llaman "miembros de datos" y se declaran con la palabra `Private`. Esta palabra asegura que las variables estarán disponibles solo dentro de ese módulo de clase. Usar la palabra `Private` en lugar de `Dim`, impide que otras partes de la aplicación hagan referencia a ellos. Solo los procedimientos dentro del módulo de clase en el que se definieron pueden modificar el valor de las variables.

Como el nombre de una variable también puede servir como nombre de propiedad, el uso de nombres con significado propio toma más protagonismo. Es habitual que se preceda el nombre de la variable con `m_` para indicar que son miembros de una clase de datos.

Continuemos con nuestro ejercicio declarando miembros de datos (variables) para nuestra clase **ClsComponentes**:

1. Escribe las siguientes líneas de código para declarar las propiedades:

   ```
   Private m_Tipo As String
   Private m_Fabricante As String
   Private m_Modelo As String
   Private m_Precio As Currency
   ```

   Observa que el nombre de cada miembro comienza con el prefijo **m_**.

3.2 Definiendo las propiedades de la clase

Al declarar las variables con la palabra **Private**, garantizamos que no se pueda acceder directamente desde el exterior del objeto. Esto significa que los procedimientos de fuera del módulo de clase no podrán establecer o leer datos almacenados en esas variables. Para permitir que otras partes de la aplicación VBA establezcan o extraigan los datos de los componentes, debes añadir procedimientos especiales de propiedad al módulo **ClsComponentes**.

Existen tres tipos de procedimientos de propiedad:

- **Property Let**: permite que otras partes de la aplicación establezcan el valor de una propiedad.
- **Property Get**: permite que otras partes de la aplicación obtengan o lean el valor de una propiedad.
- **Property Set**: se utiliza en lugar de **Property Let** cuando se establece la referencia a un objeto.

Los procedimientos de propiedad se ejecutan cuando la propiedad de un objeto necesita ser establecida o recuperada. La propiedad **Property Get** puede tener el mismo nombre que el procedimiento **Property Let**.

Deberías crear procedimientos de propiedad para cada propiedad del objeto que se pueda acceder desde otras parte de la aplicación VBA. El más sencillo y fácil de entender de los tres tipos de declaraciones de propiedad es **Property Get**.

Examinemos la sintaxis de los procedimientos de propiedades echando un vistazo al procedimiento **Property Get** llamado **Tipo**. Como norma, los procedimientos de propiedad tienen las siguientes partes:

- Una declaración de procedimiento que especifica el nombre de la propiedad y el tipo de datos:

  ```
  Property Get Tipo As String.
  ```

  Tipo es el nombre de la propiedad y **As String** determina el tipo de datos que devuelve la propiedad.
- Una declaración de asignación como la que se usa en un procedimiento **Function**:

```
Tipo = m_Tipo
```

`Tipo` es el nombre de la propiedad, y `m_Tipo` es la variable que contiene el valor de la propiedad que se quiere recuperar o establecer. La variable `m_Tipo` debe definirse con la palabra clave `Private` en la parte superior del módulo de clase.

- Las palabras `End Property`, que especifican el final del procedimiento de propiedad.

```
Property Get Tipo() As String
    Tipo = m_Tipo
End Property
```

3.3 Escribiendo los procedimientos de propiedad

La clase `ClsComponentes` tiene cuatro propiedades (`Tipo`, `Fabricante`, `Modelo` y `Precio`) que deben ser volcadas a un procedimiento que escribirás más tarde. Como este procedimiento necesitará leer un archivo de datos y luego escribir dichos datos en la colección de objetos `ClsComponentes`, el siguiente paso requiere escribir los procedimientos necesarios de `Property Get` y `Property Let`.

1. Escribe los procedimientos `Property Get` y `Property Let` en el módulo `ClsComponentes`, justo debajo de la declaración de las variables.

```
' Procedimientos de propiedad
Property Get Tipo() As String
    Tipo = m_Tipo
End Property
Property Let Tipo(ByVal aType As String)
    m_Tipo = aTipo
End Property
Property Get Fabricante() As String
    Fabricante = m_Fabricante
End Property
Property Let Fabricante(ByVal aMake As String)
    m_Fabricante = aFabricante
End Property
Property Get Modelo() As String
    Modelo = m_Modelo
End Property
Property Let Modelo(ByVal aModelo As String)
    m_Modelo = aModelo
End Property
Property Get Precio() As Currency
    Precio = m_Precio
End Property
Property Let Precio(ByVal aPrecio As Currency)
```

```
        m_Precio = aPrecio
    End Property
```

Observa que cada tipo de información del componente requiere un procedimiento **Property Get** aparte. Cada uno de los procedimientos **Property Get** devuelve el valor actual de la propiedad. El procedimiento **Property Get** es como un procedimiento de función. Como en los procedimientos **Function**, los procedimientos **Property Get** contienen una declaración de asignación. Si recuerdas, en el capítulo 4, para que una función devuelva un valor, debes asignar este valor al nombre de la función.

Salir inmediatamente de un procedimiento Property

Al igual que las instrucciones **Exit Sub** y **Exit Function** permiten salir antes de procedimientos **Sub** y **Function**, la instrucción **Exit Property** te ofrece la forma de salir inmediatamente de un procedimiento **Property**. La ejecución del programa continuará con las declaraciones que siguen a la declaración que llamó al procedimiento **Property Get**, **Property Let** o **Property Set**.

Además de recuperar los valores almacenados en los miembros de datos (variables **Private**) con los procedimientos **Property Get**, introdujiste los procedimientos **Property Let** correspondientes para permitir que otras partes de la aplicación cambien los valores de estas variables según sea necesario. Puedes hacer que una propiedad sea de solo lectura si omites el procedimiento **Property Let** correspondiente.

Los procedimientos **Property Let** requieren al menos un parámetro que especifique el valor que quieres asignar a la propiedad. Este parámetro se puede especificar como valor (ver palabra clave **ByVal** en el procedimiento mostrado anteriormente) o como referencia (**ByRef** es el valor por defecto). Si necesitas un repaso del significado de estas palabras clave, consulta el apartado **Paso de argumentos por referencia y valor** del Capítulo 4.

El tipo de datos pasado al procedimiento **Property Let** debe tener el mismo tipo de datos que el valor devuelto por el procedimiento **Property Get** con el mismo nombre. Observa que los procedimientos **Property Let** tienen el mismo nombre que los procedimientos **Property Get** que se mostraron en la sección anterior.

3.4 Los métodos de clase

Además de las propiedades, los objetos suelen tener uno o más métodos. Un método es una acción que el objeto puede realizar. Los métodos permiten manipular los datos almacenados en módulos de clase. Los métodos se crean con los procedimientos **Sub** o **Function**. Para hacer que un método esté disponible fuera del módulo de clase, utiliza la palabra **Public** delante de la definición de la subrutina o la función.

Ámbito de los procedimientos Property

Puedes colocar la palabra clave **Public**, **Private** o **Static** antes del nombre de un procedimiento **Property** para definir su alcance o ámbito. Por ejemplo, para indicar que el procedimiento **Property Get** es accesible a otros procedimientos en todos los módulos, utiliza el siguiente formato de declaración:

```
Public Property Get ClsComponentes() As String
```

Para que el procedimiento sea accesible solo a otros procedimientos en el módulo en el que se declara, utiliza el siguiente formato de declaración:

```
Private Property Get Modelo() As String
```

Para mantener las variables locales del procedimiento **Property Get** en las llamadas al procedimiento, utiliza el siguiente formato de declaración:

```
Static Property Get Fabricante() As String
```

Si no se especifican explícitamente las palabras **Public** o **Private**, los procedimientos **Property** son públicos por defecto. Además, si no se utiliza la palabra **Static**, los valores de las variables locales no se conservan entre llamadas a los procedimientos.

Como se comentó al principio del ejercicio, la clase **ClsComponentes** tiene un método que te permite calcular nuevos precios. Supongamos que el precio del componente disminuye en un porcentaje o en una cantidad fija. Continuemos con el ejercicio escribiendo el método de clase que realiza este cálculo:

1. Escribe la siguiente función en el módulo de clase **ClsComponentes**:

```
' Función para calcular el nuevo precio
Public Function NuevoPrecio(tipoDescuento As Integer, _
    precioActual As Currency, _
    cantidad As Long) As Currency
    If cantidad >= precioActual Then
        NuevoPrecio = precioActual
        Exit Function
    End If
    Select Case tipoDescuento
        Case 1 ' por porcentaje
            If cantidad > 50 Then
                cantidad = 50
            End If
            NuevoPrecio = precioActual - ((precioActual * _
            cantidad) / 100)
        Case 2 ' por cantidad
```

```
                NuevoPrecio = precioActual - cantidad
        End Select
    End Function
```

La función `NuevoPrecio`, definida como `Public` en un módulo de clase, sirve como método para la clase `ClsComponentes`. Para calcular un nuevo precio, un procedimiento VBA desde fuera del módulo de clase debe pasar tres argumentos: `tipoDescuento`, `precioActual` y `cantidad`. El argumento `tipoDescuento` especifica el tipo de cálculo. Puedes elegir entre reducir el precio del producto en un 5 % o en 5 €. La primera opción reducirá el precio en el porcentaje especificado y la segunda opción restará la cantidad especificada al precio actual.

El argumento `precioActual` es la cifra del precio actual de un componente, y `cantidad` determina el valor por el cual el precio debe cambiarse.

3.5 Creando una instancia de una clase

Ya está definida la clase `ClsComponentes`. Cada vez que quieras definir una clase debes seguir todos estos pasos en un módulo de clase. El nombre de la clase es el nombre del módulo. Una clase es un modelo a partir de la cual se pueden crear objetos. La clase especifica las propiedades y métodos, que serán comunes para todos los objetos creados a partir de esa clase.

Después de definir la clase, es hora de crear los objetos basados en esa clase. Este proceso tiene lugar en un módulo estándar. Se empieza declarando una variable de objeto. Si el nombre del módulo de clase es `ClsComponentes`, declara una variable de tipo `Clscomponentes` y establécela en una nueva instancia de la clase, de esta forma:

```
Dim componente As ClsComponentes
Set componente = New ClsComponentes
```

También es posible combinar las dos declaraciones en una sola del siguiente modo:

```
Dim componente As New ClsComponente
```

La variable `componente` representa una referencia a un objeto de la clase `ClsComponentes`. Puedes ponerle el nombre que quieras a la variable de objeto (excepto las palabras reservadas). Todas las propiedades y todos los métodos definidos en `ClsComponentes` estarán ahora disponibles en la variable `componentes`. Al declarar el objeto con la palabra clave `New`, VBA crea el objeto y le asigna memoria. Sin embargo, el objeto no se creará hasta que te refieras a él con el código del procedimiento, asignando un valor a su propiedad o ejecutando alguno de sus métodos. Continuemos nuestro proyecto escribiendo el procedimiento VBA que lee los datos del archivo de texto en una colección de objetos `ClsComponentes`:

1. Desde el editor de VBA haz clic en el menú **Insertar – Módulo**, para agregar un módulo estándar al proyecto.
2. En la ventana **Propiedades** cambia el nombre al módulo por `InfoComponentes`.
3. En el explorador de proyectos haz clic en el nuevo módulo para activar la ventana **Código**.
4. Introduce el siguiente procedimiento:

```
Sub InfoComponente()
    ' Declara dos variables de objeto
    ' una para el propio objeto y la otra
    ' para la colección de objetos
    Dim componente As ClsComponentes
    Dim ColeccionComponentes As Collection
    ' Declara las variables para leer el archivo de datos
    Dim strTipo As String
    Dim strFabricante As String
    Dim strModelo As String
    Dim PrecioElemento As String
    ' Declara una variable para especificar el tipo de descuento
    ' en el cálculo del precio nuevo
    Dim intDescuento As Integer
    ' Declara variables usadas en la función MsgBox
    Dim strTitulo As String
    Dim strMensaje As String
    ' Declara variables para facilitar la introducción
    ' de datos en la hoja y en la ventana Inmediato
    Dim strRuta As String
    Dim strRegistro As String
    Dim wFila As Integer
    ' Declara las variables que se usarán en la colección
    Dim contador As Integer
    Dim aKey As String
    ' Declara las variables para acceder a un objeto
    ' de la colección via "clave"
    Dim componenteClave As String

    'Dim m As Object
    ' Si se produce un error, el código continua después
    On Error Resume Next
    ' initialize various variables
    strRuta = "C:\Archivos Manual VBA\Capítulo 8 -
Componentes.txt"
    contador = 0
    wFila = 1
    strMensaje = "Introduce 1 para descuento en porcentaje o "
    strMensaje = strMensaje + _
      " 2 para descuento en cantidad fija."
    strTitulo = "Tipo de descuento"
```

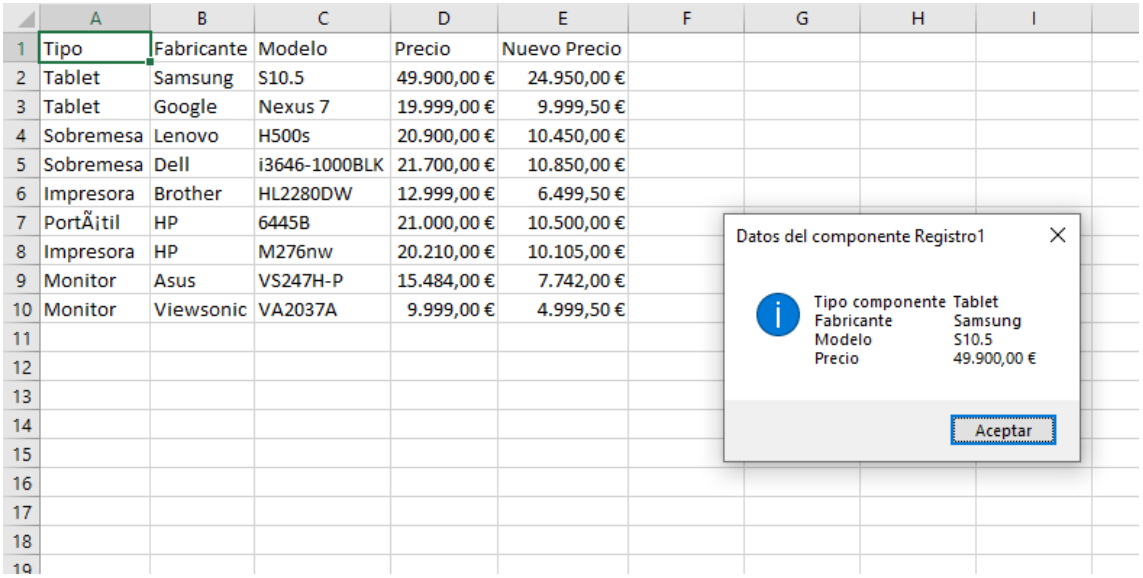
```
' Crea una nueva instancia de la colección
Set ColeccionComponentes = New Collection
' Abr el archivo de texto para lectura
Open strRuta For Input As #1
' Comprueba si el archivo está disponible
If Err.Number <> 0 Then
    MsgBox "¡El archivo no existe!", vbCritical, _
    "Error de archivo"
    Exit Sub
End If
' Pregunta al usuario el tipo de descuento a aplicar
intDescuento = CInt(InputBox(strMensaje, strTitulo, 1))
' Agrega una hoja vacía
ActiveWorkbook.Worksheets.Add
' ------------------------------------
' Bucle hasta que se encuentra el final del archivo
' ------------------------------------
Do While Not EOF(1)
    ' Lee los datos del archivo de texto en cuatro variables
    Input #1, strTipo, strFabricante, strModelo, _
    PrecioElemento
    If strTipo = "Tipo" Then
        ' ---------------------------------------------
        ' Introduce los encabezados de las columnas en la
        ' primera fila
        ' La quinta columna se reserva para el nuevo precio
        ' ---------------------------------------------
        With ActiveSheet
            .Cells(1, 1) = strTipo
            .Cells(1, 2) = strFabricante
            .Cells(1, 3) = strModelo
            .Cells(1, 4) = PrecioElemento
            .Cells(1, 5) = "Nuevo " & PrecioElemento
        End With
        ' Salta a la siguiente declaración
        GoTo Label_SaltaEncabezados
    End If
    '--------------------------------------
    ' Crea una instancia de la clase ClsComponentes
    '--------------------------------------
```

```
            Set componente = New ClsComponentes
            contador = contador + 1
            aKey = "Registro" & contador
            '-----------------------------------
            ' Establece las propiedades del objeto componente
            '-----------------------------------
            componente.Tipo = strTipo
            componente.Fabricante = strFabricante
            componente.Modelo = strModelo
            componente.Precio = PrecioElemento
            '-------------------------------------------
            ' Agrega un objeto componente a la colección
            ' y asigna una clave personalizada al objeto
            '-------------------------------------------
            ColeccionComponentes.Add componente, aKey
            Set componente = Nothing
Label_SaltaEncabezados:
            Resume Next
        Loop
        ' Cierra el archivo de texto
        Close #1
        ' Muestra un mensaje con información
        MsgBox "La colección Componentes contiene " & _
        ColeccionComponentes.Count & " elementos.", _
        vbInformation, "Total elementos"
        '----------------------------------------
        ' Recorre la colección y accede
        ' a cada instancia de la clase ClsComponentes
        ' mostrando los datos en la ventana Inmediato
        '----------------------------------------
        For Each componente In ColeccionComponentes
            Debug.Print componente.Tipo & vbTab & _
            componente.Fabricante & vbTab & _
            componente.Modelo & vbTab & _
            FormatNumber(componente.Precio, 2)
        Next componente
        '-------------------------------------------
        ' Recorre a colección y accede
        ' a cada instancia de la clase ClsComponentes
        ' mostrando los datos en la hoja activa
```

```
'-------------------------------------------
For Each componente In ColeccionComponentes
    ' Activa la siguiente fila en la hoja
    wFila = wFila + 1
    ' Escribe el registro en la hoja
    With ActiveSheet
        .Cells(wFila, 1) = componente.Tipo
        .Cells(wFila, 2) = componente.Fabricante
        .Cells(wFila, 3) = componente.Modelo
        .Cells(wFila, 4) = componente.Precio
        ' Calcula el descuento
        .Cells(wFila, 5) = _
        componente.NuevoPrecio(intDescuento, _
        componente.Precio, 100)
    End With
Next componente
Selection.CurrentRegion.Columns.AutoFit
'Obtiene el componente de la colección (por su clave)
componenteClave = InputBox("Introduce clave", "Recepción", _
  "Registro1")
Set componente = ColeccionComponentes.Item(componenteClave)
strRegistro = "Tipo componente" & vbTab _
  & componente.Tipo & vbCrLf
strRegistro = strRegistro & "Fabricante" & vbTab & _
  componente.Fabricante & vbCrLf
strRegistro = strRegistro & "Modelo" & vbTab & vbTab & _
  componente.Modelo & vbCrLf
strRegistro = strRegistro & "Precio" & vbTab & vbTab & _
  Format(componente.Precio, "Currency")
MsgBox strRegistro, vbInformation + vbOKOnly, _
"Datos del componente " & componenteClave
End Sub
```

5. Ejecuta el procedimiento. Responde a todas las preguntas aceptando los valores por defecto.
6. Tras ejecutar el procedimiento, deberías ver los datos de los componentes introducidos en una hoja y en la ventana **Inmediato**, según muestran la Imagen 3.1 y la Imagen 3.2.

Analicemos el procedimiento:



	A	B	C	D	E
1	Tipo	Fabricante	Modelo	Precio	Nuevo Precio
2	Tablet	Samsung	S10.5	49.900,00 €	24.950,00 €
3	Tablet	Google	Nexus 7	19.999,00 €	9.999,50 €
4	Sobremesa	Lenovo	H500s	20.900,00 €	10.450,00 €
5	Sobremesa	Dell	i3646-1000BLK	21.700,00 €	10.850,00 €
6	Impresora	Brother	HL2280DW	12.999,00 €	6.499,50 €
7	PortÃ¡til	HP	6445B	21.000,00 €	10.500,00 €
8	Impresora	HP	M276nw	20.210,00 €	10.105,00 €
9	Monitor	Asus	VS247H-P	15.484,00 €	7.742,00 €
10	Monitor	Viewsonic	VA2037A	9.999,00 €	4.999,50 €

Imagen 3.1 Los datos de los componentes en el archivo de texto se almacenan en una colección de objetos y se escriben en la hoja. La columna "Nuevo precio" no existe en el archivo original y se agregó para mostrar el uso de los métodos de clase.

Se comienza declarando e iniciando un gran número de variables que serán utilizadas en varias partes del código. Como se trata de acceder a un archivo externo, debes asegurarte de que se puede encontrar. En caso de que no sea así, se muestra un mensaje y el procedimiento termina. La propiedad **`Number`** del objeto **`Err`** devolverá un número diferente a cero si se ha encontrado algún problema al abrir el archivo. Para leer el archivo, debes abrirlo en modo de entrada (**`Input`**) utilizando la declaración:

```
Open strRuta For Input As #1
```

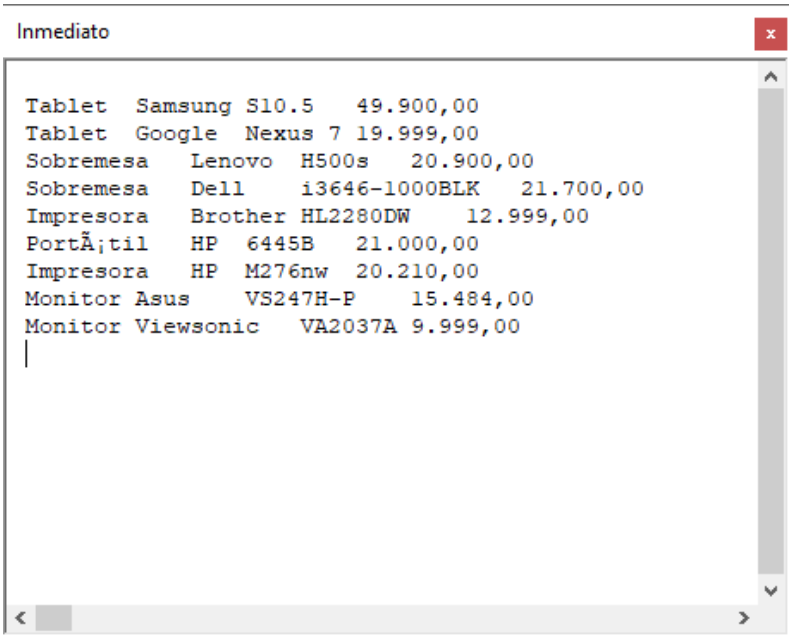


Imagen 3.2 Los datos también se imprimen en la ventana Inmediato

Una vez que el archivo está abierto, debe leerse secuencialmente de arriba a abajo. Esto puede hacerse usando el bucle **Do While** o **Do Until** que aprendiste en el Capítulo 6. Los archivos de texto contienen un carácter especial conocido como "marcador de fin de archivo", que el sistema operativo añade al archivo de forma automática. Al leer el archivo puedes usar la función **EOF** para detectar ese marcador y así saber si se llegó al final del archivo.

La declaración **Do While Not EOF(1)** expresa que quieres seguir ejecutando las declaraciones de dentro del bucle hasta que todos los datos del archivo hayan sido leídos. Esta declaración es equivalente a **Do Until EOF(1)**.

El número entre paréntesis es el número que corresponde al número de fichero del que se quieren leer los datos (el mismo número que se utilizó en la declaración **Open**). Cada vez que se accede al bucle, se utiliza la instrucción **Input #** para leer los datos del archivo en cuatro variables:

```
Input #1, strTipo, strFabricante, strModelo, PrecioElemento
```

Existen otras formas de leer archivos de texto en VBA, pero están más allá del objetivo de este libro.

Después de escribir los nombres de las columnas en la hoja, creamos nuestro objeto activo y establecemos sus cuatro propiedades (**Tipo**, **Fabricante**, **Modelo** y **Precio**) usando los valores almacenados en las variables:

```
componente.Tipo = strTipo
        componente.Fabricante = strFabricante
        componente.Modelo = strModelo
        componente.Precio = PrecioElemento
```

Cada una de las declaraciones de asignación es en realidad una llamada al procedimiento **Let** apropiado en el módulo de clase **ClsComponentes**. Por ejemplo, para establecer la propiedad **Tipo** del objeto activo, se ejecuta el siguiente procedimiento:

```
Property Let Tipo(ByVal aTipo As String)
    m_Tipo = aTipo
End Property
```

Para comprender mejor qué sucede cuando se ejecutan estas declaraciones, puedes ejecutar el procedimiento línea a línea (como se muestra en el siguiente capítulo).

En este punto, el objeto activo contiene los datos del primer registro, que es la segunda línea del archivo de texto. Antes de gestionar los datos del siguiente registro, utilizamos el método **Add** para añadir el objeto **componente** a la colección:

```
ColeccionComponentes.Add componente, aKey
```

Cada objeto de la colección se identifica con una clave que creamos concatenando la palabra "Registro" con un número, obteniendo "Registro1", "Registro2", "Registro3", etc. Después de añadir el objeto activo a la colección, liberamos la memoria estableciéndola en **Nothing** y continuamos con el siguiente registro, ejecutando las declaraciones dentro del bucle y

saltándose solo las utilizadas a modo de encabezados de columnas. Se crea un nuevo objeto, se establecen sus propiedades y se añade el objeto a la colección.

El mismo proceso se repite hasta que se alcanza el `EOF`(End Of File). Cuando terminamos de hacer el bucle, cerramos el archivo utilizando la instrucción `Close #1`. Ahora deberíamos tener nueve objetos activos en la colección `ColeccionComponentes`. El código restante del procedimiento recorre la colección de objetos e imprime los datos en la ventana **Inmediato** y en la hoja. Cuando recuperamos los objetos de la colección, VBA pasa a ejecutar los procedimientos `Property Get` que escribiste en el módulo `ClsComponentes`. Cuando escribimos el "Nuevo precio" en la hoja, llamamos al método `NuevoPrecio`. Este método utiliza la variable `intDescuento`, cuyo valor se obtuvo del usuario en el procedimiento. Si se aceptó el valor por defecto en el `InputBox`, entonces el precio se reduce en el porcentaje especificado. El último parámetro del método `NuevoPrecio`, que contiene la cantidad, está codificado de forma fija. En función a la cantidad introducida (1 o 2), las declaraciones `If` incluidas en el método `NuevoPrecio` se ejecutarán o se omitirán. Cuando se introducen números, a veces es necesario dar un formato adecuado a los datos. El procedimiento `InfoComponente` utiliza la función `FormatNumber` para dar formato a los datos de los precios en la ventana **Inmediato**.

```
… FormatNumber(componente.Precio, 2)
```

El segundo argumento de la función `FormatNumber` especifica cuántos decimales se deben mostrar. Para dar formato de moneda, cambia la función anterior a:

```
… FormatCurrency(componente.Precio, 2)
```

4 Resumen

En este capítulo has aprendido a crear y utilizar tus propios objetos y colecciones dentro de procedimientos. Has utilizado un módulo de clase para crear un objeto personalizado y has aprendido a definir las propiedades de un objeto utilizando los procedimientos `Property Get` y `Property Let`. También has aprendido a escribir un método para tu objeto personalizado.

En el próximo capítulo aprenderás a solucionar los problemas que pueden darse en los procedimientos VBA.

Capítulo 9
Resolución de errores con VBA

No es complicado que se produzca un error al trabajar con procedimiento en VBA,no importa lo cuidadoso que seas. Es raro que un procedimiento funcione correctamente la primera vez. Existen tres tipos de error en VBA: de sintaxis, lógicos y de ejecución. Este capítulo te presenta las herramientas disponibles en el editor de VBA para analizar el código de los procedimientos y para detectar los errores.

1 Probando los procedimientos VBA

Como la mayoría de los procedimientos escritos en capítulos anteriores, eran bastante cortos, encontrar errores no fue muy complicado. Sin embargo, localizar el origen de errores en procedimientos largos y complejos es más tedioso y lleva más tiempo. Afortunadamente, el editor de VBA cuenta con un conjunto de herramientas útiles que pueden hacer que el proceso de rastreo de problemas en VBA sea más rápido, fácil y menos frustrante. Se llaman “bugs” a los errores en los programas informáticos. La “depuración” es el proceso de localizar y arreglar esos bugs recorriendo el código del procedimiento o comprobando los valores de las variables.

Cuando pruebes un procedimiento VBA, usa los siguientes trucos:

- Para analizar un procedimiento, ejecútalo línea por línea presionando **F8** o haciendo clic en el menú **Depuración – Paso a paso por instrucciones**.
- Para localizar un error en un lugar específico del procedimiento, utiliza **puntos de interrupción**.
- Para monitorizar o vigilar el valor de una variable o expresión de un procedimiento, utiliza “**inspecciones”**.
- Para saltar rápidamente a las secciones de código que te interesan, configura **marcadores**.

Todos estos trucos o pautas se muestran de forma práctica en este capítulo.

2 Parar un procedimiento

Mientras haces pruebas con un procedimiento, puede que quieras detener su ejecución. Esto se puede hacer simplemente presionando la tecla **Esc**, que hace que VBA detenga el programa y muestre el mensaje que se muestra en la Imagen 2.1. VBA también ofrece otros métodos para detener un procedimiento. Cuando detienes un procedimiento de forma manual, VBA entra en lo que se llama "modo de interrupción". Para entrar en este modo, puedes realizar una de las siguientes acciones:

- Presionar **Ctrl + Inter**.
- Establecer uno o más puntos de ruptura.
- Insertar la declaración `Stop` en el código del procedimiento.
- Agregar una inspección de vigilancia.

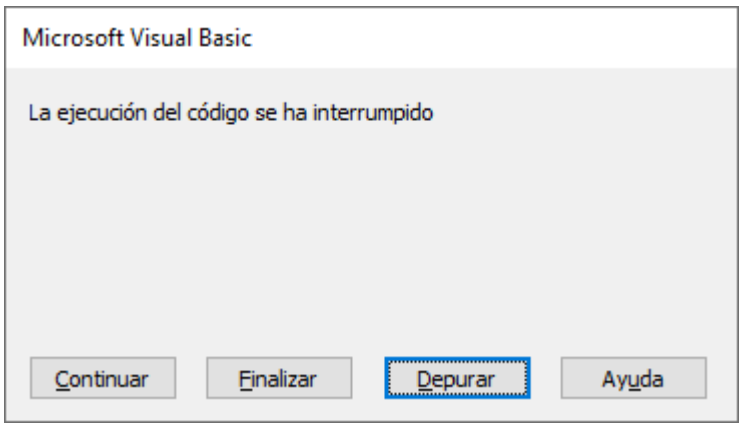


Imagen 2.1 Este mensaje aparece cuando presionas la tecla Esc o Ctrl + Inter mientras se está ejecutando el procedimiento.

Se produce una ruptura cuando se suspende la ejecución de un procedimiento. VBA recuerda los valores de todas las variables y la línea desde la que debe reanudar la ejecución del procedimiento cuando el usuario decida continuar haciendo clic en **Ejecutar Sub/UserForm** en la barra de herramientas (o en la opción del menú **Ejecutar**), o haciendo clic en el botón **Continuar** del cuadro de diálogo. Este cuadro de error que se muestra en la Imagen 2.1 informa de que el procedimiento se ha detenido. Los botones de este cuadro de diálogo se describen en la siguiente tabla:

Botón	Descripción
Continuar	Se reanuda la ejecución del código. El botón estará deshabilitado si se encontró un error.
Finalizar	Finaliza la ejecución del procedimiento. Se suele utilizar en caso de bucles infinitos.
Depurar	Entra en modo Interrupción. Aparecerá la ventana de Código y VBA resaltará la línea en la que se suspendió la ejecución del

Botón	Descripción
	procedimiento. Puedes examinar, depurar, o reiniciar el código.
Ayuda	Muestra la ayuda online que explica la causa por la que se ha producido el error.

Es posible evitar que los usuarios de la aplicación detengan un procedimiento incluyendo la siguiente declaración en el código:

```
Application.EnableCancelKey = xlDisabled
```

Si el usuario intenta parar el procedimiento mientras se está ejecutando, no sucederá nada. La propiedad `EnableCancelKey` del objeto `Application` se desactiva.

3 Los puntos de ruptura

Si al escribir un procedimiento sabes dónde se podría producir un error, puedes suspender la ejecución del código en una línea determinada. Presionando **F9** es posible establecer un punto de interrupción en la línea en la que se encuentre el cursor. Cuando VBA llegue a esa línea durante la ejecución del procedimiento, mostrará la ventana **Código** inmediatamente. En ese momento, puedes comenzar a ejecutar el procedimiento línea a línea presionando **F8**. Para ver cómo funciona esto, veamos el siguiente ejemplo. Supongamos que, durante la ejecución del siguiente procedimiento, esta línea de código producealgún problema:

```
ActiveCell.FormulaR1C1 "=VLOOKUP(RC[1],Codes.xlsx!R1C1:R6C2,2)"
```

1. Copia el archivo “Capítulo 9 – Depuración errores.xlsm” en la carpeta **Archivos Manual VBA** en C:\.
2. Copia también el archivo “Códigos.xlsx” en la misma carpeta.
3. Abre los dos archivos y mantenlos abiertos.
4. Observa los datos que contienen la Imagen 3.1 y la Imagen 3.2.

	A	B	C	D	E	F
1	Profesor	Puesto	Cantidad	Código	Código 1	
2	Romeo Acera	A	6500	227.163-23-220	66	
3	Isabel Neyra	A	6500	227.163-14-100	62	
4	Leónidas Almansa	A	6500	211.163-23-330	73	
5	Manuel Camero	B	6300	211.163-23-330	73	
6	Zuriñe Cla	B	6300	211.163-23-330	73	
7	Enrique Porsimecopian	B	6300	211.163-23-220	65	
8	Graciela Rico	B	6300	227.163-11-100	67	
9						
10						
11						

Imagen 3.1 Los datos introducidos en la columna D de esta hoja serán sustituidos por el procedimiento CambiarCodigo con los datos de la siguiente imagen.

	A	B	C	D
1	62	227.163-14-100		
2	65	211.163-23-220		
3	66	227.163-23-220		
4	67	227.163-11-100		
5	73	211.163-23-330		
6	78	211.163-28-330		
7				
8				
9				
10				
11				
12				

Imagen 3.2 El procedimiento CambiarCodigo utiliza los datos de esta tabla.

5. Cierra el archivo Códigos.xlsx. Deja el otro archivo abierto.
6. Con el archivo "Capítulo 9 – Depuración errores.xlsm" activo, cambia al editor de VBA.
7. En el **Explorador de proyectos**, abre la carpeta Módulos del proyecto `Depuracion`. A continuación, haz doble clic en el módulo `Interrupciones`. La ventana **Código** muestra el procedimiento `CambiarCodigo`:

```
Sub CambiarCodigo()
    Workbooks.Open Filename:= _
      "C:\Archivos Manual VBA\Códigos.xlsx"
    Windows("Capítulo 9 - Depuración errores.xlsm").Activate
    Columns("D:D").Insert Shift:=xlToRight
    Range("D1").Formula = "Código"
    Columns("D:D").SpecialCells(xlBlanks).Select
    ActiveCell.FormulaR1C1 = _
      "=VLookup(RC[1],Códigos.xlsx!R1C1:R6C2,2)"
    Selection.FillDown
        With Columns("D:D")
            .EntireColumn.AutoFit
            .Select
        End With
    Selection.Copy
    Selection.PasteSpecial Paste:=xlValues
    Rows("1:1").Select
        With Selection
            .HorizontalAlignment = xlCenter
            .VerticalAlignment = xlBottom
            .Orientation = xlHorizontal
        End With
    Workbooks("Códigos.xlsx").Close
End Sub
```

8. Busca la siguiente línea en el código y haz clic en ella para seleccionarla:

```
ActiveCell.FormulaR1C1 = _
"=VLookup(RC[1],Códigos.xlsx!R1C1:R6C2,2)"
```

9. Crea un punto de interrupción presionando **F9** (o desde el menú, seleccionando **Depuración – Alternar punto de interrupción**).
   Al establecer el punto de interrupción, VBA muestra un círculo rojo en el margen. Además, la línea que tiene el punto se muestra con texto blanco sobre fondo rojo, como en la Imagen 3.3. El color del punto de interrupción o ruptura se puede cambiar en la pestaña **Formato** del editor del cuadro **Opciones**, situado en el menú **Herramientas**.
10. Presiona **F5** para ejecutar el procedimiento.
    Cuando lo hagas, VBA ejecutará todas las instrucciones hasta que se encuentre con el punto de interrupción. La Imagen 3.4 muestra la línea donde se ha interrumpido el procedimiento con fondo amarillo y una flecha en el margen. Esta marca indica la declaración que está a punto de ser ejecutada.
    Si la declaración actual contiene un punto de interrupción, en el margen se mostrarán ambos marcadores (el círculo rojo y la flecha amarilla).

```
Sub CambiarCodigo()
    Workbooks.Open Filename:="C:\Archivos Manual VBA\Códigos.xlsx"
    Windows("Capítulo 9 - Depuración errores.xlsm").Activate
    Columns("D:D").Insert Shift:=xlToRight
    Range("D1").Formula = "Código"
    Columns("D:D").SpecialCells(xlBlanks).Select
    ActiveCell.FormulaR1C1 = "=VLookup(RC[1],Códigos.xlsx!R1C1:R6C2,2)"
    Selection.FillDown
        With Columns("D:D")
            .EntireColumn.AutoFit
            .Select
        End With
    Selection.Copy
    Selection.PasteSpecial Paste:=xlValues
    Rows("1:1").Select
        With Selection
            .HorizontalAlignment = xlCenter
            .VerticalAlignment = xlBottom
```

Imagen 3.3 La línea de código donde se establece el punto de ruptura se muestra en el color que se especifique en el cuadro de diálogo Opciones.

```
Option Explicit

Sub CambiarCodigo()
    Workbooks.Open Filename:="C:\Archivos Manual VBA\Códigos.xlsx"
    Windows("Capítulo 9 - Depuración errores.xlsm").Activate
    Columns("D:D").Insert Shift:=xlToRight
    Range("D1").Formula = "Código"
    Columns("D:D").SpecialCells(xlBlanks).Select
    ActiveCell.FormulaR1C1 = "=VLookup(RC[1],Códigos.xlsx!R1C1:R6C2,2)"
    Selection.FillDown
        With Columns("D:D")
            .EntireColumn.AutoFit
            .Select
        End With
    Selection.Copy
    Selection.PasteSpecial Paste:=xlValues
    Rows("1:1").Select
        With Selection
            .HorizontalAlignment = xlCenter
```

Imagen 3.4 Cuando VBA se encuentra con un punto de interrupción, muestra la ventana Código e indica la siguiente declaración que se ejecutará.

En el modo interrupción, puedes cambiar el código, añadir nuevas declaraciones, ejecutar el procedimiento **Paso a paso por instrucciones**, saltarte líneas, establecer la siguiente declaración, utilizar la ventana **Inmediato** y otras acciones. Cuando VBA está en modo interrupción, todos los comandos del menú **Depuración** están activos. Si cambias cierto código mientras el procedimiento está en el modo interrupción, VBA te pedirá que reinicies el proyecto mostrando el mensaje de error: "Esta acción restablecerá su proyecto ¿desea continuar?". Puedes hacer clic en **Aceptar** para detener la ejecución del programa y proceder a editar el código, o hacer clic en **Cancelar** para eliminar los nuevos cambios y continuar ejecutando el código desde el punto en que se suspendió.

11. Presiona **F5** (o haz clic en el menú **Ejecutar – Ejecutar Sub/UserForm**) para continuar ejecutando el procedimiento. VBA abandona el modo interrupción y continúa ejecutando las instrucciones hasta que llega al final. Cuando el procedimiento termina de ejecutarse, VBA no elimina automáticamente el punto de interrupción. Observa que la línea con la función `VLookup` sigue resaltada en rojo.
    En este ejemplo sólo se ha creado un punto de interrupción, pero VBA permite establecer la cantidad que sea necesaria en un procedimiento. De esta forma, puedes suspender y continuar la ejecución del procedimiento tantas veces como desees. Puedes analizar el código del procedimiento y comprobar los valores de las variables mientras la ejecución está suspendida. También puedes hacer pruebas de código escribiendo declaraciones en la ventana **Inmediato**.
12. Elimina el punto de interrupción haciendo clic en el menú **Depuración – Borrar todos los puntos de interrupción**, presionando **Ctrl + Mayús + F9** o haciendo clic en el círculo rojo al margen.
    Se eliminarán todos los puntos de interrupción. Si has creado varios de ellos en un procedimiento determinado y deseas eliminar uno o varios, haz clic en la línea que contiene el código y presiona **F9**. Debes eliminar los puntos de interrupción cuando ya no sean necesarios, aunque se borran automáticamente cuando cierras el libro.
13. Cambia a la ventana de Excel y observa que se agregó una nueva columna con los códigos buscados (ver Imagen 3.5).

	A	B	C	D	E	F
1	Profesor	Puesto	Cantidad	Código	Código1	
2	Romeo Acera	A	6500	227.163-23-220	66	
3	Isabel Neyra	A	6500	227.163-14-100	62	
4	Leónidas Almansa	A	6500	211.163-23-330	73	
5	Manuel Camero	B	6300	211.163-23-330	73	
6	Zuriñe Cla	B	6300	211.163-23-330	73	
7	Enrique Porsimecopian	B	6300	211.163-23-220	65	
8	Graciela Rico	B	6300	227.163-11-100	67	
9						
10						
11						
12						
13						
14						

Imagen 3.5 Este es el resultado de ejecutar el procedimiento CambiarCodigo.

3.1 Cuándo utilizar un punto de interrupción

Es buena idea crear un punto de interrupción si sospechas que el procedimiento no ejecuta correctamente un cierto bloque de código.

En el modo interrupción, puedes averiguar rápidamente el contenido de la variable observando el cursor en la ventana **Código** al situarlo encima de ella. Por ejemplo, en el procedimiento `VarValor` que se muestra en la Imagen 3.6, el punto de interrupción se ha establecido en la instrucción `Workbooks.Add`. Cuando VBA se encuentra con esta instrucción, aparece la ventana **Código** (en modo interrupción). Dado que VBA ya ha ejecutado la instrucción que almacena el nombre de `ActiveWorkbook` en la variable `strNombre`, puedes averiguar rápidamente el valor de esta variable colocando el puntero del ratón sobre su nombre. El nombre de la variable y su valor actual aparecen en un cuadro de información.

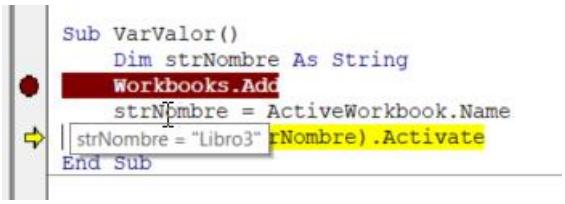


Imagen 3.6 En el modo interrupción es posible averiguar el valor de una variable situando el puntero del ratón encima de ella.

> ### Nota
>
> Para mostrar a un mismo tiempo los valores de varias variables utilizadas en un procedimiento, se debe utilizar la ventana **Locales**, de la que se hablará más adelante en este capítulo.

4 Uso de la ventana Inmediato en modo Interrupción

Una vez que se suspende la ejecución de un procedimiento y aparece la ventana **Código**, se puede activar la ventana **Inmediato** y escribir instrucciones para saber, por ejemplo, qué celda está activa en ese momento o el nombre de la hoja activa. También puedes utilizar la ventana **Inmediato** para cambiar el contenido de las variables con el fin de corregir los valores que puedan estar causando errores.

La Imagen 5.1 muestra el procedimiento `CambiarCodigo` suspendido y la ventana **Inmediato** con algunas preguntas que se le hicieron a VBA mientras estaba en modo interrupción.

5 Uso de las instrucciones Stop y Assert

En ocasiones no podrás probar tu procedimiento inmediatamente. Si configuras puntos de interrupción y luego cierras el archivo, Excel los eliminará y la próxima vez que desees probar el procedimiento, deberás establecerlos de nuevo. Si deseas probar el procedimiento después de haber cerrado el archivo, debes insertar la instrucción `Stop` en el lugar donde desees detener el procedimiento. La Imagen 5.2 muestra una instrucción `Stop` antes del bucle `For`

`Each` … `Next`. VBA suspenderá la ejecución del procedimiento cuando se encuentre con dicha instrucción. La pantalla mostrará la ventana **Código** en modo interrupción.

Aunque la instrucción `Stop` tiene exactamente el mismo efecto que establecer un punto de interrupción, tiene una desventaja: todas las declaraciones `Stop` permanecen en el procedimiento hasta que las eliminas manualmente. Cuando ya no necesites detener el procedimiento, debes encontrar y eliminar todas las declaraciones `Stop`.

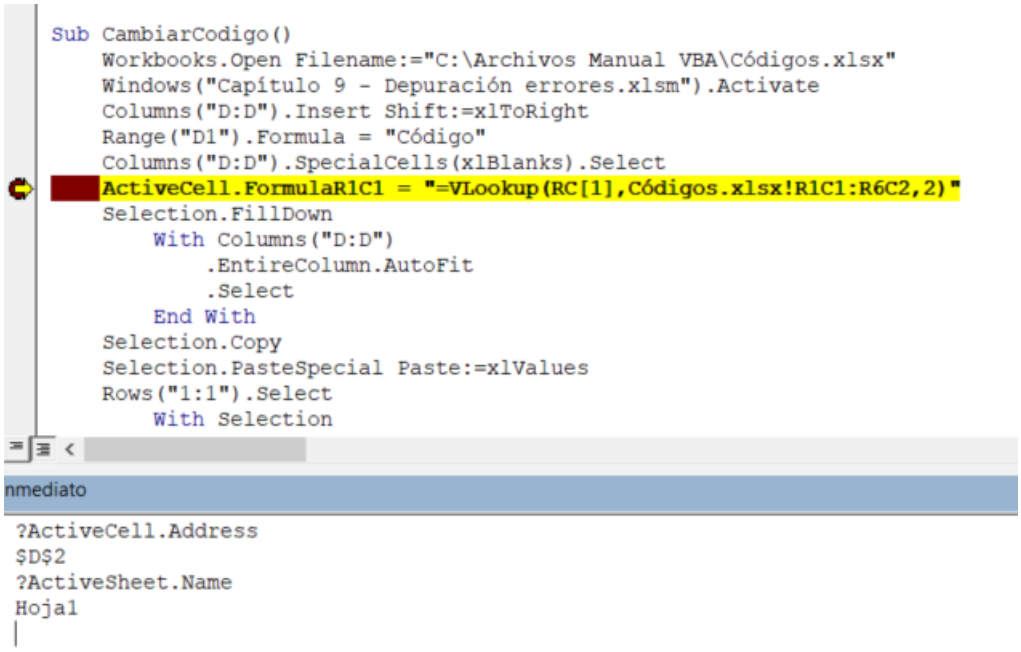


Imagen 5.1 Cuando se suspende la ejecución del código, puedes encontrar los valores de las variables y ejecutar algunas instrucciones introduciéndolas en la ventana Inmediato.

```
Sub DetenerEjemplo()
    Dim CeldaActual As Range
    Dim num As Integer

    ActiveWorkbook.Sheets(1).Select
    ActiveSheet.UsedRange.Select
    num = Selection.Columns.Count
    Selection.Resize(1, num).Select
    Stop
    For Each CeldaActual In Selection
        Debug.Print CeldaActual.Text
    Next
End Sub
```

Imagen 5.2 Puedes insertar una instrucción Stop en cualquier lugar del código del procedimiento. El procedimiento se detendrá cuando llegue a dicha instrucción y la ventana Código aparecerá con la línea resaltada.

Aunque la instrucción `Stop` tiene exactamente el mismo efecto que establecer un punto de interrupción, tiene una desventaja: todas las declaraciones `Stop` permanecen en el

procedimiento hasta que las eliminas manualmente. Cuando ya no necesites detener el procedimiento, debes encontrar y eliminar todas las declaraciones `Stop`.

Una técnica de depuración muy potente y fácil de aplicar es el uso de la declaración `Debug.Assert`. Si incluyes esta declaración en tu código, puedes comprobar que una condición es verdadera. Ofrece una retroalimentación inmediata cuando ocurre un error. Son excelentes para detectar errores lógicos. El hecho de que un procedimiento se haya ejecutado sin generar un error no significa que no haya errores en ese procedimiento. La expresión `Debug.Assert` evalúa cualquier expresión que pueda ser verdadera o falsa y activa el modo de interrupción cuando devuelva el valor `False`. Esta es su sintaxis:

```
Debug.Assert condición
```

`Condición` es el código o expresión de VBA que devuelve verdadero o falso. Si la condición se evalúa en falso o en 0 (cero), VBA entrará en modo interrupción. Por ejemplo, cuando se ejecuta la siguiente estructura de bucle, el código dejará de ejecutarse cuando la variable `i` sea igual a 50:

```
Sub ControlErroresAssert()
    Dim i As Integer

    For i = 1 To 100
        Debug.Assert i <> 50
    Next
End Sub
```

Ten en cuenta que `Debug.Asset` no hace nada hasta que la declaración sea falsa. La ejecución simplemente se detiene en esa línea de código y el editor de VBA se abre mostrando resaltada la línea que contiene la afirmación falsa para que puedas comenzar a depurar el código. Puede que tengas que escribir un gestor de errores para manejar el error que se produzca. Los procedimientos de gestión de errores se detallan más adelante en este capítulo.

Aunque puedes detener la ejecución del código usando la instrucción `Stop`, la instrucción `Debug.Asset` se diferencia de ella en su aspecto condicional: detendrá el código sólo bajo condiciones específicas. Los puntos de interrupción condicionales también pueden ser establecidos desde la ventana **Inspecciones** (ver siguiente sección).

Tras depurar y probar el código, comenta o elimina las declaraciones `Debug.Assert` del código final. La forma más fácil de hacerlo es utilizar el comando **Reemplazar** para comentar las declaraciones. Puedes acceder desde el menú o presionando **Ctrl + H**.

En el cuadro de diálogo que aparece escribe `Debug.Assert` en el cuadro de texto **Buscar** y `'Debug.Assert` en el cuadro de texto **Reemplazar**. A continuación, haz clic en el botón **Reemplazar todos**. Esto hará que se comenten todas las líneas que contienen la instrucción escrita. Observa el apóstrofe delante de la instrucción.

En caso de quieras eliminar las instrucciones `Debug.Assert`, solo tienes que dejar vacío el cuadro de texto **Reemplazar**.

6 Cómo usar la ventana Inspecciones

Muchos errores que surgen son producidos por variables que asumen valores inesperados.

Si un procedimiento utiliza una variable cuyo valor cambia en varios lugares, puede que quieras detener el procedimiento y comprobar el valor de esa variable en un momento dado. VBA ofrece una ventana especial de vigilancia o inspección que te permite seguir variables o expresiones mientras el procedimiento está en marcha. Para añadir una expresión de inspección a un procedimiento, haz lo siguiente:

- En la ventana **Código**, selecciona la variable cuyo valor deseas monitorear.
- Haz clic en el menú **Depuración – Agregar** inspección. Se mostrará el cuadro de diálogo **Agregar inspección**, como se muestra en la Imagen 6.1. Como se puede apreciar, el cuadro contiene tres secciones que se describen en la siguiente tabla:

Sección	Descripción
Expresión	Muestra el nombre de la variable resaltada en el procedimiento. Si abriste el cuadro **Agregar inspección** sin seleccionar una variable, escribe el nombre de la variable que deseas supervisar en el cuadro de texto.
Contexto	En esta sección se debe indicar el nombre del procedimiento que contiene la variable y el nombre del módulo donde se encuentra el procedimiento.
Tipo de inspección	Especifica cómo se debe monitorear la variable. Si seleccionas la opción **Expresión de inspección**, podrás leer el valor de la variable en la ventana Inspecciones mientras el procedimiento está en modo interrupción. Si seleccionas **Interrupción cuando el valor sea verdadero**, VBA detendrá automáticamente el procedimiento cuando la variable se evalúe como verdadera (no cero). La última opción, **Interrupción cuando el valor cambia**, detiene el procedimiento cada vez que cambia el valor de la variable o expresión.

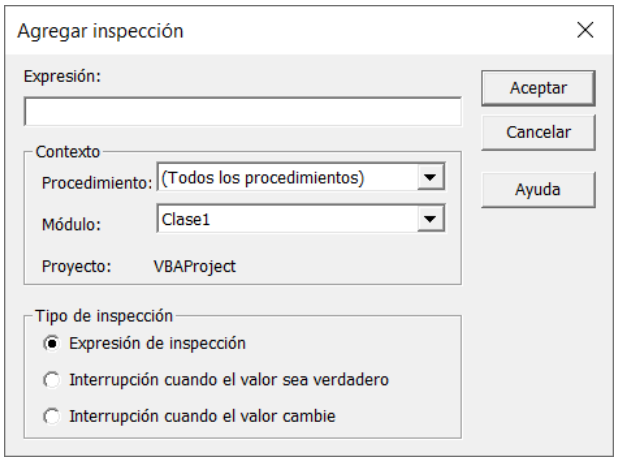


Imagen 6.1 La ventana Agregar inspección.

Puedes añadir una expresión de inspección antes de ejecutar un procedimiento o después de que la ejecución se haya suspendido.

La diferencia entre un punto de interrupción y una expresión de interrupción es que el punto siempre detiene un procedimiento en una ubicación especificada mientras que la expresión detiene el procedimiento solo cuando se cumple la condición impuesta. Las inspecciones son extremadamente útiles cuando no estás seguro de dónde se modifica el valor de la variable.

En lugar de repasar las líneas de código para encontrar la ubicación donde una variable toma el valor especificado, puedes simplemente poner una expresión de inspección en la variable y ejecutar su procedimiento de forma normal. Veamos cómo funciona esto:

1. El módulo Interrupciones contiene un procedimiento llamado **enFecha**.

```
Sub enFecha()
    Dim fechaActual As Date
    Dim nuevaFecha As Date
    Dim x As Integer

    fechaActual = Date
    For x = 1 To 365
        nuevaFecha = Date + x
    Next
End Sub
```

   El procedimiento utiliza el bucle **For … Next** para calcular la fecha, que es **x** días en el futuro. Si ejecutas el procedimiento, no obtendrás ningún resultado a menos que introduzcas la siguiente instrucción en el código:

```
MsgBox "En " & x & " días, será " & nuevaFecha
```

   ¿Qué ocurre si quieres detener el programa cuando el valor de la variable **x** llegue a 211? En otras palabra, ¿qué fecha será dentro de 211 días? Para obtener la respuesta podrías insertar la siguiente declaración en el procedimiento:

```
If x = 211 Then MsgBox "En " & x & " días, será " & nuevaFecha
```

   No siempre es posible introducir nuevas declaraciones en el procedimiento solo para obtener el valor de cierta variable cuando se valida una condición. En lugar de insertar la función **MsgBox** u otras declaraciones de depuración en el procedimiento, que tendrás que borrar más tarde, puedes usar la ventana **Inspecciones** y evitar códigos innecesarios. Si agregas inspecciones al procedimiento, VBA detendrá el bucle **For … Next** cuando se cumpla la condición especificada y podrás comprobar los valores de las variables deseadas.

2. Haz clic en el menú **Depuración – Agregar inspección**.
3. En el cuadro de texto **Expresión**, introduce la siguiente expresión: x=211. En la sección **Contexto**, selecciona el procedimiento **enFecha** del cuadro combinado Procedimiento e

Interrupciones del cuadro combinado “Módulo”. A continuación, haz clic en la opción “Interrumpir cuando el valor sea verdadero”.

4. Haz clic en **Aceptar** para cerrar el cuadro de diálogo **Agregar inspección**. Ya has agregado tu primera inspección. VBA abrirá la ventana Inspecciones y mostrará la expresión x=211.

Ahora vamos a agregar otra expresión a la ventana **Inspecciones** que nos permitirá monitorizar la fecha actual.

5. En la ventana **Código**, sitúa el cursor en cualquier lugar dentro del nombre de la variable **`fechaActual`**.
6. Haz clic en el menú **Depuración – Agregar inspección** y haz clic en **Aceptar** para configurar la inspección predeterminada. Observa que la variable **`fechaActual`** aparece ahora en la columna **Expresión** dentro de la ventana **Inspecciones**.

También vamos a seguirle la pista a la variable **`nuevaFecha`**.

7. En la ventana **Código** sitúa el cursor en cualquier lugar de la variable **`nuevaFecha`**.
8. Haz clic en **Depuración – Agregar inspección** y haz clic en **Aceptar** para configurar el tipo de inspección predeterminado. Observa también cómo aparece el nombre de la variable en la columna **Expresión** en la ventana **Inspecciones**.

Tras realizar los pasos anteriores, el procedimiento **`enFecha`** contiene las siguientes inspecciones:

- X=211 – Interrupción cuando el valor sea verdadero.
- fechaActual – Expresión de inspección.
- nuevaFecha – Expresión de inspección.

9. Sitúa el cursor en cualquier lugar dentro del código de **`enFecha`** y presiona **F5**.
   La Imagen 6.2 muestra la ventana **Inspecciones** cuando VBA detiene el procedimiento porque **`x`** es igual a 211.

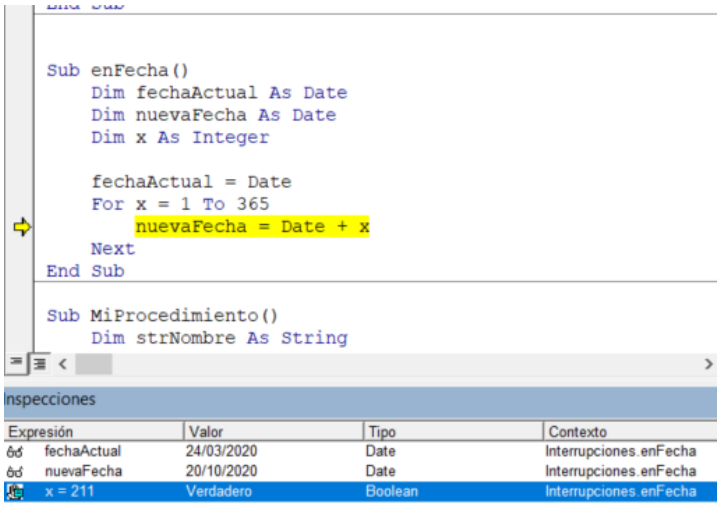


Imagen 6.2 La ventana Inspecciones muestra el estado en un punto determinado de la ejecución del procedimiento.

Observa que el valor de `x` en la ventana **Inspecciones** es el mismo que especificaste en el cuadro **Agregar inspección**. Además, la inspección muestra el valor de las variables `fechaActual` y `nuevaFecha`. En este momento el procedimiento se encuentra en modo interrupción. Puedes presionar **F5** para continuar o preguntar cosas como ¿qué fecha será dentro de 277 días? El siguiente paso muestra cómo hacerlo:

10. Selecciona **Depuración – Modificar inspección** e introduce la expresión x=277.
11. Haz clic en **Aceptar** para cerrar el cuadro de diálogo.
    Observa que en la ventana **Inspecciones** ahora se muestra un nuevo valor para la expresión x: es falso.
12. Presiona **F5** para continuar ejecutando el procedimiento, que se detiene de nuevo cuando el valor de x es igual a 277. El valor `fechaActual` es el mismo; sin embargo, la `nuevaFecha` ahora contiene otro valor, que es 277 días a partir de hoy. Puedes cambiar de nuevo el valor de la expresión o terminar de ejecutar el procedimiento.
13. Presiona **F5** para terminar de ejecutar el procedimiento.
    Cuando el procedimiento se está ejecutando y una expresión contiene un valor, la ventana **Inspecciones** muestra el valor de esa expresión. Si abres la ventana **Inspecciones** después de que el procedimiento haya finalizado, se muestra <Fuera de contexto> en lugar de los valores de las variables. Dicho de otro modo, cuando la expresión inspeccionada no se está ejecutando, no tiene valor.

6.1 Cómo eliminar expresiones de inspección

Para eliminar una inspección, haz clic con el botón derecho del ratón en la expresión en la ventana **Inspecciones** y haz clic en **Eliminar inspección** del menú contextual. Ahora puedes eliminar todas las expresiones definidas para este ejemplo.

7 Inspecciones rápidas

En el modo interrupción puedes comprobar el valor de una expresión para la que no hayas definido una inspección como tal.

Imagen 7.1 El cuadro Inspección rápida muestra el valor de la expresión seleccionada en un procedimiento VBA.

Puedes acceder al cuadro de diálogo **Inspección rápida** de las siguientes formas:

- Mientras el procedimiento se encuentra en modo interrupción, sitúa el cursor en cualquier lugar dentro del nombre de una variable o expresión que desees inspeccionar.
- Haz clic en el menú **Depuración – Inspección rápida**.
- Presiona **Mayús + F9**.

El botón **Agregar** en el cuadro de diálogo **Inspección rápida** te permite agregar la expresión a la ventana **Inspecciones**. Averigüemos cómo trabajar con este cuadro de diálogo:

1. Asegúrate de que el procedimiento `enFecha` que introdujiste en el ejercicio anterior no contenga ninguna inspección.
2. Sitúa el cursor en el nombre de la variable `x`.
3. Haz clic en el menú **Depuración – Agregar inspección**.
4. Introduce la expresión x=50.
5. Elige la opción **Interrumpir cuando el valor sea verdadero** y haz clic en **Aceptar**.
6. Ejecuta el procedimiento `enFecha`.
   VBA suspenderá la ejecución del procedimiento cuando `x` sea igual a 50. Observa que la ventana **Inspecciones** no contiene las variables `fechaActual` ni `nuevaFecha`. Para comprobar los valores de estas variables, puedes situar el cursor sobre el nombre de la variable apropiada en la ventana **Código**, o puedes abrir el cuadro de diálogo **Inspección rápida**.
7. En la ventana **Código**, posiciona el puntero del ratón dentro de `nuevaFecha` y presiona **Mayús + F9**. Se mostrará el cuadro de diálogo **Inspección rápida** con la expresión y su valor actual.
8. Haz clic en **Cancelar** para volver a la ventana **Código**.
9. En la ventana código posiciona el cursor dentro de la variable `fechaActual` y presiona **Mayús + F9**. Se vuelve a mostrar el cuadro Inspección rápida con el nombre de la variable seleccionada.
10. De nuevo, haz clic en **Cancelar** para volver a la ventana **Código**.
11. Presiona **F5** para continuar ejecutando el procedimiento.
12. En la ventana **Inspecciones**, resalta la fila que contiene la expresión x=50 y presiona **Eliminar inspección**, del menú contextual.

8 La ventana Locales

Si durante la ejecución de un procedimiento quieres conocer los valores de todas las variables declaradas, haz clic en el menú **Ver – Ventana Locales** antes de ejecutar el procedimiento. La Imagen 8.1 La ventana Locales muestra los valores actuales de todas las variables declaradas en el procedimiento actual de VBA. muestra una lista de variables y sus valores correspondientes en la ventana **Locales** que se muestra mientras VBA está en modo interrupción.

La ventana **Locales** contiene tres columnas. La primera de ellas, Expresión, muestra los nombres de las variables que se declaran en el procedimiento actual. La primera fila muestra el nombre del módulo precedido por el signo más. Al hacer clic en el signo más, se puede comprobar si se ha declarado alguna variable a nivel de módulo. Para los módulos de clase se define la variable de sistema `Me`. Para los módulos estándar, la primera variable es el nombre del módulo actual. Las variables globales y las variables de otros proyectos no son accesibles desde la ventana **Locales**.

La segunda columna muestra los valores actuales de las variables. En esta columna puedes cambiar el valor de una variable haciendo clic en ella y escribiendo el nuevo valor. Tras

cambiar el valor, pulsa **Intro** para registrar el cambio. También puedes presionar **Tab, Mayús + Tab**, las flechas hacia arriba y hacia abajo o hacer clic en cualquier lugar dentro de la ventana **Locales**. La tercera columna muestra el tipo de cada variable declarada.

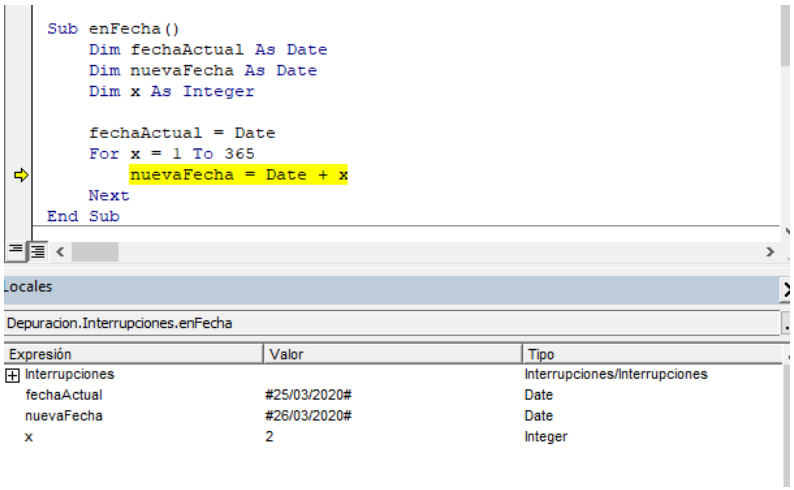


Imagen 8.1 La ventana Locales muestra los valores actuales de todas las variables declaradas en el procedimiento actual de VBA.

Para observar los valores de las variables en la ventana **Locales**, hagamos el siguiente ejercicio:

1. Haz que se muestre la ventana Locales desde el menú **Ver – Ventana Locales**.
2. Haz clic en cualquier lugar del procedimiento `enFecha` y presiona **F8** para iniciarlo. Al presionar **F8** el procedimiento se pone en modo interrupción. La ventana **Locales** muestra el nombre del módulo actual y las variables locales, además de sus valores iniciales.
3. Ve presionando **F8** unas cuantas veces más mientras te fijas en la ventana **Locales**. La ventana también contiene un botón con tres puntos. Este botón abre el cuadro de diálogo **Pila de llamadas** (ver Imagen 8.2), que muestra una lista de todas las llamadas de procedimientos activos. Una llamada a un procedimiento activo es un procedimiento que se inicia pero no se completa. También se puede activar el cuadro Pila de llamadas desde el menú **Ver – Pila de llamadas**. Esta opción sólo está disponible en el modo interrupción.

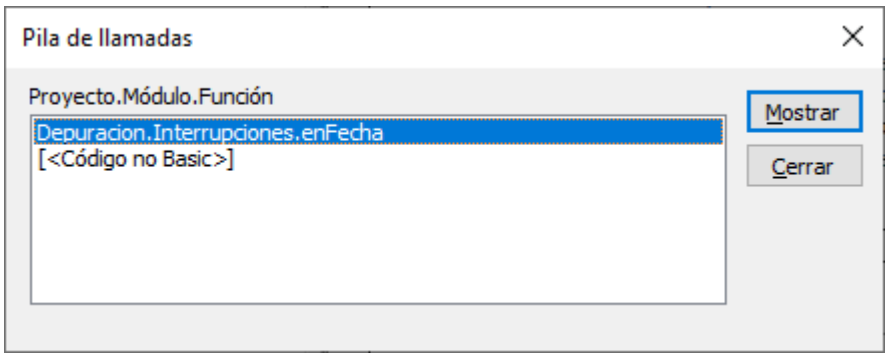


Imagen 8.2 El cuadro Pila de llamadas muestra una lista de los procedimientos que se inician pero no se completan.

Este cuadro es especialmente útil para rastrear procedimientos anidados. Recuerda que un procedimiento anidado es un procedimiento que está siendo invocado desde otro procedimiento. Si un procedimiento llama a otro, el nombre del procedimiento llamado se añade automáticamente a la lista de llamadas del cuadro **Pila de llamadas**. Cuando VBA termina de ejecutar las instrucciones del procedimiento llamado, el nombre del procedimiento se elimina automáticamente del cuadro.

Puedes utilizar el botón **Mostrar** del cuadro **Pila de llamadas** para mostrar la instrucción que llama al siguiente procedimiento que se muestra en el cuadro.

4. Presiona **F5** para seguir ejecutando el procedimiento `enFecha`.
5. Cierra la ventana **Locales**.

9 Navegar con marcadores

Mientras analizas o revisas tus procedimientos VBA, a menudo te encontrarás saltando a ciertas áreas de código. Utilizando las herramientas de marcadores, puedes marcar fácilmente los puntos del código entre los que deseas navegar.

Para establecer un marcador:

- Haz clic en cualquier parte de la declaración que quieras definir como marcador.
- Haz clic en **Edición – Marcadores – Alternar marcador**, o haz clic en el botón **Alternar marcador** en la barra de herramientas **Edición** como se muestra en la Imagen 9.1.

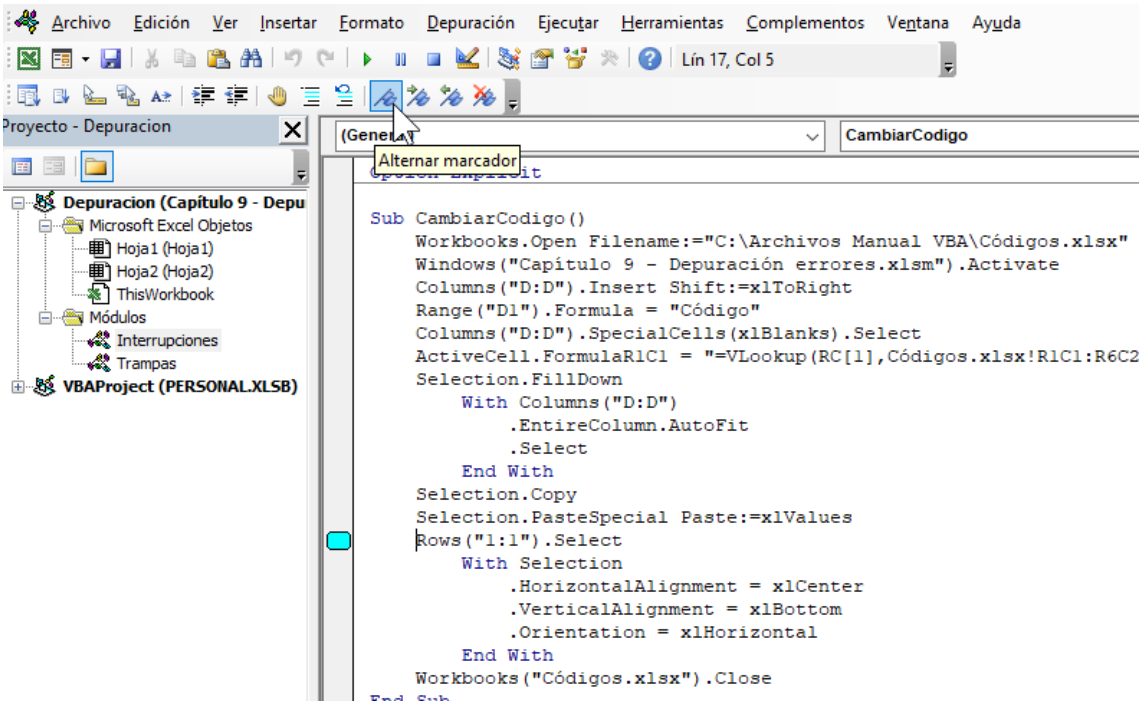


Imagen 9.1 Utilizando marcadores puedes navegar rápidamente por las secciones de código más utilizadas en tus procedimientos.

10 Errores de captura

Nadie escribe procedimientos sin errores la primera vez. Cuando se crean procedimientos VBA, hay que determinar cómo responderá el programa ante los errores. Muchos errores inesperados ocurren durante el tiempo de ejecución. Por ejemplo, un procedimiento puede intentar dar a un libro de Excel el mismo nombre que otro libro abierto en ese momento. Los errores en tiempo de ejecución son a menudo descubiertos por los usuarios que intentan hacer algo a lo que el programador no se ha anticipado.

Si se produce un error cuando el procedimiento se está ejecutando, VBA muestra un mensaje de error y el procedimiento se detiene. La mayoría de las veces, el usuario es incapaz de entender el mensaje de error que muestra VBA. Puedes evitar que los usuarios vean muchos de los errores en tiempo de ejecución si incluyes en el código de los procedimientos gestores de errores. De esta forma, cuando VBA encuentra un error, en lugar de mostrar el mensaje de error por defecto, mostrará uno mucho más amigable que puede mostrar más información.

En programación, los errores y los fallos no son lo mismo. Un fallo puede ser una declaración mal escrita, una coma mal colocada o la asignación de un valor a una variable de un tipo diferente (incompatible). Estos fallos se solucionan realizando la depuración adecuada. Pero aunque el código esté libre de fallos, no significa que no se vayan a producir errores. Un error es el resultado de un evento o una operación que no funciona como se esperaba. Por ejemplo, si un procedimiento VBA intenta acceder a un archivo en concreto en el disco y alguien lo ha eliminado o lo ha movido a otra ubicación, obtendrás un error que no esperabas. En otras palabras, un error impide que el procedimiento lleve a cabo una tarea específica.

Para implementar la gestión de errores, se utiliza la declaración `On Error`. Esta declaración le dice a VBA qué hacer si ocurre un error mientras el programa está funcionando. VBA usa la declaración `On Error` para activar un procedimiento de gestión de errores que capturará los errores de ejecución. Dependiendo del tipo de procedimiento, se pueden esquivar los errores usando alguna de las siguientes declaraciones:

```
Exit Sub.
Exit Function
Exit Property.
End Sub.
EndFunction.
End Property.
```

La declaración `On Error` se puede utilizar de tres formas diferentes, según se muestra en la siguiente tabla:

Declaración	Descripción
`On Error GoTo etiqueta`	Especifica una etiqueta a la que se debe saltar cuando se produce el error. Esta etiqueta marca el comienzo de la rutina de gestión de errores. La etiqueta debe

Declaración	Descripción
	aparecer en el mismo procedimiento que la declaración `On Error`.
`On Error Resume Next`	Cuando se produce el error, VBA ignora la línea que lo causó y no muestra ningún mensaje de error, sino que continúa el procedimiento en la siguiente línea.
`On Error GoTo 0`	Desactiva la captura de errores. Cuando VBA ejecuta esta declaración, los errores se detectan, pero no quedan capturados en el procedimiento.

10.1 Uso del objeto Err

El código de gestión de errores puede utilizar varias propiedades y métodos del objeto `Err`. Por ejemplo, para comprobar qué error se produjo, comprueba el valor de `Err.Number`. La propiedad `Number` del objeto `Err` te dirá el valor del último error que se produjo en el procedimiento, y la propiedad `Description` te devolverá la descripción del error. También puedes encontrar el nombre de la aplicación que causó el error usando la propiedad `Source` del objeto `Err` (esto es muy útil cuando el procedimiento lanza otras aplicaciones). Después de manejar el error, utiliza la instrucción `Err.Clear` para restablecer el número de error a cero. Para probar el código de gestión de errores puedes usar el método `Raise` del objeto `Err`. Por ejemplo, para detectar el error "Disco no preparado", utiliza la instrucción:

```
Err.Raise 71
```

El siguiente procedimiento muestra el uso de las declaraciones `Resume Next` y `Error`, así como el objeto `Err`.

1. Crea un nuevo módulo en el proyecto `Depuracion` del archivo "Capítulo 9 – Depuración errores.xlsm" y llámalo `Trampas`.
2. Introduce el siguiente código en el módulo.

```
Option Explicit


Sub AbrirLeer()

Dim miArchivo As String

Dim miCar As String

Dim miTexto As String

Dim ArchivoExiste As Boolean
```

```
ArchivoExiste = True

On Error GoTo GestionErrores

miArchivo = InputBox("Introduce el nombre del archivo a abrir:")
Open miArchivo For Input As #1
If ArchivoExiste Then
    Do While Not EOF(1)     ' Bucle hasta el final del archivo
      miCar = Input(1, #1)     ' Obtiene un caracter
      miTexto = miTexto + miCar ' almacena en la variable miTexto
    Loop
    Debug.Print miTexto     ' muestra en la vetnana Inmediato
    Close #1     ' Cierra el archivo
End If
Exit Sub

GestionErrores:
  ArchivoExiste = False
  Select Case Err.Number
    Case 76
    MsgBox "La ruta introducida no se ha encontrado."
    Case 53
    MsgBox "El archivo no se ha encontrado en " & _
        "la unidad especificada."
    Case 75
      Exit Sub
    Case Else
      MsgBox "Error " & Err.Number & ": " & Error(Err.Number)
    Exit Sub
  End Select
```

```
    Resume Next

End Sub
```

El objetivo del procedimiento es leer el contenido de un archivo de texto carácter a carácter. Cuando el usuario introduce un nombre de archivo, pueden producirse varios errores. Por ejemplo, el nombre del archivo o la ruta pueden se erróneas, o el usuario puede tratar de abrir un archivo que ya está abierto. Para capturar esos errores, la rutina de gestión de errores del final del procedimiento utiliza la propiedad **Number** del objeto **Err**. Existen varios métodos de lectura de un archivo de texto. En este ejemplo el procedimiento utiliza un método de bajo nivel I/O (Input/Output). Para abrir el archivo para su lectura se necesita una instrucción como esta:

```
Open miArchivo For Input As #1
```

Esta es la sintaxis general de la declaración **Open**, seguida de una explicación de cada componente:

```
Open pathname For mode [ Access access ] _
[ lock ] As [ # ] filenumber [ Len = reclength ]
```

La declaración **Open** tiene tres argumentos obligatorios: **pathname**, **mode** y **filenumber**.

- **Pathname** es el nombre del archivo que quieres abrir. Este nombre puede incluir el nombre de la unidad y la carpeta donde se encuentra.
- **Mode** es una palabra que determina la forma de abrir el archivo. Los archivos de texto pueden abrirse en uno de los siguientes métodos: **Append**, **Binary**, **Input**, **Output** o **Random**. Si no se especifica, se abre el archivo para acceso aleatorio. Utiliza el método **Input** para leer el archivo, **Output** para escribirlo, sobrescribiendo cualquier archivo existente y **Append** para escribir en el archivo agregando datos a la información existente.
- **Access** es una cláusula opcional que puede ser utilizada para especificar los permisos del archivo (lectura, escritura o ambos).
- **Lock** es un argumento opcional para determinar qué operaciones se permiten en otros procesos. Por ejemplo, si un archivo está abierto en un entorno de red, **Lock** determina cómo pueden acceder a él otras personas. Se pueden utilizar los siguiente parámetros: **Shared**, **Lock Read**, **Lock Write** o **Lock Read Write**.
- **Filenumber** es un número del 1 al 511 que se utiliza para hacer referencia al archivo en siguientes operaciones. Se puede obtener un número único de archivo usando la función **FreeFile** de VBA.
- **Reclength** especifica el tamaño del buffer (número total de caracteres) para el archivo secuenciales (de texto) o el tamaño de registro para los archivos de acceso aleatorio (archivos que se almacenan en registros de igual longitud y archivos separados por comas).

Si ya existe el archivo especificado, el procedimiento utiliza el bucle **Do … While** para decirle a VBA que ejecute las declaraciones que contiene hasta que llegue al final del

archivo. El final del archivo está determinado por el resultado de la función **EOF**. La función **Input** se utiliza para devolver el número de caracteres especificado:

```
miCar = Input(1, #1)
```

#1 es el número de archivo utilizado en el proceso de apertura de la declaración **Open**. Cada carácter que se lee se almacena en la variable **miCar**. Luego, la variable **miCar** se agrega a la variable **miTexto** de esta forma:

```
miTexto = miTexto + miCar
```

Entonces, el procedimiento escribe el contenido de la variable **miTexto** en la ventana **Inmediato** usando la declaración **Debug.Print**. Cuando el archivo ha sido leído, debemos cerrarlo con la declaración **Close**.

```
Close #1
```

El objeto **Err** contiene información sobre los errores en tiempo de ejecución. Si se produce un error mientras se está ejecutando el procedimiento, la instrucción **Err.Number** devolverá el número de error. Si se producen los errores 76, 75 o 53, almacenados dentro del bloque **Select … Case,** la instrucción **Resume Next** enviará el flujo de programación a la línea de código siguiente a la que produjo el error. Si se produce otro error, VBA devolverá el código de error (**Err.Number**) y su descripción (**Error (Err.Number)**). Al principio del procedimiento, la variable **ArchivoExiste** se establece en verdadero (**True**). De esta forma, si el programa no encuentra un error, se ejecutarán todas las instrucciones del bloque **If ArchivoExiste Then**. Sin embargo, si VBA encuentra un error, el valor de la variable **ArchivoExiste** será **False** (véase la primera instrucción en la rutina de gestión de errores, justo debajo de la etiqueta **GestionErrores**).

3. Utiliza el Bloc de notas de Windows para preparar un archivo de texto. Introduce el texto que desees en el archivo. Cuando finalices, guarda el archivo como C:\Archivos Manual VBA\Vacaciones.txt.
4. Ejecuta el procedimiento **AbrirLeer** tres veces con **F8** (modo **Paso a paso por instrucciones**). Introduce una de las siguientes variantes:
    - El nombre real del archivo: C:\Archivos Manual VBA\Vacaciones.txt.
    - Un nombre de archivo que no exista en la unidad C:\
    - Una ruta que no exista (por ejemplo K:\Pruebas).

10.2 Configuración de las opciones de captura de errores en un proyecto VBA

Es posible especificar la configuración de la gestión de errores para el proyecto actual de VBA haciendo clic en el menú **Herramientas – Opciones** y seleccionando la pestaña **General** (que se muestra en la Imagen 10.1).

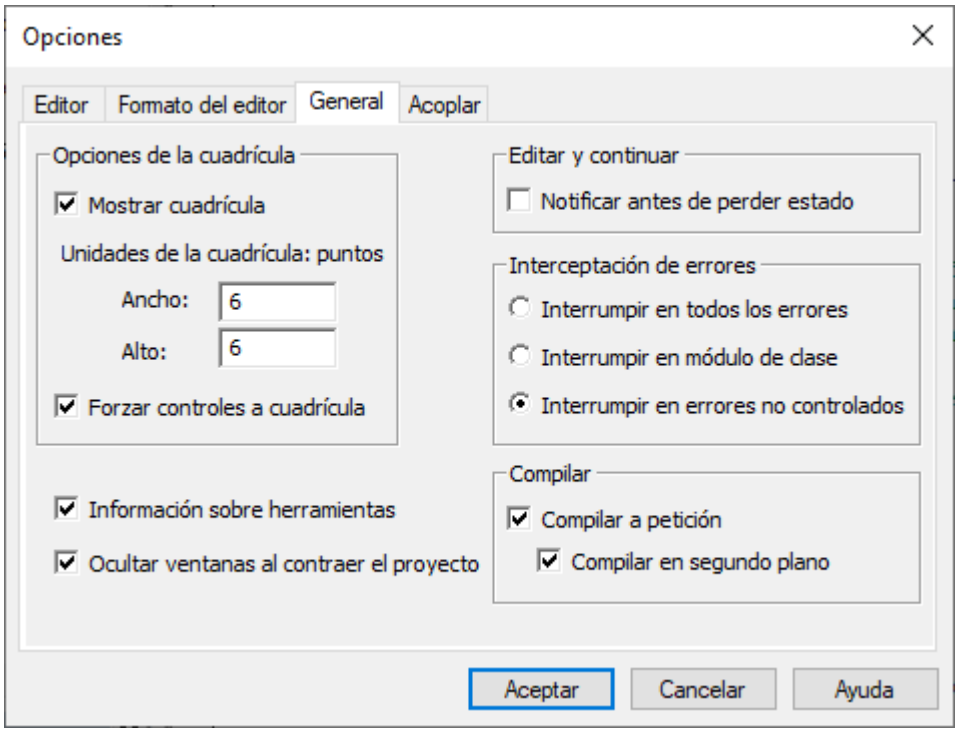


Imagen 10.1 La configuración de las opciones de captura de errores en el cuadro Opciones afectará a todas las instancias de VBA que se inicien después de cambiar la configuración.

La sección **Interceptación de errores**, ubicada en la pestaña **General**, determina cómo gestionar los errores en el entorno de VBA. Están disponibles las siguientes opciones:

- **Interrumpir en todos los errores.**
  Esta opción hará que VBA entre en modo interrupción en cualquier error si la gestión de errores está activa o si el código se encuentra en un módulo de clase.
- **Interrumpir en módulo de clase.**
  Esta opción interceptará cualquier error no gestionado en un módulo de clase. VBA activará el modo interrupción cuando ocurra un error y resaltará la línea de código en el módulo de clase que produjo el error.
- **Interrumpir en errores no controlados.**
  Esta opción capturará los errores para los que no hayas escrito un gestor de errores. El error hará que VBA active el modo interrupción en la línea de código que llamó al procedimiento de clase.

11 Navegación por pasos en los procedimientos VBA

Navegar por pasos significa ejecutar instrucciones de una en una. Esto te permite comprobar cada línea de cada procedimiento. Para comenzar a recorrer por pasos un procedimiento desde el principio, sitúa el cursor en cualquier lugar del procedimiento y haz clic en **Depuración – Paso a paso por instrucciones** o presiona **F8**.

La Imagen 11.1 muestra el menú **Depuración** que contiene varias opciones que permiten ejecutar procedimientos paso a paso.

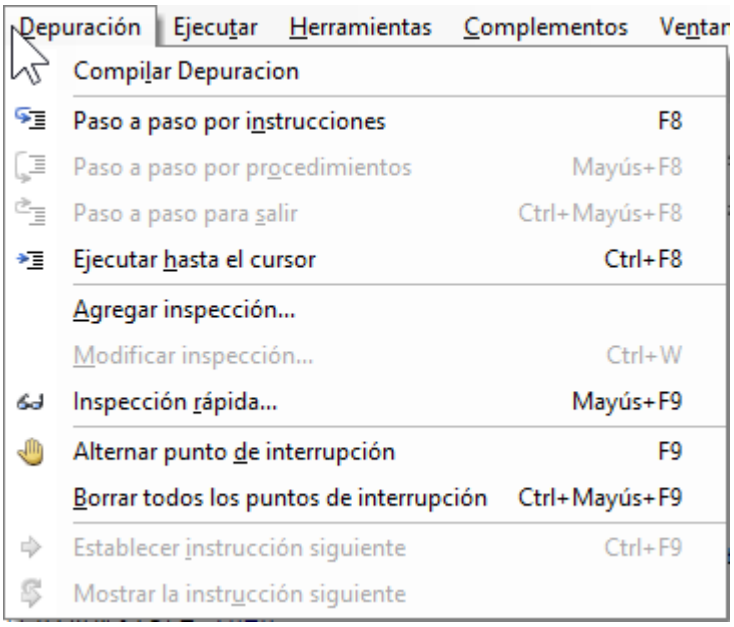


Imagen 11.1 El menú Depuración ofrece muchos comandos para recorrer los procedimientos de VBA.

Cuando se ejecuta un procedimiento de forma ininterrumpida, VBA ejecuta cada declaración hasta que se encuentra con las palabras clave `End Sub`. Si no quieres que VBA se pare en todas las líneas puedes presionar **F5** en cualquier momento para ejecutar el resto del procedimiento de forma seguida.

Practiquemos con la navegación paso a paso con el siguiente ejemplo:

1. Sitúa el cursor en cualquier punto dentro del código del procedimiento cuya ejecución deseas rastrear. Por ejemplo, prueba el procedimiento `AbrirLeer` que preparaste anteriormente.
2. Presiona **F8** o selecciona la opción **Paso a paso por instrucciones** del menú **Depuración**.
   VBA ejecuta la instrucción actual. A continuación, avanza automáticamente a la siguiente instrucción y suspende la ejecución. Mientras está en el modo interrupción, puedes activar la ventana **Inmediato**, la ventana **Inspecciones** o la ventana **Locales** para ver el efecto de una instrucción determinada en los valores de las variables y expresiones. Y si el procedimiento por el que se está pasando llama a otros procedimientos, puedes activar la ventana **Pila de llamadas** para ver qué procedimientos están activos en ese momento.
3. Presiona de nuevo **F8** para ejecutar la línea seleccionada.
   Tras ejecutar esta instrucción, VBA seleccionará la siguiente línea, y la ejecución del procedimiento se detendrá de nuevo.
4. Continúa con el procedimiento presionando **F8** o presiona **F5** para continuar la ejecución del código sin que se detenga.

11.1 Otras formas de ejecutar un procedimiento

Cuando ejecutas **Paso a paso por procedimientos** (**Mayús + F8**), VBA ejecuta cada procedimiento como si fuese una sola declaración. Esta opción es particularmente útil si un procedimiento contiene llamadas a otros procedimientos y no quieres pasar a estos

procedimientos porque ya han sido probados y depurados, o quieres concentrarte solo en el nuevo código que todavía no ha sido depurado.

Supón que una declaración del procedimiento `MiProcedimiento` (ver siguiente ejercicio) llama al procedimiento `MensajeEspecial`. Si haces clic en **Depuración – Paso a paso por procedimientos** (**Mayús + F8**), VBA ejecutará de una vez todas las instrucciones de `MensajeEspecial` y seleccionará la siguiente instrucción en el procedimiento que lo llamó (`MiProcedimiento`). Durante la ejecución del procedimiento `MensajeEspecial`, VBA continúa mostrando la ventana de código con el procedimiento actual.

1. En el módulo **Interrupciones**, busca los siguientes procedimientos:

```
Sub MiProcedimiento()
    Dim strNombre As String
    Workbooks.Add
    strNombre = ActiveWorkbook.Name
    SpecialMsg strNombre
    Workbooks(strNombre).Close
End Sub

Sub MensajeExpecial(n As String)
    If n = "Libro2" Then
        MsgBox "Por favor, cambia el nombre."
    End If
End Sub
```

2. Crea un punto de interrupción en la siguiente línea:

```
SpecialMsg strNombre
```

3. Coloca el cursor en cualquier punto del código de `MiProcedimiento` y presiona **F5** para ejecutarlo.
   VBA detiene la ejecución cuando alcanza el punto de interrupción.
4. Presiona **Mayús + F8** o haz clic en **Depuración – Paso a paso por procedimientos**.
   VBA ejecutará el procedimiento `MensajeEspecial` y se parará en la declaración siguiente a la llamada del procedimiento.
5. Presiona **F5** para terminar de ejecutar el procedimiento sin que se detenga.
6. Ahora elimina el punto de interrupción que estableciste en el paso 2.
   Saltar un procedimiento es especialmente útil cuando no quieres analizar declaraciones individuales dentro del procedimiento llamado.

   Otro comando del menú **Depuración** es **Paso a paso para salir** (**Ctrl + Mayús + F8**), que se utiliza cuando se entra en un procedimiento y luego se decide que no se quiere pasar por él. Al elegir esta opción, VBA ejecutará el resto de las instrucciones de este procedimiento en un solo paso y procederá a activar la siguiente instrucción en el

procedimiento de llamada. En el proceso de pasar por un procedimiento, puedes alternar entre Paso a paso por instrucciones, Paso a paso por procedimiento y Paso a paso para salir. La opción que selecciones depende del fragmento de código que desees analizar en cada momento.

El comando **Ejecutar hasta el cursor** del menú **Depuración** (**Ctrl + F8**) te permite ejecutar tu procedimiento hasta que se encuentra la línea que tienes seleccionada. Este comando es útil cuando quieres detener la ejecución antes de un bucle grande o si tienes la intención de saltar algún procedimiento al que se va a llamar.

Ahora supón que quieres ejecutar `MiProcedimiento` hasta la línea del procedimiento que llama al procedimiento `MensajeEspecial`.

7. Haz clic dentro de la declaración `SpecialMsg strNombre`.
8. Ahora haz clic en el menú **Depuración – Ejecutar hasta el cursor**. VBA detendrá la ejecución del código de `MiProcedimiento` cuando llegue hasta esta línea.
9. Presiona **Mayús + F8** para pasar al procedimiento `MensajeEspecial`.
10. Presiona **F5** para ejecutar las declaraciones restantes del procedimiento.

11.2 Establecer instrucción siguiente

A veces puede que desees volver a ejecutar líneas de código anteriores en el procedimiento o saltarte una parte del código que esté causando problemas. En cada una de estas situaciones puedes usar la opción **Establecer instrucción siguiente** en el menú **Depuración**. Cuando se detiene la ejecución de un procedimiento, se puede reanudar desde cualquier declaración que desees. VBA omitirá la ejecución de las líneas entre la instrucción seleccionada y la instrucción en que se suspendió la ejecución. Supón que en `MiProcedimiento` (el procedimiento anterior) has establecido un punto de interrupción en la instrucción que llama al procedimiento `MensajeEspecial`. Para saltar la ejecución del procedimiento `MensajeEspecial` puedes colocar el cursor en la instrucción `Workbooks(strNombre).Close` y presionar **Ctrl + F9** (o hacer clic en el menú **Depuración – Establecer instrucción siguiente**).

No es posible utilizar **Establecer instrucción siguiente**, a menos que hayas suspendido la ejecución del procedimiento. Aunque saltarse líneas de código puede ser muy útil en el proceso de depuración de los procedimientos de VBA, debe hacerse con cuidado. Cuando utilizas la opción **Establecer instrucción siguiente**, le dices a VBA que esta es la línea que quieres ejecutar a continuación. Todas las líneas intermedias se ignoran. Esto significa que no ocurrirán ciertas cosas, lo que puede llevar a errores inesperados.

11.3 Mostrar la instrucción siguiente

Si no estás seguro de a partir de qué instrucción se reanudará la ejecución del procedimiento, puedes hacer clic en **Depuración – Mostrar la instrucción siguiente** para que VBA coloque el cursor en la línea que se ejecutará a continuación. Esto es realmente útil cuando has estado mirando otros procedimientos y no está seguro de dónde se reanudará la ejecución. La opción **Mostrar la instrucción siguiente** sólo está disponible en el modo interrupción.

11.4 Parar y reiniciar procedimientos

En cualquier momento, al pasar por el código de un procedimiento en la ventana **Código** podemos:

- Presionar **F5** para ejecutar las instrucciones sin necesidad de ir línea por línea.
- Seleccionar **Ejecutar – Restablecer** para finalizar el procedimiento sin ejecutar las líneas restantes.

Al restablecer el procedimiento todas las variables pierden sus valores actuales. Las variables numéricas toman el valor de cero, las cadenas de longitud variable se restablecen con una cadena de longitud cero (""), y las cadenas de longitud fija se rellenan con el carácter representado por el código ASCII o Chr(0). Las variables del tipo Variant se restablecen en `Empty` y las variables de objeto se restablecen en `Nothing`.

12 Finalizar un procedimiento en función de una condición

Vamos a abordar este problema ahora que tenemos más conocimientos de VBA. Este es el procedimiento `Insertar_nueva_hoja` tal y como lo modificamos en el Capítulo 1.

```
Sub Insertar_nueva_hoja()
'
' Insertar_nueva_hoja Macro
' Inserta una nueva hoja en el libro y le cambia el nombre.
'
'
    Sheets.Add after:=ActiveSheet
    ActiveSheet.Name = Application.InputBox _
    ("Introduce un nombre para la hoja", "Renombrar la hoja")
End Sub
```

El método `InputBox` es un miembro del objeto `Application` de Excel y esto requiere que precedamos su nombre con el nombre del objeto (`Application`). Debemos tener en cuenta que el código que se muestra a continuación utiliza el carácter de continuación de línea (un guion bajo) para separar la declaración larga que podemos obtener al introducir los argumentos al método. Encontraremos la lista de argumentos y sus descripciones en el capítulo 4. Uno de los argumentos que debemos añadir al método `InputBox` para obtener los resultados esperados con el botón **Cancelar** se llama "tipo" y especifica el tipo de datos devueltos cuando el usuario hace clic en el botón **Cancelar**.

El método `InputBox` devuelve `False`. Por lo tanto, necesitamos introducir alguna lógica condicional para probar el tipo de retorno. También necesitamos evitar que el usuario introduzca espacios en blanco para el nombre de la hoja. A estas alturas ya debemos estar familiarizados con la escritura de declaraciones condicionales en VBA. La lógica condicional nos permitirá hacer muchas mejoras en el código de la macro grabada. Veamos el procedimiento revisado de `Insertar_nueva_hoja_rev`.

```
Sub Insertar_nueva_hoja_rev()
```

```
'
' Inserta una nueva hoja. Versión revisada
' Inserta y renombra una hoja
'
    Dim valorUsuario As Variant
    valorUsuario = Application.InputBox _
    ("Introduce el nombre de la hoja:", _
    "Renombrar hoja", , , , , , 2)

    If valorUsuario = False Then
        MsgBox ("Has presionado el botón Cancelar." & _
        "El procedimiento finalizará.")
        sFlag = True
        Exit Sub
            ElseIf valorUsuario = "" Or Trim(valorUsuario) = ""
Then
                MsgBox "Por favor introduce le nombre de la hoja
o haz clic en Cancelar para salir."
                Insert_NewSheet
            Else
                Sheets.Add After:=ActiveSheet
                ActiveSheet.Name = valorUsuario
    End If
End Sub
```

Tengamos en cuenta que guardaremos el nombre de la hoja introducida por el usuario en la variable **valorUsuario**. Esta variable se declara como Variant porque el método **InputBox** puede devolver diferentes tipos de datos y queremos que Excel los maneje por nosotros.

Comenzamos pidiendo al usuario el nombre de la hoja. Primero, debemos definir el mensaje que se mostrará al usuario. A continuación, especificamos el texto que aparece en la barra de título del cuadro de diálogo. Los cinco argumentos siguientes no son relevantes, por lo que se introducen comas (se puede prescindir de las comas si se introducen las etiquetas de los argumentos como vimos en el Capítulo 4.)

Lo que nos importa es el último argumento. El valor **2** especifica que esperamos obtener una cadena de texto. Una vez que el resultado de la interacción del usuario con el método **InputBox** está almacenado en la variable **valorUsuario** es el momento de las instrucciones **If**. Si el contenido de la variable es **False**, entonces queremos mostrar un mensaje al usuario y finalizar el procedimiento.

Sabemos que podemos salir de un procedimiento VBA antes de que finalice utilizando **Exit Sub**. Sin embargo, antes de finalizarlo, podemos querer almacenar alguna información vital en variables adicionales. En el caso del procedimiento **Insertar_nueva_hoja_rev**, debemos recordar que hemos salido del procedimiento, por lo que no ejecutamos otros procedimientos

que puedan depender de éste. Recordemos que nuestro procedimiento **Insertar_nueva_hoja** forma parte de un procedimiento maestro más grande que también tendrá que ser revisado. La variable **sFlag** tendrá un valor booleano de **True** si el usuario hizo clic en Cancelar, y **False** en caso contrario.

Somos libres de elegir los nombres de las variables. Observemos que la variable **sFlag** no se declara en ninguna parte del procedimiento. Dado que también debemos utilizarla en el procedimiento maestro **Crear_hoja_empleado**, necesitamos una declaración de ámbito a nivel de proyecto. En el Capítulo 3 aprendimos que las variables públicas pueden ser usadas en cualquier módulo. Este es el momento perfecto para utilizarlas. La Imagen 12.1 muestra el procedimiento revisado de **Crear_hoja_empleado** del Capítulo 1. Fijémonos en la declaración de la variable **sFlag** en la parte superior del módulo. La primera línea de código en el procedimiento se asegura de que **sFlag** se ponga en False cuando empecemos. Cuando el procedimiento **Insertar_nueva_hoja_rev** haya terminado de ejecutarse, **sFlag** será True si el usuario hace clic en Cancelar. De nuevo podemos usar la instrucción **Exit Sub** para detener la ejecución de más código. Y si **sFlag** es **False** continuaremos con las demás instrucciones.

Observemos que el procedimiento **Insertar_nueva_hoja** también comprueba en la cláusula **ElseIf** si el usuario hizo clic en Aceptar sin introducir ningún dato o si introdujo uno o varios espacios. La función **Trim** de VBA elimina los espacios anteriores y posteriores de una cadena de texto dada. Si el valor de la variable **valorUsuario** es una cadena vacía (“”), entonces se muestra un mensaje al usuario y se vuelve a llamar al procedimiento. De esta forma el usuario tiene la posibilidad de introducir los datos necesarios o hacer clic en Cancelar. Finalmente, si todo se ve bien, entonces se ejecutan las declaraciones de la cláusula **Else**. Se inserta una nueva hoja después de la hoja actual y se le da un nuevo nombre con el texto almacenado en la variable **valorUsuario**.Modifica los procedimientos tratados en esta sección y utilízalos para practicar todas las técnicas de depuración de errores del capítulo.

```
Public sFlag

Sub Crear_hoja_empleado()

sFlag = False
    Insertar_nueva_hoja
    Insertar_encabezados
    Datos_empleados
    Obtener_nombre
    Obtener_apellido
    Calcular_importes
    Formato_tabla
End Sub
```

Imagen 12.1 El procedimiento revisado utiliza una variable pública.

Asegúrate de ejecutar el procedimiento maestro al menos tres veces para comprobar todas las condiciones utilizadas en el procedimiento `Insertar_nueva_hoja`.

13 Resumen

En este capítulo has aprendido a gestionar los errores y a probar procedimientos VBA para asegurarte de que funcionan según lo planeado. Depuraste código mediante puntos de interrupción e inspecciones. Aprendiste a trabajar con la ventana **Inmediato** en modo interrupción y descubriste cómo la ventana **Locales** puede ayudarte a monitorear los valores de las variables. También aprendiste que el cuadro de diálogo **Pila de llamadas** puede ser útil para llevar un registro de dónde estás en todo momento cuando se ejecuta un procedimiento complejo.

Utilizando las herramientas de depuración integradas puedes localizar rápidamente los puntos problemáticos.

Mi recomendación es que intentes dedicar el tiempo suficiente a familiarizarte con los comandos del menú **Depuración** y las herramientas que se tratan en este capítulo, pues dominar el arte de la depuración puede ahorrarte mucho tiempo al buscar errores.

Capítulo 10
Manipulación de archivos y carpetas con VBA

Mientras trabajas seguramente has accedido, creado, renombrado, copiado y eliminado cientos de archivos y carpetas. Sin embargo, probablemente nunca has realizado estas tareas de forma automática. Por lo tanto, esta es tu oportunidad. Este capítulo se centra en las funciones e instrucciones de VBA que se ocupan específicamente de archivos y carpetas. Al finalizar el capítulo podrás:

- Averiguar el nombre de la carpeta actual (función **`CurDir`**).
- Cambiar el nombre de un archivo o carpeta (función **`Name`**).
- Comprobar si existe un archivo o carpeta en un disco (función **`Dir`**).
- Averiguar la fecha y la hora en que se modificó por última vez un archivo (función **`FileDateTime`**).
- Obtener el tamaño de un archivo (función **`FileLen`**).
- Comprobar y cambiar los atributos de los archivos (funciones **`GetAttr`** y **`SetAttr`**).
- Cambiar la carpeta o unidad predeterminada (instrucciones **`ChDir`** y **`ChDrive`**).
- Crear y borrar una carpeta (instrucciones **`MkDir`** y **`RmDir`**).
- Copiar y eliminar un archivo o carpeta (instrucciones **`FileCopy`** y **`Kill`**).

1 Manipular archivos y carpetas

En esta sección se examinan un conjunto de funciones VBA utilizadas para realizar operaciones con archivos y carpetas.

1.1 Averiguar el nombre de la carpeta activa

Cuando trabajamos con archivos, a menudo es necesario averiguar el nombre de la carpeta donde se encuentra el libro de Excel. Esta información se puede obtener fácilmente mediante la función **`CurDir`**. Tiene una apariencia similar a esta:

```
CurDir([unidad])
```

Tengamos en cuenta que la unidad es un argumento opcional. Si la omitimos, VBA utilizará la unidad actual. La función **CurDir** devuelve una ruta de archivo como Variant. Para devolver la ruta como String, utilizamos **CurDir$** (**$** es el tipo de carácter de declaración para cadenas de texto).

Para ver esta función en acción, vamos a realizar un par de ejercicios en la ventana **Inmediato**.

1. Crea un libro nuevo y guárdalo como "Capítulo 10 – Archivos y carpetas.xlsm".
2. Abre el editor de VBA con **Alt + F11** y presiona **Ctrl + G** para activar la ventana **Inmediato**. A continuación, escribe la siguiente instrucción y presiona **Intro**.

   ```
   ?CurDir
   ```

   Cuando presionas **Intro**, VBA muestra el nombre de la carpeta actual:
   C:\Archivos Manual VBA
3. Si tienes una segunda unidad de disco (o una unidad de CD-ROM), puedes encontrar la carpeta actual en la unidad D de la siguiente forma:

   ```
   ?CurDir("E:\")
   ```

   Si introduces una letra de una unidad que no existe, VBA mostrará el mensaje de error "Dispositivo no disponible".
4. Para almacenar el nombre de la unidad de disco actual en una variable llamada **miUnidad**, escribe la siguiente declaración y presiona **Intro**:

   ```
   miUnidad=Left(CurDir$,1)
   ```

   Al presionar **Intro**, VBA almacena la letra de la unidad actual en la variable **miUnidad**. Fíjate en cómo la función **CurDir$** se utiliza como primer argumento de la función **Left**. La función **Left** le dice a VBA que extraiga el carácter más a la izquierda de la cadena devuelta por la función **CurDir$** y lo almacene en la variable **miUnidad**.
5. Para comprobar el contenido de la variable **miUnidad**, escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro**:

   ```
   ?miUnidad
   ```
6. Para hacer que se devuelva la letra de la unidad seguida de los dos puntos que la acompañan, escribe las siguiente instrucciones, presionando **Intro** después de cada una:

   ```
   miUnidad=Left(CurDir$,1)
   ?miUnidad
   ```

1.2 Cambiar el nombre a un archivo o una carpeta

Para cambiar el nombre de un archivo o carpeta, utilizamos la función **Name** de la siguiente forma:

```
Name Nombre_antiguo As Nombre_nuevo
```

Nombre_antiguo es la ruta actual seguida del nombre del archivo o carpeta que vamos a renombrar. **Nombre_nuevo** especifica la nueva ruta y el nombre del archivo o carpeta.

Con la función **Name**, podemos mover un archivo de una carpeta a otra (no es posible mover una carpeta). A continuación, se indican algunas precauciones que debemos tener en cuenta al trabajar con la función **Name**:

- El nombre del archivo de **Nombre_nuevo** no puede referirse a un archivo que ya existe.
  Supongamos que queremos cambiar el nombre del archivo system.txt a test.txt. Esto se hace fácilmente con la siguiente declaración:

  ```
  Name "C:\system.txt" As "C:\test.txt"
  ```

  Sin embargo, si el archivo C:\test.txt ya existe en la unidad C, VBA mostrará el mensaje de error "El archivo ya existe". Del mismo modo, mostrará el error "Archivo no encontrado" si el archivo que deseamos renombrar no existe. Probamos la instrucción anterior en la ventana **Inmediato** (sustituyendo los nombres del ejemplo por nombres de archivos y carpetas reales).

- Si **Nombre_nuevo** ya existe, y es diferente de **Nombre_viejo**, la función **Name** mueve el archivo a una nueva carpeta y cambia su nombre si es necesario.
  Name "C:\system.txt" As "D:\N-test.txt"
  Si el archivo test.txt no existe en el directorio raíz de la unidad D, VBA mueve el archivo C:\system.txt a la unidad especificada; sin embargo, no renombra el archivo.
- Si **Nombre_nuevo** y **Nombre_antiguo** se refieren a diferentes directorios y los dos nombres de archivos son los mismos, la función **Name** mueve el archivo especificado a una nueva ubicación sin cambiar el nombre del archivo.

  ```
  Name "D:\test.txt" as "C:\Archivos Manual VBA\NTest.txt"
  ```

  La instrucción anterior mueve el archivo test.txt a la carpeta de la unidad C:

Atención:

Para poder renombrar un archivo abierto, primero debemos cerrarlo. También hay que tener en cuenta que no podemos utilizar como nombres de archivo determinados caracteres como *, ?, \, etc.

1.3 Comprobar la existencia de un archivo o carpeta

La función **Dir**, que devuelve el nombre de un archivo o carpeta tiene la siguiente sintaxis:

```
Dir [(nombre_ruta[, atributos])]
```

Observa cómo los dos argumentos de la función son opcionales. **Nombre_ruta** es el nombre del archivo o carpeta. Podemos utilizar una de las constantes o valores de la siguiente tabla para el argumento **atributos**:

Constante	Valor	Atributo
vbNormal	0	Normal
vbHidden	2	Oculto
vbSystem	4	Sistema
vbVolume	8	Etiqueta Volumen
vbDirectory	16	Directorio o carpeta

La función `Dir` se utiliza a menudo para comprobar si un archivo o una carpeta existe en una unidad de disco.

Si el archivo o carpeta no existe, la función devuelve la cadena vacía (“”) (ver el paso 3 del siguiente ejercicio).

Probemos la función `Dir` en varios ejercicios en la ventana **Inmediato**.

1. En la ventana **Inmediato** escribe la siguiente declaración y presiona **Intro**.

   ```
   ?Dir("C:\", vbNormal)
   ```

   Cuando presionas **Intro**, VBA devuelve el nombre del archivo en la carpeta especificada.
2. Para mostrar los nombres de otros archivos en el directorio actual, escribe la función `Dir` sin argumentos y presiona **Intro**:

   ```
   ?Dir
   ```
3. Introduce las siguientes instrucciones en la ventana **Inmediato** y examina los resultados al presionar **Intro**:

   ```
   miarchivo = Dir("C:\", vbHidden)
   ?miarchivo
   miarchivo = Dir
   ?miarchivo
   ```
4. Escribe la siguiente instrucción:

   ```
   If Dir("C:\correo.bat") = "" Then Debug.Print _
   "Archivo no encontrado."
   ```

   Como el archivo correo.bat no existe en la unidad C, VBA muestra el mensaje “Archivo no encontrado” en la ventana **Inmediato**.
   La función `Dir` permite usar comodines en la ruta (un asterisco (`*`) para varios caracteres y una interrogación (`?`) para uno solo). Por ejemplo, para buscar todos los archivos del Panel de control en la carpeta Windows \System32, puedes buscar todos

los archivos con extensión .msc como se muestra a continuación (las líneas en color verde muestran las respuestas a las instrucciones):

```
?Dir("C:\WINDOWS\System32\*.msc", vbNormal)
azman.msc
?dir
certlm.msc
?dir
certmgr.msc
```

Probemos ahora un par de procedimientos completos que usan la función `Dir`. ¿Qué ocurre si escribimos los nombres de los archivos del directorio especificado en la ventana **Inmediato** y en la hoja de cálculo? Haremos que los datos mostrados sean consistentes usando la función `LCase$`, que hace que los nombres de los archivos aparezcan en minúsculas.

1. Abre el archivo **Capítulo 10 – Archivos y carpetas.xlsm**. A continuación, abre el editor de VBA.
2. Cámbiale el nombre al proyecto por `GestionArchivos_VBA`.
3. Inserta un nuevo módulo en el proyecto y llámalo `Funcion_Dir`.
4. Introduce el siguiente procedimiento en el módulo:

```
Sub MisArchivos()
    Dim miarchivo As String
    Dim miruta As String
    Dim mensaje As String

    mensaje = "Introduce la ruta, "
    mensaje = mensaje & "por ejemplo C:\Archivos Manual VBA"
    miruta = InputBox(mensaje)
    If Right(miruta, 1) <> "\" Then miruta = miruta & "\"
    miarchivo = Dir(miruta & "*.*")
    If miarchivo <> "" Then Debug.Print _
    "Archivos en la carpeta" & miruta
    Debug.Print LCase$(miarchivo)
    If miarchivo = "" Then
        MsgBox "No se encontraron archivos."
    Exit Sub
    End If
    Do While miarchivo <> ""
        miarchivo = Dir
        Debug.Print LCase$(miarchivo)
    Loop
End Sub
```

El procedimiento anterior pregunta al usuario por la ruta de acceso. Si la ruta no termina con la barra invertida, la función `Right` la agrega al final de la cadena. A continuación, VBA busca todos los archivos (`*`) en la ruta de acceso especificada. En caso de que no existan archivos, se muestra un mensaje. Si existen archivos, los nombres de estos aparecen en la ventana **Inmediato**.

5. Ejecuta el procedimiento.
6. Para mostrar los nombres de los archivos en una hoja de cálculo, introduce el procedimiento `ObtenerArchivos` en el mismo módulo en el que introdujiste el procedimiento `MisArchivos` y luego ejecútalo.

```
Sub ObtenerArchivos()
    Dim miarchivo As String
    Dim filaSiguiente As Integer

    filaSiguiente = 1
    With Worksheets("hoja1").Range("A1")
        miarchivo = Dir("C:\Archivos Manual VBA\*.*", vbNormal)
        .Value = miarchivo
        Do While miarchivo <> ""
            miarchivo = Dir
            .Offset(filaSiguiente, 0).Value = miarchivo
            filaSiguiente = filaSiguiente + 1
        Loop
    End With
End Sub
```

Este procedimiento obtiene los nombres de los archivos ubicados en el directorio especificado y escribe todos sus nombres en una hoja.

1.4 Averiguar la fecha de modificación de un archivo

Si un procedimiento debe comprobar cuándo se modificó un archivo por última vez, utilizamos la función `FileDateTime` de la siguiente manera:

```
FileDateTime(ruta)
```

`ruta` es una cadena que especifica el archivo con que queremos trabajar. La ruta de acceso puede incluir la unidad y la carpeta donde se encuentra ubicado el archivo. La función devuelve la fecha y la hora de modificación del archivo. El formato de fecha y hora depende de la configuración regional seleccionada en el Panel de control de Windows. Practiquemos el uso de esta función en la ventana **Inmediato**:

1. Introduce la siguiente instrucción en la ventana **Inmediato**:

```
?FileDateTime("C:\Archivos Manual VBA\Capítulo 10 - Archivos y carpetas.xlsm")
```

Cuando presionas **Intro**, VBA devuelve la fecha y la hora en el siguiente formato (el equipo en el que se ha ejecutado tiene la configuración regional Español (España)):

```
07/04/2020 11:33:37
```

Para hacer que VBA devuelva la fecha y la hora por separado utiliza la función `FileDateTime` como argumento de las funciones `DateValue` o `TimeValue`. Por ejemplo, introduce las siguientes instrucciones en la ventana **Inmediato**:

```
?DateValue(FileDateTime("C:\Archivos Manual VBA\Capítulo 10 - Archivos y carpetas.xlsm"))
?TimeValue(FileDateTime("C:\Archivos Manual VBA\Capítulo 10 - Archivos y carpetas.xlsm"))
```

2. Introduce la siguiente instrucción en la ventana **Inmediato**:

```
If DateValue(FileDateTime("C:\Archivos Manual VBA\Capítulo 10 - Archivos y carpetas.xlsm")) < Date then Debug.Print "Este archivo no ha sido modificado hoy."
```

La función `Date` devuelve la fecha actual del sistema tal y como está configurada en el cuadro de diálogo **Propiedades de fecha y hora** al que se accede a través del Panel de control de Windows.

1.5 Averiguar el tamaño del archivo

Para comprobar el tamaño de un archivo, utilizamos la función `FileLen` de la siguiente forma:

```
FileLen(ruta)
```

`FileLen` devuelve el tamaño del archivo especificado en bytes. Si el archivo se encuentra abierto, VBA devuelve el tamaño del archivo cuando se guardó por última vez.

1.6 Obtener y configurar los atributos del archivo

Los archivos y carpetas pueden tener atributos como “solo lectura”, “oculto”, “del sistema” y “archivo”. Para conocer los atributos de un archivo o carpeta, utilizamos la función `GetAttr`, que devuelve un número entero que representa la suma de una o más de las constantes que se muestran en la siguiente tabla. El único argumento de esta función es el nombre del archivo o carpeta con el que se quiere trabajar.

Constante	Valor	Atributo
vbNormal	0	Normal (no se han establecido otros atributos).
vbReadOnly	1	Solo lectura (no se puede modificar).

vbHidden	2	Oculto (no es visible desde la configuración estándar).
vbSystem	4	Archivo del sistema.
vbDirectory	16	El objeto es un directorio.
vbArchive	32	Archivo.

Para averiguar si un archivo tiene alguno de los atributos que se muestran en la tabla, utiliza el operador **AND** para comparar el resultado de la función **GetAttr** con el valor de la constante. Si la función devuelve un valor distinto de cero, el archivo o carpeta especificado en la ruta de acceso tiene el atributo para el que se está realizando la prueba.

1. Inserta un módulo nuevo en el proyecto GestionArchivos_VBA y llámalo **Funcion_GetAttr**.
2. Introduce el siguiente procedimiento y ejecútalo:

```
Sub ObtenerAtributos()
    Dim Atributo As Integer
    Dim Mensaje As String
    Dim strNombreArchivo As String
    strNombreArchivo = _
    InputBox("Introduce el nombre completo del archivo:", _
    "Unidad\Carpeta\Archivo")
    If strNombreArchivo = "" Then Exit Sub
    Atributo = GetAttr(strNombreArchivo)
    Mensaje = ""
    If Atributo And vbReadOnly Then Mensaje = Mensaje & _
    "Sólo lectura"
    If Atributo And vbHidden Then Mensaje = Mensaje & _
    Chr(10) & "Oculto"
    If Atributo And vbSystem Then Mensaje = Mensaje _
    & Chr(10) & "Sistema"
    If Atributo And vbArchive Then Mensaje = Mensaje _
    & Chr(10) & "Archivo"
    MsgBox Mensaje, , strNombreArchivo
End Sub
```

La función opuesta a **GetAttr** es **SetAttr**, que permite establecer los atributos de los archivos o carpetas (cuando están cerrados). Su sintaxis es:

```
SetAttr ruta, atributos
```

ruta es una cadena de texto que especifica el archivo o la carpeta con la que queremos trabajar.

atributos hace referencia a una o más constantes que especifican los atributos que queremos establecer. Podemos consultar la tabla anterior para ver la lista de constantes.

Supongamos que tenemos un archivo llamado correo.txt y queremos establecer dos atributos: solo lectura y oculto.

1. Para establecer los atributos del archivo, escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro** (reemplaza el nombre del archivo con uno que exista en tu equipo):

   ```
   SetAttr "C:\correo.txt", vbReadOnly + vbHidden
   ```

2. Para ver los atributos que has establecido en el paso anterior, escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro** (comprueba el valor devuelto en la tabla anterior):

   ```
   ?GetAttr("C:\correo.txt")
   ```

1.7 Cambiar la unidad o carpeta predeterminada

Podemos cambiar fácilmente la carpeta predeterminada utilizando la instrucción **ChDir** de esta forma:

```
ChDir ruta
```

En esta instrucción, **ruta** es el nombre de la nueva carpeta predeterminada. **Ruta** puede incluir el nombre de la unidad de disco. En caso de que no la incluya, la carpeta predeterminada se cambiará en la carpeta actual. La unidad no cambiará.

Supongamos que la carpeta predeterminada es C:\Windows. La declaración

```
ChDir "D:\Mis Documentos"
```

cambia la carpeta predeterminada a D:\"Mis Documentos", sin embargo la unidad actual continúa siendo la unidad C.

Para cambiar la unidad actual, utiliza la declaración **ChDrive** en el siguiente formato:

```
ChDrive unidad
```

El argumento **unidad** especifica la unidad predeterminada. Por ejemplo, para cambiar la unidad predeterminada por D ó E utilizamos las siguientes declaraciones:

```
ChDrive "D"
ChDrive "E"
```

Si hacemos referencia a una unidad que no existe obtendremos el mensaje de error "Dispositivo no disponible".

1.8 Crear y eliminar carpetas

Podemos crear nuevas carpetas utilizando la siguiente sintaxis de la instrucción **MkDir**:

```
MkDir ruta
```

ruta especifica la nueva carpeta que vamos a crear. Si no incluimos el nombre de la unidad, VBA creará la carpeta en la unidad actual.

Para eliminar una carpeta que ya no necesitamos, utilizamos la función **RmDir**. Tiene esta sintaxis:

```
RmDir ruta
```

En este caso, **ruta** especifica la carpeta que deseamos eliminar. Igual que cuando la creamos, esta ruta puede incluir el nombre de la unidad. Si lo omitimos, VBA eliminará la carpeta en la unidad actual en caso de que exista una carpeta con el mismo nombre. De lo contrario, VBA mostrará el mensaje de error "No se ha encontrado la ruta de acceso".

Vamos a probar con algunos ejemplos en la ventana **Inmediato**:

1. Escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro** para crear una carpeta llamada "Correo" en la unidad C:.

   ```
   MkDir "C:\Correo"
   ```

2. Para cambiar la carpeta predeterminada a C:\Correo, introduce la siguiente instrucción y presiona **Intro**:

   ```
   ChDir "C:\Correo"
   ```

3. Para conocer el nombre de la carpeta activa introduce la siguiente instrucción y presiona **Intro**:

   ```
   ?CurDir
   ```

4. Para eliminar la carpeta C:\Correo creada en el paso 1 introduce las siguientes instrucciones en la ventana Inmediato y presiona **Intro**:

   ```
   ChDir "C:\"
   RmDir "C:\Correo"
   ```

Atención:

No es posible eliminar una carpeta si todavía contiene archivos. En primer lugar, debemos eliminar esos archivos con la instrucción **Kill** (la veremos más adelante) y posteriormente eliminar la carpeta con **RmDir**.

1.9 Copiar archivos

Para copiar archivos de una carpeta a otra, utilizamos la declaración **FileCopy** como se muestra a continuación:

```
FileCopy origen, destino
```

Origen especifica el nombre del archivo que queremos copiar. El nombre puede incluir la unidad en la que se ubica el archivo.

Destino es el nombre de la carpeta de destino. Ambos parámetros son obligatorios.

Supongamos que queremos copiar un archivo especificado por un usuario a una carpeta llamada C:\Documentos archivados. En el siguiente ejercicio se muestra cómo hacerlo:

1. Inserta un nuevo módulo y llámalo **CopiaArchivoElimina**.
2. Copia el siguiente código dentro de módulo:

```
Sub CopiarADocuArchivados()
    Dim carpeta As String
    Dim origen As String
    Dim destino As String
    Dim msj1 As String
    Dim msj2 As String
    Dim p As Integer
    Dim s As Integer
    Dim i As Long

    On Error GoTo GestorErrores
    carpeta = "C:\Documentos Archivados"
    msj1 = "El archivo seleccionado ya se encuentra en la carpeta."
    msj2 = "ha sido copiado a"
    p = 1
    i = 1
    ' se obtiene el nombre del archivo del usuario
    origen = Application.GetOpenFilename
    ' no hace nada si se cancela
    If origen = "False" Then Exit Sub
    ' obtiene el número total de barras invertidas en el origen
    ' contenidos de la variable
    Do Until p = 0
        p = InStr(i, origen, "\", 1)
        If p = 0 Then Exit Do
        s = p
        i = p + 1
    Loop
    ' crea el nombre del archivo de destino
    destino = carpeta & Mid(origen, s, Len(origen))
    ' crea la nueva carpeta con este nombre
    MkDir carpeta
    ' comprueba si el archivo ya existe en
    ' la carpeta de destino
```

```
    If Dir(destino) <> "" Then
        MsgBox msj1
    Else
        ' copia el archivo seleccionado a la carpeta C:\Documentos
         'Archivados
        FileCopy origen, destino
        MsgBox origen & " " & msj2 & " " & destino
    End If
    Exit Sub
GestorErrores:
    If Err = "75" Then
        Resume Next
    End If
    If Err = "70" Then
        MsgBox "No se puede copiar un archvo abierto."
        Exit Sub
    End If
End Sub
```

Este procedimiento utiliza el método `GetOpenFilename` del objeto `Application` para obtener el nombre del archivo del usuario. Este método hace que aparezca el cuadro de diálogo **Abrir** (el predeterminado). Usando este cuadro de diálogo puedes elegir cualquier archivo, en cualquier directorio y en cualquier unidad de disco. Si el usuario lo cancela, VBA devuelve el valor `False` y el procedimiento termina. Si el usuario selecciona un archivo y hace clic en **Abrir**, el archivo seleccionado se asignará a la variable `origen`.

Para copiar, solo necesitarás el nombre del archivo (sin la ruta), para que el bucle `Do … Loop Until` encuentre la posición de la última barra invertida (\) almacenada en la variable `origen`, el primer argumento de la instrucción `FileCopy`.

A continuación, VBA prepara una cadena de caracteres y la asigna a la variable destino, que es el segundo argumento de la instrucción `FileCopy`. Esta variable contiene la variable obtenida por la concatenación del nombre de la carpeta de destino (C:\Documentos Archivados) con el nombre del archivo especificado por el usuario, precedido de una barra invertida.

La función `MkDir` crea una nueva carpeta llamada C:\Documentos Archivados si no existe en la unidad C. Si dicha carpeta ya existe, VBA tendrá que gestionar el error 75. Este error será interceptado por el gestor de errores incluido al final del procedimiento. Observa que el gestor de errores es un fragmento de código que comienza con la etiqueta `GestorErrores` seguida de dos puntos.

Cuando VBA se encuentra con la declaración `Resume Next`, continuará ejecutando el procedimiento a partir de la instrucción siguiente a la que causó el error. Esto significa que la declaración `MkDir` no se ejecutará.

A continuación, el procedimiento **comprueba** si el archivo seleccionado ya existe en la carpeta de destino. Si es así, el usuario recibirá el mensaje almacenado en la variable **msj1**. Si el archivo no existe en la carpeta de destino y no está abierto en estos momentos, VBA copiará el archivo y notificará al usuario con el mensaje apropiado. Si el archivo está abierto, VBA se encontrará con el error en tiempo de ejecución 70 y ejecutará las instrucciones correspondientes en la sección del procedimiento "**GestorErrores**".

3. Ejecuta el procedimiento varias veces seleccionando archivos de diferentes carpetas.
4. Intenta copiar el mismo archivo dos veces seguidas.
5. Intenta copiar un archivo abierto.
6. Ejecuta el procedimiento Mis_Archivos, creado anteriormente en el capítulo para escribir en la ventana **Inmediato** el contenido de la carpeta "Documentos Archivados".

Atención

No elimines la carpeta "Documentos Archivados" ni los documentos que has copiado en ella. Borraremos tanto la carpeta como los archivos en la siguiente sección usando un procedimiento nuevo.

1.10 Eliminar archivos

Anteriormente se mencionó que no es posible eliminar una carpeta si todavía contiene archivos. Para eliminar los archivos de cualquier carpeta, utilizamos la siguiente declaración:

```
Kill ruta
```

El argumento ruta especifica los nombres de uno o más archivos que vamos a eliminar. De forma opcional, esta ruta puede contener el nombre de la unidad y la carpeta donde se encuentra el archivo. Para permitir la eliminación rápida de archivos podemos utilizar los caracteres comodín (* ó ?) en el argumento de la ruta.

No es posible eliminar un archivo que está abierto. Si has realizado los ejercicios de la sección anterior, en estos momentos tienes la carpeta C:\Documentos Archivados con varios archivos. Escribamos un procedimiento para eliminar la carpeta y los archivos que contiene.

1. Inserta un nuevo módulo en el proyecto GestionArchivos_VBA y llámalo **MetodoKill**.
2. Introduce en él el siguiente procedimiento:

```
Sub Eliminacion()
    Dim Carpeta As String
    Dim miArchivo As String

    ' Asigna el nombre de la carpeta a la variable Carpeta
    ' Atención a la barra invertida final
    Carpeta = "C:\Documentos Archivados\"
    miArchivo = Dir(Carpeta, vbNormal)
```

```
    Do While miArchivo <> ""
        Kill Carpeta & miArchivo
        miArchivo = Dir
    Loop
    RmDir Carpeta
End Sub
```

3. Ejecuta el procedimiento. Cuando termine, comprueba desde el **Explorador de Windows** que la carpeta "Documentos Archivados" ha sido eliminada.

2 Resumen

En este capítulo has aprendido y probado algunas funciones y métodos de VBA que te permiten trabajar con el sistema de archivos. Has descubierto cómo gestionar archivos y carpetas utilizando las funciones integradas:

- `CurDir` para obtener la carpeta actual.
- `GetAttr` y `SetAttr` para comprobar y cambiar los atributos de los archivos.
- `MkDir`, `FileCopy` y `RmDir` para crear, copiar y borrar carpetas respectivamente.
- `Kill` para eliminar archivos.

En el próximo capítulo veremos Windows Script Host (WSH), una útil herramienta ActiveX que nos permitirá controlar y recuperar información del sistema operativo Windows.

Capítulo 11
Gestión de archivos y carpetas con Windows Script Host (WSH)

1 Introducción a WSH

Existe un tesoro escondido en tu ordenador. Se llama **Windows Script Host** (WSH) y te permite crear pequeños programas que controlan el sistema operativo Windows y sus aplicaciones. También permite obtener información de Windows. WSH es un control ActiveX que se encuentra en el archivo Wshom.ocx (ver la Imagen 1.1). Este archivo se puede utilizar para crear scripts que realicen operaciones simples o complejas que antes solo se podían realizar escribiendo archivos .bat. Windows Script Host es un lenguaje de script.

Un script es un conjunto de comandos o instrucciones que se pueden ejecutar automáticamente. Pueden ser creados y ejecutados directamente desde la línea de comandos usando el Command Script Host (Cscript.exe) o desde Windows, usando el Windows Script Host (Wscript.exe). En las siguientes secciones de este capítulo aprenderás cómo funciona el WSH junto con VBA.

Algunas acciones que podrás realizar con WSH son:

- Trabajar y gestionar unidades, carpetas y archivos de Windows usando la función `FileSystemObject`.
- Obtener información sobre Windows, acceder al registro de Windows leer y establecer variables de entorno y obtener el nombre de usuario, dominio y equipo.
- Abrir otras aplicaciones.
- Mostrar los cuadros de diálogo y recoger los datos introducidos por el usuario.
- Crear y gestionar accesos directos en el escritorio de Windows.

WSH tiene su propia jerarquía de objetos. Usando la función `CreateObject`, puedes hacer referencia a los objetos de WSH desde el procedimiento VBA. Antes de comenzar a crear procedimientos propios, veamos algunos de los objetos que podrás controlar:

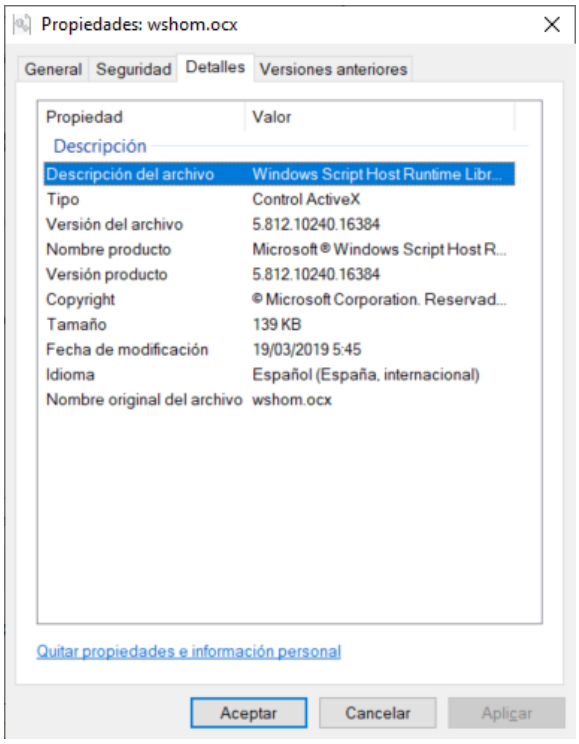


Imagen 1.1 Para comprobar la versión del archivo WSH en tu equipo, busca el archivo en tu directorio de Windows y haz clic con el botón derecho del ratón para acceder a la ventana de propiedades.

1. Crea un nuevo libro de Excel y guárdalo en la carpeta C:\Archivos Manual VBA con el nombre “Capítulo 11 – Gestión con WSH.xlsm”.
2. Activa la ventana del editor de VBA y haz clic en el menú **Herramientas – Referencias**. Haz clic en la casilla de verificación de **Microsoft Scripting Runtime**, como se muestra en la Imagen 1.2 y cierra el cuadro de diálogo.

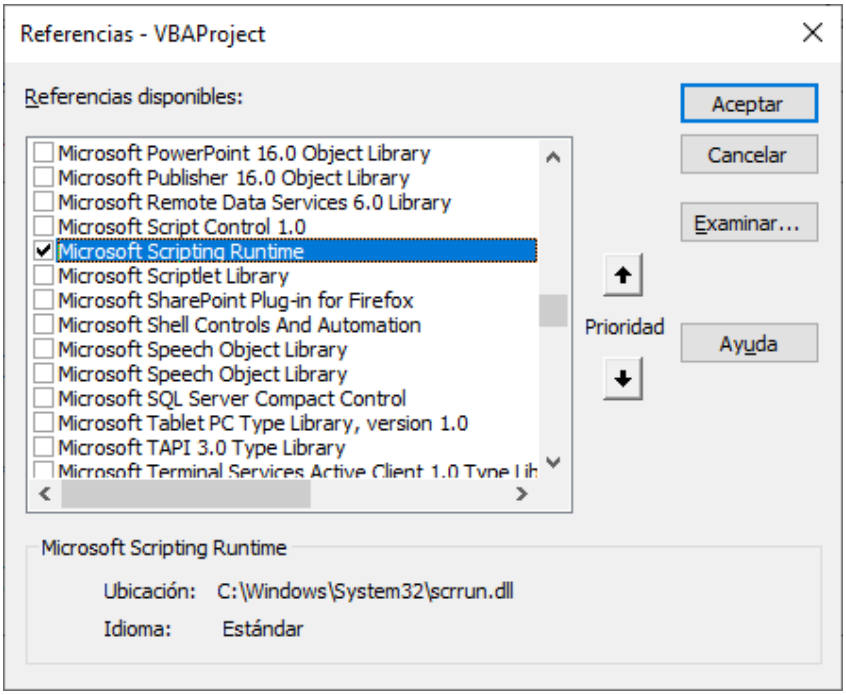


Imagen 1.2 Creando una referencia a Microsoft Scripting Runtime.

3. Presiona **F2** para abrir el **Examinador de objetos**.
4. En el desplegable <Todas >, selecciona **Scripting**. Se mostrará la lista de objetos que forman parte de la biblioteca de WSH, como se muestra en la Imagen 1.3.

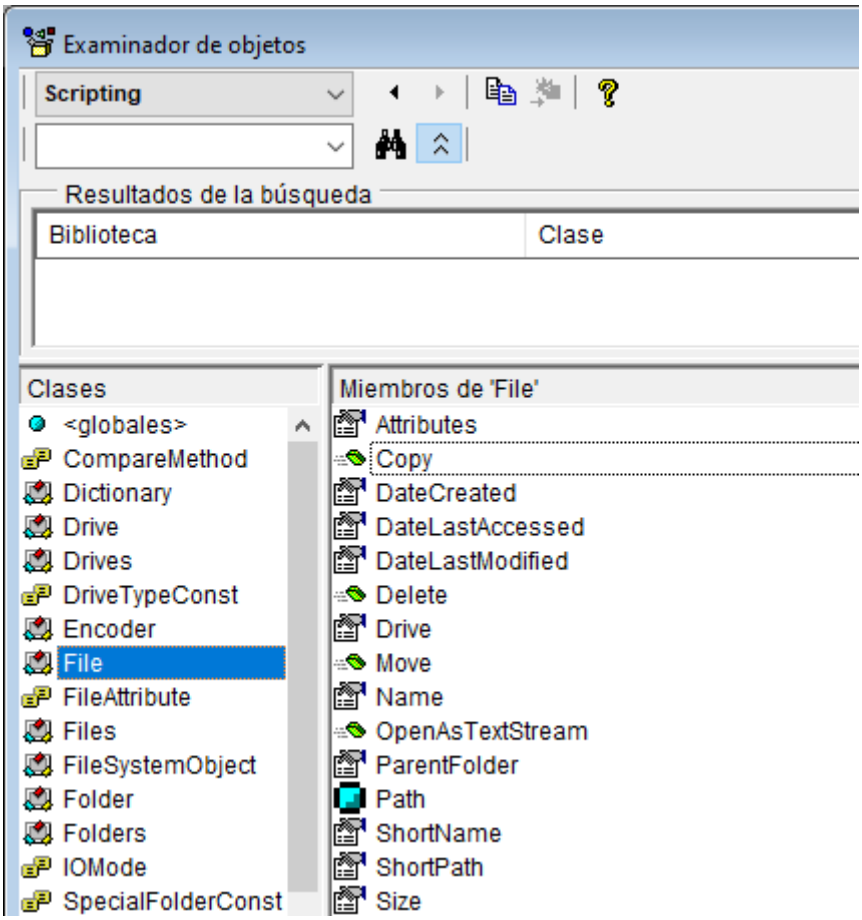


Imagen 1.3 Después de activar la referencia a Microsoft Scripting Runtime, el Examinador de objetos muestra muchos objetos que permiten trabajar con unidades, carpetas, archivos y sus contenidos.

WSH te permite obtener respuestas de forma rápida a preguntas como: "¿en qué disco puedo localizar un archivo en concreto?" (método `GetDrive`), "¿cuál es la extensión de un nombre de archivo?" (método `GetExtensionName`), "¿cuándo se modificó esto por última vez?" (propiedad `DateLastModified`), o "¿existe esta carpeta o archivo en un disco determinado?" (métodos `FolderExists` y `FileExists`).

5. Cierra el Examinador de objetos.

2 Buscar información sobre archivos con Windows Script Host

WSH tiene un objeto llamado `FileSystemObject`. Este objeto tiene varios métodos para trabajar con el sistema de archivos. Veamos cómo se puede obtener algo de información sobre un archivo específico.

1. En la ventana del editor de VBA, activa la ventana **Propiedades** y cambia el nombre del proyecto por `GestionArchivos_WSH`.
2. Inserta un módulo nuevo y llámalo **WSH**.

3. En la ventana **Código** del módulo introduce el siguiente procedimiento (es posible que tengas que cambiar la ruta de la carpeta de Windows para que se ejecute en tu ordenador):

```
Sub InfoArchivo()
    Dim objFs As Object
    Dim objArchivo As Object
    Dim strMsj As String

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objArchivo = objFs.GetFile("C:\WINDOWS\System.ini")
    strMsj = "Nombre del archivo: " & _
    objArchivo.Name & vbCrLf
    strMsj = strMsj & "Unidad: " & _
    objArchivo.Drive & vbCrLf
    strMsj = strMsj & "Fecha de creación: " & _
    objArchivo.DateCreated & vbCrLf
    strMsj = strMsj & "Última modificación: " & _
    objArchivo.DateLastModified & vbCrLf
    MsgBox strMsj, , "Información del archivo"
End Sub
```

Este procedimiento utiliza la función `CreateObject` para crear un objeto ActiveX (`FileSystemObject`) que forma parte de la biblioteca de Windows Script Host. Este objeto proporciona acceso al sistema de archivos de un ordenador.

```
Dim ObjFs As Object
Set objFs = CreateObject("Scripting.FileSystemObject")
```

Este código declara una variable de objeto llamada `objFs`. A continuación, utiliza la función `CreateObject` para crear un objeto ActiveX y asigna el objeto a una variable de objeto. La declaración

```
Set objArchivo = objFs.GetFile("C:\WINDOWS\System.ini")
```

crea y devuelve una referencia al objeto `File` para el archivo System.ini en la carpeta C:\WINDOWS y lo asigna a la variable `objArchivo`. El objeto `File` tiene muchas propiedades que puedes leer. Por ejemplo, la declaración `ObjArchivo.Name` devuelve el nombre del archivo. La declaración `objArchivo.Drive` devuelve el nombre de la unidad donde se encuentra el archivo. Las instrucciones `objArchivo.DateCreated` y `objArchivo.DateLastModified` devuelven la fecha en que se creó el archivo y cuándo se modificó por última vez. Este procedimiento puede modificarse fácilmente para que también devuelva el tipo de archivo, sus atributos y el nombre de la carpeta principal.

Intenta modificar este procedimiento por ti mismo añadiendo las siguientes instrucciones al código:

```
ObjArchivo.Type
ObjArchivo.Attributes
ObjArchivo.ParentFolder
ObjArchivo.Size
```

Comprueba con el **Examinador de objetos** qué otras informaciones puedes obtener sobre el archivo haciendo referencia al objeto `File`.

4. Ejecuta el procedimiento `InfoArchivo`

2.1 Propiedades y métodos de FileSystemObject

Podemos acceder al sistema de archivos del ordenador utilizando el objeto `FileSystemObject`. Este objeto ofrece una gran cantidad de métodos, algunos de los cuales se muestran a continuación:

- `FileExists`. Devuelve `True` si el archivo especificado existe.

```
Sub FileExists()
    Dim objFs As Object
    Dim strArchivo As String

    Set objFs = CreateObject("Scripting.FileSystemObject")
    strArchivo = InputBox("Introduce el nombre completo del
archivo: ")
    If objFs.FileExists(strArchivo) Then
        MsgBox strArchivo & " se ha encontrado."
    Else
        MsgBox "El archivo no existe."
    End If
End Sub
```

- `GetFile`. Devuelve un objeto `File`.
- `GetFileName`. Devuelve el nombre del archivo junto con su ruta.
- `CopyFile`. Copia un archivo:

```
Sub CopyFile()
    Dim objFs As Object
    Dim strArchivo As String
    Dim strNuevoArchivo As String

    strArchivo = "C:\Hola.doc"
    strNuevoArchivo = "C:\Archivos Manual VBA\Hola.doc"
    Set objFs = CreateObject("Scripting.FileSystemObject")
    objFs.CopyFile strArchivo, strNuevoArchivo
    MsgBox "Se ha creado una copia del archivo especificado."
    Set objFs = Nothing
```

```
End Sub
```

- `MoveFile`. Mueve un archivo de una ubicación a otra.
- `DeleteFile`. Elimina un archivo:

```
Sub DeleteFile()
    ' Este procedimiento requiere que establezcas
    ' referencia a Microsoft Scripting Runtime
    Dim objFs As FileSystemObject

    Set objFs = New FileSystemObject
    objFs.DeleteFile "C:\Archivos Manual VBA\Hola.doc"
    MsgBox "El archivo especificado se ha eliminado."
End Sub
```

- `DriveExists`. Devuelve `True` si existe la unidad especificada:

```
Function DriveExists(disco)
    Dim objFs As Object
    Dim strMsj As String
    Set objFs = CreateObject("Scripting.FileSystemObject")
    If objFs.DriveExists(disco) Then
    strMsj = "La unidad " & UCase(disco) & " existe."
    Else
    strMsj = UCase(disco) & " No se ha encontrado."
    End If
    DriveExists = strMsj
    ' Ejecuta esta función desde la hoja de cálculo
    ' introduciendo la siguiente fórmula: =DriveExists("E:\")
End Function
```

- `GetDrive`. Devuelve un objeto Drive.

```
Sub GetDrive()
    Dim objFs As Object
    Dim objDisco As Object
    Dim Mensaje As String
    Dim NombreUnidad As String

    NombreUnidad = InputBox("Introduce la letra de una Unidad de
disco:", _
    "Nombre de Unidad", "C:\")
    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objDisco =
objFs.GetDrive(objFs.GetDriveName(NombreUnidad))
    Mensaje = "Unidad: " & UCase(NombreUnidad) & vbCrLf
```

```
    Mensaje = Mensaje & "Letra de Unidad: " & _
    UCase(objDisco.DriveLetter) & vbCrLf
    Mensaje = Mensaje & "Tipo de Unidad: " & objDisco.DriveType
& vbCrLf
    Mensaje = Mensaje & "Sistema de archivos: " & _
    objDisco.FileSystem & vbCrLf
    Mensaje = Mensaje & "Núm. de serie: " & _
    objDisco.SerialNumber & vbCrLf
    Mensaje = Mensaje & "Tamaño total en bytes: " & _
    FormatNumber(objDisco.TotalSize / 1024, 0) & " Kb" & vbCrLf
    Mensaje = Mensaje & "Espacio disponible: " & _
    FormatNumber(objDisco.FreeSpace / 1024, 0) & " Kb" & vbCrLf
    MsgBox Mensaje, vbInformation, "Información de la unidad"
End Sub
```

- `GetDriveName`. Devuelve el nombre de una cadena de texto que contiene el nombre de la unidad o carpeta compartida:

```
Function NombreUnidad(Unidad) As String
    Dim objFs As Object
    Dim strNombreUnidad As String
    Set objFs = CreateObject("Scripting.FileSystemObject")
    strNombreUnidad = objFs.GetDriveName(Unidad)
    NombreUnidad = strNombreUnidad
    ' Ejecuta esta función desde la ventana Inmediato
    ' introduciendo la instrucción ?NombreUnidad("C:\")
End Function
```

- `FolderExists`. Devuelve `True` si ya existe la carpeta especificada:

```
Sub ExisteCarpeta()
    Dim objFs As Object
    Set objFs = CreateObject("Scripting.FileSystemObject")
    MsgBox objFs.FolderExists("C:\Archivos Manual VBA")
End Sub
```

- `GetFolder`. Devuelve un objeto `Folder`:

```
Sub ArchivosEnCarpeta()
    Dim objFs As Object
    Dim objCarpeta As Object
    Dim objArchivo As Object

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objCarpeta = objFs.GetFolder("C:\")
```

```
    Workbooks.Add
    For Each objArchivo In objCarpeta.Files
        With ActiveCell
            .Formula = objArchivo.Name
            .Offset(0, 1).Range("A1").Formula = objArchivo.Type
            .Offset(1, 0).Range("A1").Select
        End With
    Next
    Columns("A:B").AutoFit
End Sub
```

- `GetSpecialFolder`. Devuelve la ruta de las carpetas de instalación del sistema operativo:
    - 0 - Carpeta Windows.
    - 1 – Carpeta System.
    - 2 - Carpeta Temp.

```
Sub CarpetaInstalacion()
    Dim objFs As Object
    Dim strCarpetaWindows As String
    Dim strCarpetaSystem As String
    Dim strCarpetaTemp As String

    Set objFs = CreateObject("Scripting.FileSystemObject")
    strCarpetaWindows = objFs.GetSpecialFolder(0)
    strCarpetaSystem = objFs.GetSpecialFolder(1)
    strCarpetaTemp = objFs.GetSpecialFolder(2)
    MsgBox strCarpetaWindows & vbCrLf _
    & strCarpetaSystem & vbCrLf _
    & strCarpetaTemp, vbInformation + vbOKOnly, _
    "Carpetas especiales"
End Sub
```

- `CreateFolder`. Crea una carpeta:

```
Sub NuevaCarpeta()
    Dim objFs As Object
    Dim objCarpeta As Object

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objCarpeta = objFs.CreateFolder("C:\Carpeta de prueba")
    MsgBox "La carpeta con el nombre " & _
    objCarpeta.Name & " fue creada con éxito."
End Sub
```

- `CopyFolder`. Crea una copia de la carpeta:

```
Sub CopiaCarpeta()
    Dim objFs As FileSystemObject

    Set objFs = New FileSystemObject
    If objFs.FolderExists("C:\Carpeta de prueba") Then
        objFs.CopyFolder "C:\Carpeta de prueba", "C:\Carpeta
final"
    MsgBox "La carpeta ha sido copiada con éxito."
End If
```

- `MoveFolder`. Mueve una carpeta a otra ubicación.
- `DeleteFolder`. Elimina la carpeta especificada:

```
Sub EliminaCarpeta()
    Dim objFs As Object
    Dim objCarpeta As Object

    Set objFs = CreateObject("Scripting.FileSystemObject")
    If objFs.FolderExists("C:\Carpeta de prueba") Then
        objFs.DeleteFolder "C:\Carpeta de prueba"
        MsgBox "La carpeta fue eliminada."
    End If
End Sub
```

- `CreateTextFile`. Crea un archivo de texto (ver procedimiento de ejemplo más adelante en este capítulo).
- `OpenTextFile`. Abre un archivo de texto:

```
Sub LeerArchivoTexto()
    Dim objFs As Object
    Dim objArchivo As Object
    Dim strContenido As String
    Dim strNombreArchivo As String

    strNombreArchivo = "C:\Archivos Manual VBA\Vacaciones.txt"
    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objArchivo = objFs.OpenTextFile(strNombreArchivo)
    Do While Not objArchivo.AtEndOfStream
        strContenido = strContenido & objArchivo.ReadLine _
        & vbCrLf
    Loop
    objArchivo.Close
```

```
        Set objArchivo = Nothing
        ActiveWorkbook.Sheets(1).Select
        Range("A1").Formula = strContenido
        Columns("A:A").Select
        With Selection
        .ColumnWidth = 62.43
        .Rows.AutoFit
        End With
    End Sub
```

El objeto `FileSystemObject` solo tiene una propiedad, `Drives`, la cual devuelve una referencia a la colección de unidades de disco. Usando esta propiedad podemos crear una lista de unidades de disco de un ordenador, como se muestra a continuación:

```
    Sub ListaUnidades()
        Dim objFs As Object
        Dim colUnidades As Object
        Dim strUnidad As String
        Dim Unidad As Variant

        Set objFs = CreateObject("Scripting.FileSystemObject")
        Set colUnidades = objFs.Drives
        For Each Unidad In colUnidades
            strUnidad = "Unidad " & Unidad.DriveLetter & ": "
            Debug.Print strUnidad
        Next
    End Sub
```

2.2 Propiedades del objeto File

Con el objeto `File` se accede a todas las propiedades de un archivo especificado. Las siguiente líneas de código crean una referencia al objeto `File`:

```
    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objFile = objFs.GetFile("C:\Mis documentos\miArchivo.doc")
```

Encontrarás un ejemplo de uso del objeto `file` en el procedimiento `InfoArchivo` anterior. Estas son las propiedades del objeto `file`:

- `Attributes`. Devuelve los atributos del archivo (comparemos esta propiedad con la función `GetAttr` explicada en el Capítulo 10).
- `DateCreated`. Fecha de creación del archivo.
- `DateLastAccess`. Fecha del ultimo acceso al archivo.
- `DateLastModified`. Fecha de la última modificación del archivo.
- `Drive`. Letra de la unidad seguida de dos puntos.
- `Name`. Nombre del archivo.

- **ParentFolder**. Carpeta padre del archivo.
- **Path**. Ruta completa del archivo.
- **Size**. Tamaño del archivo en bytes (comparemos esta propiedad con la función **FileLen** de VBA).
- **Type**. Tipo de archivo. Es el texto que aparece en la columna "Tipo de archivo" del explorador de Windows.

2.3 Propiedades del objeto Folder

El objeto **Folder** proporciona acceso a todas las propiedades de una carpeta especificada. Las siguientes líneas de código crean una referencia al objeto **Folder**:

```
Set objFs = CreateObject("Scripting.FileSystemObject")
Set objFolder = objFs.GetFolder("C:\Mis documentos")
```

El objeto **Folder** tiene las siguiente propiedades:

- **Attibutes**. Atributos de la carpeta.
- **DateCreated**. Fecha de la creación de la carpeta.
- **Drive**. Devuelve la letra de la Unidad donde se encuentra la carpeta.
- **Files**. Devuelve la colección de archivos en la carpeta.

```
Sub CuentaArchivosCarpeta()
    Dim objFs As Object
    Dim strCarpeta As String
    Dim objCarpeta As Object
    Dim objArchivos As Object

    strCarpeta = InputBox("Introduce el nombre de la carpeta:")
    If Not IsFolderEmpty(strCarpeta) Then
        Set objFs = CreateObject("Scripting.FileSystemObject")
        Set objCarpeta = objFs.GetFolder(strCarpeta)
        Set objArchivos = objCarpeta.Files
        MsgBox "Número de archivos en la carpeta " & _
        strCarpeta & " = " & objArchivos.Count
    Else
        MsgBox "La carpeta " & strCarpeta &_
         " no tiene archivos."
    End If
End Sub
```

(El procedimiento anterior hace referencia a la función **CarpetaVacia** que veremos en el siguiente ejemplo.)

- **IsRootFolder**. Devuelve True si la carpeta es la raíz de la Unidad.
- **Name**. Devuelve el nombre de la carpeta.
- **ParentFolder**. Devuelve el nombre de la carpeta padre.

- `Path`. Ruta completa a la carpeta.
- `Size`. Tamaño de la carpeta en bytes.

```
Function CarpetaVacia(miCarpeta)
    Dim objFs As Object
    Dim objCarpeta As Object

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objCarpeta = objFs.GetFolder(miCarpeta)
    CarpetaVacia = (objCarpeta.Size = 0)
End Function
```

- `SubFolders`. Colección de subcarpetas en la carpeta especificada.
- `Type`. Tipo de carpeta, por ejemplo **Carpeta de archivos** o **Papelera de reciclaje**.

2.4 Propiedades del objeto Drive

El objeto `Drive` proporciona acceso a las propiedades de la Unidad especificada en un ordenador o en un servidor. Las siguiente líneas de código crean una referencia al objeto `Drive`.

```
Set objFs = CreateObject("Scripting.FileSystemObject")
Set objDrive = objFs.GetDrive("C:\")
```

El objeto `Drive` tiene las siguiente propiedades:

- `AvailableSpace`. Espacio disponible en bytes.
- `FreeSpace`. Igual que `AvailableSpace`.
- `DriveLetter`. Letra de la Unidad (sin los dos puntos).
- `DriveType`. Tipo de Unidad:
  - `0` – Desconocida.
  - `1` – Extraíble.
  - `2` – Fija.
  - `3` – Unidad de red.
  - `4` – CD-Rom.
  - `5` – Unidad RAM.

```
Sub LetraCDROM()
    Dim objFs As Object
    Dim colUnidades As Object
    Dim Unidad As Object
    Dim 260ontador As Integer
    Const CDROM = 4

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set colUnidades = objFs.Drives
    260ontador = 0
```

```
    For Each Unidad In colUnidades
        If Unidad.DriveType = CDROM Then
            261ontador = 261ontador + 1
            Debug.Print "Unidad de CD-ROM: " & Drive.DriveLetter
        End If
    Next
    MsgBox "Hay " & contador & " Unidades de CD-ROM."
End Sub
```

- `FileSystem`. Tipo de sistema de archivos, por ejemplo FAT, NTFS o CDFS.
- `IsReady`. Devuelve True si el dispositivo (por ejemplo, un CDROM), está listo para usarse.

```
Function CDROMListo(strLetraUnidad)
    Dim objFs As Object
    Dim objUnidad As Object

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objUnidad = objFs.GetDrive(strLetraUnidad)
    IsCDROMReady = (objUnidad.DriveType = 4) And _
    objUnidad.IsReady = True
    ' Ejecuta esta función en la ventana Inmediato
    ' introduciendo: ?CDROMListo("D:")
End Function
```

- `Path`. Ruta de la carpeta raíz.
- `SerialNumber`. Número de serie de la Unidad.
- `TotalSize`. Tamaño total de la Unidad expresado en bytes.

3 Crear un archivo de texto usando WSH

Windows Script Host ofrece tres métodos para crear archivos de texto: `CreateTextFile`, `OpenTextFile` y `OpenAsTextStream`. La sintaxis de cada uno de ellos se muestra a continuación.

3.1 Método 1

```
CreateTextFile object.CreateTextFile(filename[, overwrite[,
unicode]])
```

- `Object`. Es el nombre del objeto `FileSystemObject` o del objeto `Folder`.
- `Filename`. Es una cadena de texto que especifica el archivo a crear.
- `Overwrite` (opcional). Es un valor booleano que indica si se puede sobrescribir un archivo existente. El valor es True si el archivo puede ser sobrescrito y False en caso contrario. Si se omite este argumento, los archivos existentes no se sobrescriben.

- **Unicode** (opcional). Es un valor booleano que indica si el archivo es creado como archivo Unicode o ASCII. El valor es True si el archivo se crea como Unicode y False si se crea como ASCII. Si se omite, se supone que el archivo es ASCII.

```
Sub CrearArchivoMetodo1()
    Dim objFs As Object
    Dim objArchivo As Object
    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objArchivo = objFs.CreateTextFile("C:\Teléfonos.txt", True)
    objArchivo.WriteLine ("Juan Pérez: +34 333 33 33 33")
    objArchivo.WriteBlankLines (2)
    objArchivo.WriteLine ("Eduardo Valentino: +54 202-988-2331")
    objArchivo.Close
End Sub
```

El procedimiento anterior crea un archivo de texto para almacenar los nombres y teléfonos de dos personas. Como el argumento **overwrite** se ha establecido en **True**, el archivo C:\Teléfonos.txt se sobrescribirá si ya existe en la carpeta especificada.

3.2 Método 2

```
OpenTextFile object.OpenTextFile(filename[, iomode[, create[,
format]]])
```

- **Object**. El nombre del objeto **FileSystemObject**.
- **Filename**. Es la cadena de texto que identifica qué archivo abrir.
- **Iomode**. Indica el modo de entrada/salida. El argumento **Iomode** puede ser una de las siguientes constantes:
    - **ForReading** (1)
    - **ForWriting** (2)
    - **ForAppending** (8)
- **Create** (opcional). Es un valor booleano que indica si se puede crear un nuevo archivo cuando el argumento **filename** no existe. El valor es True, si se crea un nuevo archivo y False si no se crea. El valor predeterminado es False.
- **Format**. Uno de los tres valores **Tristate** usados para indicar el formato del archivo abierto.
    - **TristateTrue**. Abre el archivo como ASCII.
    - **TristateFalse**. Abre el archivo como Unicode.
    - **TristateUseDefault**. Abre el archivo en el formato predeterminado del sistema.

```
Sub CrearArchivoMetodo2()
    Dim objFs As Object
    Dim objArchivo As Object
    Const Escritura = 2

    Set objFs = CreateObject("Scripting.FileSystemObject")
    Set objArchivo = objFs.OpenTextFile("C:\Lista de la compra.txt", _
```

```
    Escritura, True)
    objArchivo.WriteLine ("Pan")
    objArchivo.WriteLine ("Leche")
    objArchivo.WriteLine ("Manzanas")
    objArchivo.Close
End Sub
```

3.3 Método 3

```
OpenAsTextStream object.OpenAsTextStream([iomode, [format]])
```

- `Object`. El nombre del objeto `FileSystemObject`.
- `Iomode`. Indica el modo de entrada/salida. El argumento `Iomode` puede ser una de las siguientes constantes:
  - `ForReading` (1)
  - `ForWriting` (2)
  - `ForAppending` (8)
- `Format`. Uno de los tres valores `Tristate` usados para indicar el formato del archivo abierto.
  - `TristateTrue`. Abre el archivo como ASCII.
  - `TristateFalse`. Abre el archivo como Unicode.
  - `TristateUseDefault`. Abre el archivo en el formato predeterminado del sistema.

```
Sub CrearArchivo3()
    Dim objFs As Object
    Dim objArchivo As Object
    Dim objTexto As Object
    Const Escritura = 2
    Const Lectura = 1

    Set objFs = CreateObject("Scripting.FileSystemObject")
    objFs.CreateTextFile "Nuevo.txt"
    Set objArchivo = objFs.GetFile("Nuevo.txt")
    Set objTexto = objArchivo.OpenAsTextStream(Escritura, _
    TristateUseDefault)
    objTexto.Write "Invitación a la boda"
    objTexto.Close
    Set objTexto = objArchivo.OpenAsTextStream(Lectura, _
    TristateUseDefault)
    MsgBox objTexto.ReadLine
    objTexto.Close
End Sub
```

4 Otras operaciones con WSH

Con WSH podemos manipular cualquier objeto de nuestros ordenadores. Además de acceder al sistema de archivos a través de **FileSystemObject**, WSH nos permite realizar tareas como manejar objetos WSH y Activex, mapear impresoras y unidades remotas, manipular el registro de Windows, crear atajos a Windows e Internet y acceder a Azure Active Directory.

El modelo de objetos de WSH está compuesto por tres objetos principales:

WScript

WshShell

WshNetwork

Las siguientes secciones muestran cómo podemos aprovechar el objeto **WshShell** escribiendo procedimientos que inicien otras aplicaciones. También crearemos algunos atajos.

4.1 Ejecutar otras aplicaciones

Supongamos que queremos iniciar el Bloc de notas de Windows desde un procedimiento VBA. El siguiente procedimiento muestra lo fácil que es ejecutar una aplicación usando el objeto **WshShell** que forma parte del WSH. Si prefieres abrir la aplicación Calculadora de Windows, solo hay que sustituir el nombre de la aplicación del Bloc de notas por Calc.

1. Inserta un módulo nuevo en el proyecto GestionArchivos_WSH y llámalo **WSH_Extra**.
2. Introduce el siguiente procedimiento en el módulo:

```
Sub AbrirBlocNotas()
    Dim WshShell As Object
    Set WshShell = CreateObject("WScript.Shell")
    WshShell.Run "Notepad"
    Set WshShell = Nothing
End Sub
```

El procedimiento anterior comienza declarando un objeto **WshShell**:

```
Dim WshShell As Object
Set WshShell = CreateObject("WScript.Shell")
```

La siguiente línea utiliza el método **Run** para ejecutar la aplicación solicitada:

```
WshShell.Run "Notepad"
```

Utilizando el mismo concepto, es fácil ejecutar aplicaciones de Windows como la Calculadora o el Explorador.

```
WshShell.Run "Calc"
WshShell.Run "Explorer"
```

La última línea del procedimiento elimina el objeto **WshShell** porque no lo vamos a necesitar más.

```
Set WshShell = Nothing
```

3. Ejecuta el procedimiento **AbrirBlocNotas**

En lugar de lanzar una aplicación vacía podemos iniciar la aplicación con un documento específico, como se muestra en el siguiente procedimiento:

```
Sub AbrirArchivoenBloc()
    Dim WshShell As Object
    Set WshShell = CreateObject("WScript.Shell")
    WshShell.Run "Notepad C:\Archivos Manual VBA\Vacaciones.txt"
    Set WshShell = Nothing
End Sub
```

Si no se puede encontrar el archivo especificado, VBA preguntará si deseas crear el archivo. Si la ruta contiene espacios debes escribirla entre comillas o el método `Run` provocará un error de ejecución. Por ejemplo, para abrir el archivo "C:\Mis documentos\Mi archivo de texto.txt" desde el Bloc de notas, utiliza la siguiente declaración:

```
WshShell.Run "Notepad " _
& """C:\Mis documentos\Mi archivo de texto.txt"""
```

O utiliza el equivalente ANSI para las comillas dobles, como se muestra a continuación:

```
WshShell.Run "Notepad " & Chr(34) & _
"C:\Mis documentos\Mi archivo de texto.txt" & Chr(34)
```

La declaración anterior utiliza la función `Chr` de VBA para convertir un valor ANSI a su equivalente en carácter. 34 es el valor ANSI para las comillas dobles.

Para abrir una página web, introduce la dirección de la página web en el método `Run`, como en el siguiente ejemplo:

```
WshShell.Run ("https://ayudaexcel.com")
```

La siguiente declaración invoca al Panel de control:

```
WshShell.Run "Control.exe"
```

El Panel de control tiene una gran cantidad de opciones. La siguiente declaración abre la opción **Página de propiedades del sistema** con la pestaña de **Hardware** seleccionada:

```
WshShell.Run "Control.exe Sysdm.cpl, ,2"
```

Los parámetros tras el nombre de la Página de propiedades del Panel de control (identificados por la extensión .cpl) especifican qué página se selecciona. Para seleccionar la pestaña **General**, reemplaza el 2 por un cero (0). Observa que el método `Run` tiene dos argumentos opcionales que te permiten especificar el estilo de la ventana y si el sistema debe esperar hasta que el proceso ejecutado se completa. Por ejemplo, puedes lanzar el Bloc de notas con la ventana minimizada, como se muestra a continuación:

```
WshShell.Run "Notepad", vbMinimizedFocus
```

Como no se especifica el segundo argumento opcional, el método **Run** ejecuta el comando Notepad.exe e inmediatamente finaliza el proceso. Si el segundo parámetro se establece en **True**, el método **Run** creará un nuevo proceso, ejecutará el comando y esperará hasta que el proceso termine:

```
WshShell.Run "Notepad", vbMinimizedFocus, True
```

Si ejecutas la declaración anterior con el segundo parámetro establecido en **True**, no podrás trabajar con Excel hasta que cierres el Bloc de notas.

4.2 Obtener información sobre Windows

Windows almacena varios tipos de información en variables de entorno. Podemos usar la propiedad **Environment** del objeto **WshShell** para tener acceso a estas variables. Dependiendo de la versión del sistema operativo que estemos utilizando, las variables de entorno se agrupan en las categorías: **Sistema**, **Usuario**, **Volátil** y **Proceso**. Podemos utilizar el nombre de la categoría como índice de la propiedad **Environment**. El siguiente procedimiento muestra cómo recuperar los valores de varias variables de entorno de la categoría **Proceso**:

```
Sub VariableEntorno()
    Dim WshShell As Object
    Dim objEnv As Object
    Set WshShell = CreateObject("WScript.Shell")
    Set objEnv = WshShell.Environment("Process")
    Debug.Print "Ruta=" & objEnv("PATH")
    Debug.Print "Unidad principal=" & objEnv("SYSTEMDRIVE")
    Debug.Print "Raíz del sistema=" & objEnv("SYSTEMROOT")
    Debug.Print "Carpeta de Windows=" & objEnv("Windir")
    Debug.Print "Sistema operativo=" & objEnv("OS")
    Set WshShell = Nothing
End Sub
```

4.3 Obtener información sobre el usuario, dominio u ordenador

Podemos utilizar las propiedades del objeto **WshNetwork** de WSH para recuperar el nombre del usuario, el nombre de dominio o el nombre de la carpeta del ordenador, como se muestra en el siguiente procedimiento:

```
Sub UsuarioDominioOrdenador()
    Dim WshNetwork As Object
    Dim misDatos As String
    Set WshNetwork = CreateObject("WScript.Network")
    misDatos = misDatos & "Nombre del equipo: " _
    & WshNetwork.ComputerName & vbCrLf
    misDatos = misDatos & "Dominio: " _
    & WshNetwork.UserDomain & vbCrLf
```

```
    misDatos = misDatos & "Usuario: " _
    & WshNetwork.UserName & vbCrLf
    MsgBox misDatos
End Sub
```

4.4 Crear accesos directos

Si comenzamos a distribuir nuestras macros o aplicaciones de VBA, los usuarios seguramente nos pedirán que coloquemos un acceso directo a nuestro trabajo en su escritorio. VBA no tiene una forma de crear atajos de Windows. Por suerte, ahora sabemos cómo trabajar con WSH, y podemos usar su objeto `Shell` para crear accesos directos a aplicaciones o sitios web sin ninguna intervención del usuario. El objeto `WshShell` cuenta con el método `CreateShortcut`, que podemos utilizar de la siguiente manera:

```
Set miAcceso = WshShell.CreateShortcut(ruta)
```

`Ruta` es una cadena de texto que indica la ruta al acceso directo del archivo. Todos los accesos directos tienen la extensión .lnk, que debe ser incluida en dicha ruta. El método `CreatShortcut` devuelve un objeto de acceso directo que cuenta con varias propiedades y un método:

- `TargetPath`. Es la ruta al archivo ejecutable.

  ```
  WshShell.TargetPath = ActiveWorkbook.FullName
  ```

- `WindowStyle`. Identifica el estilo de ventana utilizado por el acceso directo.
  - `1` – Ventana normal.
  - `3` – Ventana maximizada.
  - `7` – Ventana minimizada.

  ```
  WshShell.WindowStyle = 1
  ```

- `Hotkey`. Es el atajo de teclado que se utilizará para ejecutar el acceso directo (por ejemplo Ctrl + G, Ctrl + Mayús + A, Alt + F11.

  ```
  WshShell.Hotkey = "Ctrl+Alt+W"
  ```

- `IconLocation`. Es la ubicación donde vamos a colocar el atajo. Como los archivos de icono suelen contener más de un icono, debemos proporcionar la ruta de acceso al icono seguido del número de índice del icono deseado en ese archivo. Si no se especifica, Windows utiliza el icono por defecto para el archivo.

  ```
  WshShell.IconLocation = "notepad.exe, 0"
  ```

- `Description`. Contiene la cadena de texto con el valor de la descripción.

  ```
  WshShell.Description = "Formación y consultoría para Excelers"
  ```

- `Save`. Es el único método del objeto `Shortcut`. Tras utilizar el método `CreateShortcut` para crear un objeto `Shortcut` y establecer las propiedades del objeto, se debe utilizar el método `Save` para guardar el objeto `Shortcut` en el disco.

En el siguiente ejercicio crearemos un objeto `WshShell` y usaremos el método `CreateShortcut` para crear dos atajos: un atajo de Windows al archivo activo del libro de

trabajo de Microsoft Excel y un atajo de Internet a nuestro sitio web Ayuda Excel. Ambos atajos se colocan en el escritorio del usuario.

1. En el módulo **WSH_Extra** introduce el siguiente procedimiento:

```
Sub AtajoAcceso()
' Esta macro crea dos accesos directos
    Dim WshShell As Object
    Dim objAcceso As Object
    Dim strURL As String
    strURL = "https://ayudaexcel.com"
    Set WshShell = CreateObject("WScript.Shell")
    ' Crea un atajo a la web
    Set objAcceso = WshShell.CreateShortcut(WshShell. _
    SpecialFolders("Desktop") & _
    "\Formación y recursos para Excelers.url")
    With objAcceso
        .TargetPath = strURL
        .Save
    End With
    ' create a file shortcut
    ' you cannot create a shortcut to unsaved workbook file
    Set objAcceso = WshShell.CreateShortcut _
    (WshShell.SpecialFolders("Desktop") & "\" & _
    ActiveWorkbook.Name & ".lnk")
    With objAcceso
        .TargetPath = ActiveWorkbook.FullName
        .Description = "Aprende con Ayuda Excel"
        .WindowStyle = 7
        .Save
    End With
    Set objAcceso = Nothing
    Set WshShell = Nothing
End Sub
```

El procedimiento utiliza la propiedad **SpecialFolders** del objeto **WhsShell** para devolver la ruta al escritorio de Windows.

2. Ejecuta el procedimiento.
3. Haz que se muestre el escritorio y haz clic en los dos accesos directos que se han creado en él.

La propiedad SpecialFolders

Podemos encontrar la ubicación de una carpeta especial en nuestros equipos utilizando la propiedad **`SpecialFolders`**. Están disponibles las siguientes carpetas:

- AllUsersDesktop
- AllUsersPrograms
- AllUsersStartMenu
- AllUsersStartup
- Desktop
- Favorites
- Fonts
- MyDocuments
- NetHood
- PrintHood
- Programs
- Recent
- SendTo
- StartMenu
- Startup
- Templates

Si la carpeta que requerimos no es una de estas, la propiedad **`SpecialFolders`** devuelve una cadena de texto vacía.

4.5 Crear una lista de accesos directos

El siguiente procedimiento muestra en la ventana **Inmediato** la lista de todos los archivos de acceso directo que se encuentran en el escritorio. Podemos modificar fácilmente este procedimiento para mostrar otras carpetas especiales (ver cuadro anterior) o enumerar todas las carpetas si eliminamos la declaración condicional. Observemos que el procedimiento utiliza la función **`InStrRev`** para comprobar si el archivo es un acceso directo. Esta función tiene la misma sintaxis que **`InStr`**, excepto que devuelve la posición de una ocurrencia de una cadena de texto dentro de otra, pero desde el final de la cadena.

```
Sub ListadoAccesos()
    Dim objFs As Object
    Dim objCarpeta As Object
    Dim wshShell As Object
    Dim strEnlaces As String
    Dim s As Variant
    Dim f As Variant

    Set wshShell = CreateObject("WScript.Shell")
```

```
    Set objFs = CreateObject("Scripting.FileSystemObject")
    strEnlaces = ""
    For Each s In wshShell.SpecialFolders
        Set objCarpeta = objFs.GetFolder(s)
        strEnlaces = strEnlaces & objCarpeta.Name _
        & " Shortcuts:" & vbCrLf
        If objCarpeta.Name = "Desktop" Then
            For Each f In objCarpeta.Files
                If InStrRev(UCase(f), ".LNK") Then
                    strEnlaces = strEnlaces & f.Name & vbCrLf
                End If
            Next
        End If
        Exit For
    Next
    Debug.Print strEnlaces
End Sub
```

5 Resumen

En este capítulo has aprendido a utilizar Windows Script Host (WSH) para acceder al objeto **`FileSystemObject`** y realizar otras operaciones, como abrir aplicaciones y crear accesos directos de Windows con el objeto **`WshShell`**.

En el siguiente capítulo, aprenderás a utilizar VBA para trabajar con tres tipos de archivo: secuenciales, de acceso aleatorio y binarios.

Capítulo 12
Acceso a archivos de texto

Además de abrir archivos dentro de una aplicación en particular, un procedimiento VBA es capaz de abrir otros tipos de archivo y trabajar con sus contenidos. Este capítulo te introducirá en el proceso conocido como E/S (Entrada/Salida) o I/O (Input/Output en inglés).

1 Tipos de acceso a los archivos

Un ordenador puede utilizar tres tipos de archivo:

- **De acceso secuencial**. Son archivos en los que los datos se recuperan en el mismo orden a como se almacenan. Por ejemplo, los archivos en formato CSV (separados por comas), TXT (texto separado por tabulaciones) o PRN (texto separado por espacios). El acceso secuencial se utiliza a menudo para escribir textos como registros de errores, configuraciones e informes. Los archivos de acceso secuencial tienen los siguientes modos de acceso: **Input**, **Output** y **Append**. Este acceso especifica la forma de trabajar con un archivo después de que se haya abierto.
- **De acceso aleatorio**. Son archivos de texto donde los datos se almacenan en registros de igual longitud y los campos se encuentran separados por comas. Este tipo de archivos solo tienen un modo: **Random**.
- **Binarios**: Son archivos que contienen información de cualquier tipo codificada en binario, por ejemplo, imágenes. A los archivos binarios sólo se puede acceder en modo binario.

2 Los archivos de acceso secuencial

El disco duro de nuestros equipos contiene cientos de archivos secuenciales. Archivos de configuración, registros de errores, archivos HTML y todo tipo de archivos de texto plano. Estos archivos se almacenan en el disco como una secuencia de caracteres. El comienzo de una nueva línea de texto se indica con dos caracteres especiales: el retorno de carro (**\r**) y el salto de línea (**\n**). Cuando se trabaja con archivos secuenciales, se comienza al principio del archivo y se avanza carácter a carácter y línea a línea, hasta encontrar el final del archivo. Los archivos

de acceso secuencial pueden abrirse y manipularse fácilmente con casi cualquier editor de texto.

2.1 Leer datos de archivos secuenciales

Tomemos, por ejemplo, uno de los archivos que ya contienen nuestros ordenadores y leamos su contenido con VBA directamente desde la ventana del editor. Podemos leer cualquier otro archivo que queramos. Para leer los datos de un archivo, lo abrimos con la declaración **Open**. Esta es la sintaxis general de esta declaración, seguida de una explicación de cada argumento:

```
Open pathname For mode[Access access][lock] As [#]filenumber
[Len=reclength]
```

La declaración **Open** tiene tres argumentos obligatorios: **pathname**, **mode** y **filenumber**.

- **Pathname** es el nombre del archivo que queremos abrir. El nombre del archivo puede incluir el nombre de una Unidad y una carpeta.
- **Mode** es una palabra clave que determina cómo se abrirá el archivo. Los archivos secuenciales pueden abrirse en uno de los siguientes modos: **Input**, **Output** o **Append**. Utilizamos **Input** para leer el archivo, **Output** para escribir en el archivo sobrescribiendo cualquier archivo existente y **Append** para escribir en el archivo añadiendo la nueva información al contenido que ya existe.
- La cláusula opcional **Access** puede utilizarse para especificar los permisos del archivo (lectura, escritura o lectura y escritura).
- El argumento opcional **Lock** determina qué operaciones de archivo se permiten para otros procesos. Por ejemplo, si un archivo está abierto en un entorno de red, **Lock** determina cómo pueden acceder a él otras personas. Se pueden utilizar las siguientes opciones: **Shared**, **Lock Read**, **Lock Write** o **Lock Read Write**.
- El argumento **filenumber** es un número del 1 al 511. Este número se utiliza para referirse al archivo en operaciones posteriores. Puedes obtener un número de archivo único utilizando la función **FreeFile** de VBA.
- El último elemento de la declaración **Open**, **reclength**, especifica el tamaño de la memoria intermedia (número total de caracteres) para los archivos secuenciales, o el tamaño del registro para los archivos de acceso aleatorio.

Teniendo en cuenta lo anterior, para abrir el archivo Vacaciones.txt (que ya tenemos en nuestra carpeta) o cualquier otro archivo secuencial para leer sus datos, utilizamos la siguiente instrucción:

```
Open "C:\Archivos Manual VBA\Vacaciones.txt" For Input As #1
```

Si un archivo se abre en modo **Input**, solo podremos acceder a él para leer datos. Después de abrir el archivo secuencial, podemos leer su contenido con las declaraciones **Line Input #** o **Input #** o usando la función **Input**. Cuando se utiliza el acceso secuencial con el método Input, el archivo debe existir previamente.

2.2 Leer un archivo línea a línea

Para leer el contenido de un archivo secuencial línea por línea utilizamos la siguiente declaración:

¿Qué es un archivo secuencial?

Un archivo secuencial es un tipo de archivo en el que la información puede leerse y escribirse desde el principio de este. Esto significa que antes de poder acceder al tercer registro, se debe acceder primero al registro 1 y luego al registro 2.

```
Line Input #filenumber, variableName
```

#filenumber es el número de archivo que se utilizó en el proceso de apertura del archivo en la declaración **Open**.

variableName es una cadena o variable que almacenará la línea que se está leyendo. La instrucción **Line Input #** lee una sola línea en un archivo secuencial abierto y la almacena en una variable. Debemos tener en cuenta que la instrucción **Line Input #** lee un solo carácter a la vez, hasta que se encuentra con un retorno de carro (Chr(13)) o una secuencia de salto de línea (Chr(13) & Chr(10)). Estos caracteres se omiten en el texto recuperado en el proceso de lectura.

El siguiente procedimiento muestra cómo podemos usar las declaraciones **Open** y **Line Input #** para leer el contenido del archivo **Vacaciones.txt** línea a línea. También se aplica el mismo método para leer otros archivos secuenciales.

1. Crea un nuevo archivo y guárdalo en la carpeta C:\Archivos Manual VBA con el nombre "Capítulo 12 – Acceso a archivos txt-csv-prn.xlsm".
2. Accede al editor de VBA presionando **Alt + F11** y modifica el nombre del proyecto del archivo. Llámalo **Acceso_Archivos**.
3. Inserta un módulo nuevo y llámalo **Archivo_Secuencial**.
4. En el nuevo módulo inserta el siguiente procedimiento:

```
Sub Leeme(strNombreArchivo As String)
    Dim rFila As String
    Dim i As Integer

    ' Número de línea-fila
    i = 0
    On Error GoTo Salir
    Open strNombreArchivo For Input As #1
    ' permanece dentro del bucle hasta alcanzar
    ' el final del archivo
    Do While Not EOF(1)
        i = i + 1
        Line Input #1, rFila
        MsgBox "Fila " & i & " en " & strNombreArchivo & _
        Chr(13) & Chr(13) & rFila
```

```
    Loop
    MsgBox "Se han leido " & i & " líneas."
    Close #1
    Exit Sub
Salir:
    MsgBox "File " & strNombreArchivo & " no se ha encontrado."
End Sub
```

El procedimiento abre el archivo de texto especificado en modo Input (lectura) como archivo `# 1` para leer su contenido. Si el archivo especificado no puede abrirse (porque puede que no exista), VBA salta a la etiqueta `Salir` y muestra un cuadro de mensaje.

Si el archivo logra abrirse con éxito, se procede a leer su contenido. El bucle `Do … While` le dice a VBA que ejecute las instrucciones de dentro del bucle hasta que se llegue al final del archivo. El final del archivo está determinado por el resultado de la función `EOF`. La función `EOF` (End Of File) devuelve un valor lógico `True` si el siguiente carácter a leer está más allá del final del archivo. Observa que la función `EOF` requiere un argumento (el número del archivo abierto que quieres comprobar). Este número es el mismo que utilizaste en la declaración `Open`. Utiliza la función `EOF` para asegurarte de que VBA no lee más allá del final del archivo.

La instrucción `Line Input #` almacena el contenido de cada línea en la variable `rFila`. A continuación, aparece un mensaje que muestra el número de línea y su contenido. VBA sale del bucle `Do … While` cuando el resultado de la función `EOF` es True. Antes de que VBA termine el procedimiento, se ejecutan dos declaraciones más. Se muestra un mensaje con el número total de líneas que han sido leídas. Para finalizar, se cierra el archivo.

5. Para ejecutar el procedimiento, abre la ventana **Inmediato** y escribe la siguiente declaración:

```
Leeme "C:\Archivos Manual VBA\Vacaciones.txt"
```

2.3 Leer caracteres de un archivo secuencial

Supongamos que el procedimiento necesita comprobar cuántas comas aparecen en un archivo. En lugar de leer filas completas, podemos utilizar la función `Input` para devolver el número de caracteres especificado. A continuación, la declaración `If` puede utilizarse para comparar el carácter obtenido con el que se está buscando. Antes de escribir un procedimiento que haga esto, repasemos la sintaxis de la función `Input`:

```
Input(numero, [#]numeroarchivo)
```

Se requieren los dos argumentos de la función: `numero` especifica el número de caracteres que se desean leer, y `numeroarchivo` es el mismo número que utilizamos en la declaración `Open` para abrir el archivo. La función `Input` devuelve todos los caracteres que se están leyendo, incluyendo comas, retornos de carro, marcadores de fin de archivo, comillas y espacios en blanco.

1. Introduce el siguiente procedimiento en el módulo **Archivo_Secuencial**:

```
Sub ContarCaracteres(strNombreArchivo As String, buscCarac As
String)
    Dim contador As Integer
    Dim car As String

    contador = 0
    Open strNombreArchivo For Input As #1
    Do While Not EOF(1)
        car = Input(1, #1)
        If car = buscCarac Then
            contador = contador + 1
        End If
    Loop
    If contador <> 0 Then
            MsgBox "Caracteres (" & buscCarac & _
            ") encontrados: " & contador
        Else
            MsgBox "El carácter (" & buscCarac & _
            ") no se ha encontrado."
    End If
    Close #1
End Sub
```

2. Para ejecutar el procedimiento, abre la ventana **Inmediato** y escribe la siguiente declaración. Presiona **Intro** para ejecutar:

```
ContarCaracteres "C:\Archivos Manual VBA\Vacaciones.txt","."
```

Como no hay comas en el archivo Vacaciones.txt deberías recibir un mensaje advirtiendo de que el carácter especificado no fue encontrado.

3. Ejecuta de nuevo el procedimiento tras reemplazar el carácter punto (.) por cualquier otro que desees encontrar. La función **Input** te permite devolver cualquier carácter del archivo. Si utilizas la función **LOF** (Lenght Of File) de VBA como primer argumento de la función **Input**, podrás leer rápidamente el contenido del archivo secuencial sin tener que recorrer todo el archivo. La función **LOF** devuelve el número de bytes de un archivo. Cada byte corresponde a un carácter de un archivo de texto. El siguiente procedimiento, **LeerTodo**, muestra cómo leer el contenido de un archivo secuencial en la ventana **Inmediato**:

```
Sub LeerTodo(strNombreArchivo As String)
    Dim todo As String

    Open strNombreArchivo For Input As #1
```

```
    todo = Input(LOF(1), #1)
    Debug.Print all
    Close #1
End Sub
```

Para ejecutar el procedimiento anterior, abre la ventana **Inmediato**, escribe la siguiente declaración y presiona **Intro**:

```
LeerTodo "C:\Archivos Manual VBA\Vacaciones.txt"
```

En lugar de imprimir el contenido del archivo en la ventana **Inmediato**, puedes leerlo colocado en una hoja de cálculo como el de la Imagen 2.1. Invierte unos minutos para escribir este procedimiento:

1. Introduce el procedimiento `EscribirCuadroTexto` en el módulo `Archivo_Secuencial`:

```
Sub EscribirCuadroTexto(strNombreArchivo As String)
    Dim sh As Worksheet

    Set sh = ActiveSheet
    On Error GoTo CierraArchivo:
    Open strNombreArchivo For Input As #1
    sh.Shapes.AddTextbox(msoTextOrientationHorizontal, _
    10, 10, 300, 200).Select
    Selection.Characters.Text = Input(LOF(1), #1)
CierraArchivo:
    Close #1
End Sub
```

Fíjate en la instrucción `On Error GoTo CierraArchivo:` activa la captura de errores. Si se produce un error durante la ejecución de una línea del procedimiento, la macro saltará a la rutina de gestión de errores que hay a continuación de la etiqueta `CierraArchivo`. La instrucción `Close #1` se ejecutará se tanto si se encuentra un error como si no hay errores. Antes de colocar el contenido del archivo en la hoja de trabajo se añade un cuadro de texto usando el método `AddTextbox` del objeto `Shapes`.

2. Para ejecutar el procedimiento anterior, abre la ventana **Inmediato**, escribe la siguiente declaración y presiona **Intro**:

```
EscribirCuadroTexto "C:\Archivos Manual VBA\Vacaciones.txt"
```

2.4 Leer caracteres de un archivo delimitado

En algunos archivo de texto o planos (normalmente guardados en formato CSV, TXT o PRN), los datos introducidos en cada línea de texto se separan (o delimitan) con una coma, un tabulador o un espacio.

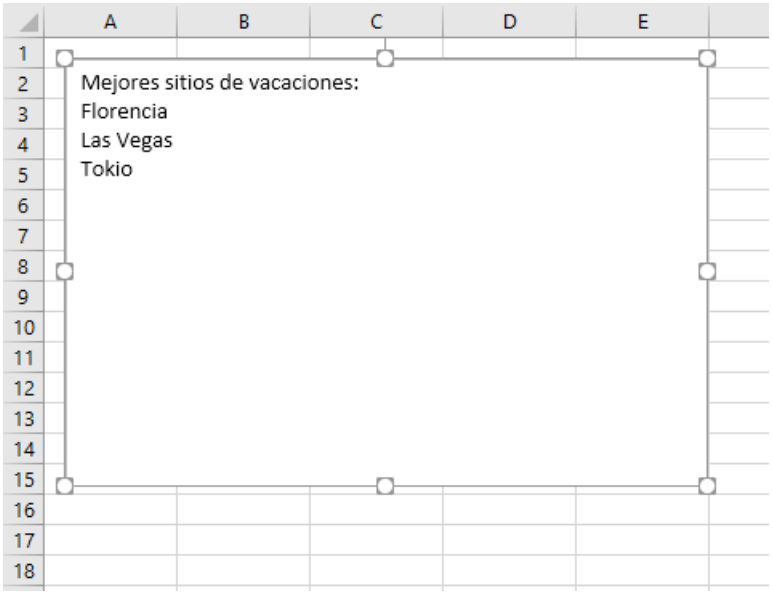


Imagen 2.1 El contenido del archivo de texto se muestra un cuadro incrustado en una hoja de Excel.

Estos tipos de archivo pueden leerse más rápidamente con la declaración `Input #` en lugar de la declaración `Line Input #` que vimos anteriormente. La instrucción `Input #` permite leer los datos de un archivo abierto en varias variables. Esta función tiene el siguiente aspecto:

```
Input #numeroarchivo, listavariables
```

El `numeroarchivo` es el mismo número de archivo que se utilizó en la declaración de apertura. La `listavariables` es una lista separada por comas de las variables que queremos usar para almacenar los datos que se están leyendo. No se pueden utilizar matrices o variables de objeto. Sin embargo, podemos utilizar una variable definida por el usuario (explicada más adelante en este capítulo). A continuación, se muestra un ejemplo de un archivo secuencial con valores delimitados por comas:

```
Martín,Alfredo,15
Acevedo,Alvaro,27
Gutierrez,María,15
```

Observemos que en este ejemplo no hay espacios antes o después de las comas. Para leer el texto formateado de esta manera, debemos especificar una variable para cada dato: apellido, nombre y edad. Probémoslo:

1. Crea un nuevo libro e introduce los datos que se muestran en la Imagen 2.2:

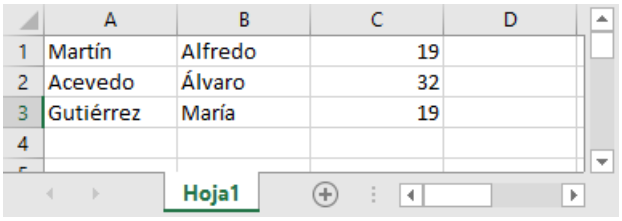


Imagen 2.2 Puedes crear un archivo delimitado por comas a partir de un libro de Excel.

2. Haz clic en la ficha **Archivo** y a continuación, en **Guardar como**. Cambia la carpeta por defecto a C:\Archivo Manual VBA. En el cuadro desplegable **Tipo**, selecciona CSV (delimitado por comas (*.csv)). Cambia el nombre del archivo por Campeones.csv y haz clic en **Guardar**.
3. Si el libro que utilizaste para crear el CSV tenía más de una hoja, Excel mostrará un mensaje indicando que algunas características se perderán si guardas el archivo como CSV. Haz clic en **Sí** para usar este formato.

Atención

El tipo de archivos CSV no es compatible con los libros de trabajo que contienen varias hojas. Si agregas más hojas al libro e intentas guardarlo como CSV, Excel mostrará un mensaje de advertencia.

4. Cierra el archivo Campeones.csv guardando los cambios.
5. Activa el libro Capítulo 12 – Acceso a archivos txt-csv-prn.xlsm y haz que se muestre el editor de VBA.
6. En el **Explorador de proyectos**, haz doble clic en el módulo `Archivo_Secuencial` del proyecto `Acceso_Archivos` e introduce el procedimiento `Campeones` como se muestra a continuación:

```
Sub Campeones()
    Dim lNombre As String
    Dim fNombre As String
    Dim Edad As Integer

    Open "C:\Archivos Manual VBA\Campeones.csv" For Input As #1
    Do While Not EOF(1)
        Input #1, lNombre, fNombre, Edad
        MsgBox lNombre & ", " & fNombre & ", " & Edad
    Loop
    Close #1
End Sub
```

Atención

El procedimiento anterior únicamente funcionará en equipos que tengan comas como delimitador de campos predeterminado. En caso de que utilices punto y coma (;), edita el archivo CSV y sustituye un carácter por otro.

El procedimiento anterior abre el archivo Campeones.csv en modo Input y recorre todo el archivo gracias a la estructura `Do … While`, hasta el final de este. La instrucción `Input`

#1 se usa para almacenar el contenido de cada línea de texto en tres variables: **lNombre**, **fNombre** y **Edad**. Luego, un cuadro de mensaje muestra el contenido de estas variables. El procedimiento finaliza cerrando el archivo Campeones.csv.

7. Ejecuta el procedimiento.

2.5 Escribir datos en archivos secuenciales

Cuando queramos escribir datos en un archivo secuencial, debemos abrir el archivo en modo **Output** o **Append**. Las diferencias entre estos modos se explican a continuación:

- Modo **Output**: Cuando abrimos un archivo en este modo, VBA borra los datos que están actualmente en él. Si el archivo no existe, se creará uno nuevo. Por ejemplo, si abrimos el archivo Vacaciones.txt en modo Output e intentamos escribir algún texto en él,el texto anterior será eliminado. Si no hacemos una copia de seguridad antes de escribir los datos, crearemos un problema que puede ser costoso. Debemos abrir un archivo existente en modo **Output** solamente si queremos reemplazar todo el contenido con nuevos datos.
- Modo **Append**: se usa para añadir datos al final de un archivo de texto existente. Por ejemplo, si abrimos el archivo Vacaciones.txt en modo **Append** y añadimos el texto "Gracias por leer el documento", VBA no borrará ni alterará el texto que se encuentra actualmente en el archivo, sino que añadirá el nuevo texto al final.

A continuación, se muestran algunos ejemplos para conocer cuándo abrir un archivo en modo **Output** o **Append**:

- Para crear un nuevo archivo de texto llamado C:\Archivos Manual VBA\Léeme.txt, abrimos el archivo en modo **Output** de la siguiente forma:

```
Open "C:\Archivos Manual VBA\Léeme.txt" For Output As #1.
```

- Para agregar texto al final del archivo C:\Archivos Manual VBA\Léeme.txt, lo abrimos en modo **Append** de la siguiente forma:

```
Open "C:\Archivos Manual VBA\Léeme.txt" For Append As #1
```

- Para reemplazar el contenido del archivo C:\Archivos Manual VBA\Campeones.csv con una lista de nuevos campeones, primero creamos una copia de seguridad del original, y luego abrimos el original en modo **Output**:

```
FileCopy "C:\Archivos Manual VBA\Campeones.csv",
"C:\Archivos Manual VBA\Campeones.old"
Open "C:\Archivos Manual VBA\Campeones.csv" For Output As #1
```

2.6 Las instrucciones Write # y Print

Ahora que ya conocemos los dos métodos (**Append** y **Output**) para abrir un archivo de texto para escribir en él, es hora de conocer las instrucciones **Write #** y **Print #**, que nos permitirán enviar datos al archivo. Cuando leemos un archivo secuencial con la declaración **Input #**, normalmente se escriben datos con la declaración **Write #**. Esta declaración tiene el siguiente aspecto:

No es posible leer y escribir a la vez

No se pueden realizar operaciones de lectura y escritura de forma simultánea. El archivo debe abrirse por separado para cada operación. Por ejemplo, después de escribir los datos de un archivo que se ha abierto en modo **Output**, este debe cerrarse antes de abrirlo en modo **Append**. No se puede tener abierto en los dos modos al mismo tiempo.

Ventajas y desventajas de archivos secuenciales

Aunque los archivos secuenciales son fáciles de crear y usar, y no desperdician espacio, tienen varias desventajas. Por ejemplo, para buscar un elemento específico debe leerse el archivo desde el principio. Además, para cambiar o eliminar un elemento individual, se debe reescribir todo el archivo.

2.7 Las instrucciones Write # y Print

Ahora que ya conocemos los dos métodos (**Append** y **Output**) para abrir un archivo de texto para escribir en él, es hora de conocer las instrucciones **Write #** y **Print #**, que nos permitirán enviar datos al archivo. Cuando leemos un archivo secuencial con la declaración **Input #**, normalmente se escriben datos con la declaración **Write #**. Esta declaración tiene el siguiente aspecto:

```
Write #filenumber, [outputlist]
```

Filenumber especifica el número del archivo con el que estamos trabajando. Es un argumento obligatorio. **Outputlist** es el texto que queremos escribir en el archivo y puede constar de una sola cadena de texto o de una lista de variables que contengan los datos que deseamos escribir. Si especificamos solo el argumento **filenumber**, VBA escribirá una única línea vacía en el archivo abierto.

Para ilustrar cómo se escriben los datos en un archivo preparemos un archivo de texto con el nombre, el apellido, la fecha de nacimiento y el número de hermanos de tres personas.

1. En el módulo **Archivo_Secuencial** introduce el siguiente procedimiento:

```
Sub IntroDatos()
    Dim apellido As String
    Dim nombre As String
    Dim FNac As Date
    Dim sib As Integer

    Open "C:\Archivos Manual VBA\Amigos.txt" For Output As #1
    apellido = "Martín"
    nombre = "María"
```

```
    FNac = #1/2/1965#
    sib = 3
    Write #1, apellido, nombre, FNac, sib
    apellido = "Jiménez"
    nombre = "Javier"
    FNac = #5/12/1948#
    sib = 1
    Write #1, apellido, nombre, FNac, sib
    apellido = "García"
    nombre = "Gerardo"
    FNac = #4/7/1957#
    sib = 0
    Write #1, apellido, nombre, FNac, sib
    Close #1
End Sub
```

El procedimiento anterior crea un nuevo archivo llamado Amigos.txt en la carpeta del manual creada en C:, lo abre en modo `Output` y luego escribe tres registros.

Los datos escritos en el archivo se almacenan en variables. Fíjate que las cadenas de texto están delimitadas con comillas ("") y la fecha de nacimiento está encerrada entre almohadillas (#).

2. Ejecuta el procedimiento `IntroDatos`.
3. Busca el archivo Amigos.txt en la carpeta C:\Archivos Manual VBA y ábrelo usando el Bloc de notas de Windows. El archivo debe parecerse a algo como esto:

"Martín","María",#1965-01-02#,3
"Jiménez","Javier",#1948-05-12#,1
"García","Gerardo",#1957-04-07#,0

La declaración `Write #` insertó automáticamente comas entre los elementos de datos individuales de cada registro y utilizó la secuencia de retorno de carro `(Chr(13) & Chr(10))` al final de cada línea, de forma que cada nuevo registro comienza en una nueva línea. Cada línea de texto muestra un registro, cada registro comienza con el apellido y finaliza con el número de hermanos.

Para mostrar los datos separados en columnas en lugar de comas, debes sustituir la declaración `Write #` por `Print #`. VBA escribirá los datos de la siguiente manera:

Martín María 02/01/1965 3
Jiménez Javier 12/05/1948 1
García Gerardo 07/04/1957 0

Aunque la declaración **Print #** tiene la misma sintaxis que **Write #**, **Print #** escribe los datos en el archivo secuencial en un formato listo para imprimir. Las variables de la lista pueden estar separadas por punto y coma o por espacios. Para imprimir varios espacios, debes usar la instrucción **Spc(n)**, donde **n** es el número de espacios. Del mismo modo, para introducir una palabra en la quinta columna debes utilizar la instrucción **Tab(5)**. Veamos algunos ejemplos de formato:

- Para agregar una línea vacía a un archivo, escríbela declaración **Write #** seguida de una coma:

```
Write #,
```

- Para introducir el texto "Fruta" en la quinta columna:

```
Write #1, Tab(5);"Fruta"
```

- Para separar las palabras "Fruta" y "Vegetales" con cinco espacios:

```
Write #1, "Fruta";Spc(5);"Vegetales"
```

3 Los archivos de acceso aleatorio

Cuando un archivo contiene datos estructurados, debemos abrirlo en modo aleatorio. Este tipo de apertura nos permite:

- Leer y escribir datos al mismo tiempo.
- Acceder rápidamente a un registro en particular.

En los archivos de acceso aleatorio, todos los registros tienen la misma longitud y cada registro tiene el mismo número de campos de tamaño fijo. La longitud de un registro o campo debe determinarse antes de escribir los datos en el archivo. Si la longitud de una cadena que se está escribiendo en un campo es menor que el tamaño especificado, VBA introduce automáticamente espacios al final de la cadena para rellenar todo el tamaño del campo. Si el texto que se está escribiendo es más largo que el tamaño del campo, el texto que no quepa no se escribirá.

¿Qué es un archivo de acceso aleatorio?

Un archivo de acceso aleatorio es aquel en el que los datos se almacenan en registros a los que se puede acceder sin tener que leer todos los registros que lo preceden.

Para comenzar a trabajar con archivos de acceso aleatorio, vamos a crear una pequeña base de datos para utilizar en una academia de idiomas. Esta base de datos contendrá registros compuestos por dos campos: un término en español y su equivalente en inglés.

1. Inserta un módulo nuevo en el proyecto **Acceso_Archivos** del archivo "Capítulo 12 – Acceso a archivos txt-csv-prn.xlsm". Llámalo **Archivo_Aleatorio**.

2. Introduce la declaración `Option Explicit` y, a continuación, las siguientes declaraciones:

```
' Crea un tipo de dato personalizado llamado Diccionario
Type Diccionario
    en As String * 16 ' Palabras en inglés hasta 16 caracteres
    es As String * 20 ' Palabras en español hasta 20 caracteres
End Type
```

Además de los tipos de datos que ya aprendiste en el Capítulo 4, VBA te permite definir un tipo de datos personalizado utilizando una declaración `Type … End Type` que se sitúa en la parte superior del módulo. Este tipo de datos personalizado se suele denominar "definido por el usuario". El tipo de datos definido por el usuario puede contener varios tipos de datos (string, integer, date, etc.). Cuando se abren los archivos de forma aleatoria, a menudo se crea una variable definida por el usuario, pues dicha variable proporciona un acceso fácil a los campos individuales de un registro.

El tipo de datos que acabas de crear, `Diccionario`, contiene dos elementos declarados como string con el tamaño especificado. El elemento `en` puede aceptar hasta 16 caracteres. El tamaño del segundo elemento (`sp`) no puede exceder los 20 caracteres.

Al sumar las longitudes de ambos elementos, obtendrás una longitud de registro de 36 caracteres (16+20).

La declaración Type

La declaración `Type` permite crear un grupo personalizado de variables mixtas. Esta declaración se utiliza generalmente en los archivos de acceso aleatorio para almacenar datos a modo de campos dentro de registros de un tamaño fijo. En lugar de declarar una variable diferente para cada campo, se agrupan los campos en una variable definida por el usuario utilizando `Type`. Por ejemplo, para definir un registro que contenga tres datos, lo haríamos de la siguiente manera:

```
Type MiRegistro
    pais As String * 20
    ciudad As String * 14
    puntuacion As Integer
End Type
```

Una vez definido el tipo de dato, se debe dar un nombre a la variable de ese tipo:

```
Dim MisDatos As MiRegistro
```

Para acceder a las variables interiores (país, ciudad o puntuación) utilizaremos el siguiente formato:

```
MisDatos.ciudad = "Barcelona"
```

3. Introduce el siguiente procedimiento tras la creación del tipo de dato:

```
Sub InglesAEspanol()
    Dim d As Diccionario
    Dim recNr As Long
    Dim eleg As String
    Dim totalRec As Long

    recNr = 1
    ' Abre el archivo en modo aleatorio
    Open "C:\Archivos Manual VBA\Traductor.txt" _
    For Random As #1 Len = Len(d)
    Do
    ' Pregunta por una palabra en inglés
    eleg = InputBox("Introduce una palabra en inglés", "INGLÉS")
    d.en = eleg
    ' Sale del bucle si se cancela
    If eleg = "" Then Exit Do
    eleg = InputBox("Introduce el equivalente en español " _
    & d.en, "ESPAÑOL" & d.en)
    If eleg = "" Then Exit Do
    d.es = eleg
    ' Escribe el registro
    Put #1, recNr, d
    ' Incrementa el contador de registros
    recNr = recNr + 1
    'Solicita palabras hasta que se cancela
    Loop Until eleg = ""
    totalRec = LOF(1) / Len(d)
    MsgBox "Este archivo contiene " & totalRec & " registros."
    ' close the file
    Close #1
End Sub
```

El procedimiento comienza con la declaración de cuatro variables. La variable `d` se declara como tipo definido por el usuario llamado `Diccionario`. Este tipo se declaro antes del procedimiento mediante la instrucción `Type`. Después de asignar el valor inicial a la variable `recNr`, VBA crea y abre el archivo Traductor.txt para el acceso aleatorio. Le asigna el número 1. La instrucción `Len(d)` le dice a VBA que el tamaño de cada registro es de 36 caracteres. (La variable `d` contiene dos elementos: `es`, de 20 caracteres y `en` de 16).

A continuación, VBA ejecuta las declaraciones que se encuentran dentro del bucle `Do … Until`.

La primera sentencia del bucle solicita una palabra en inglés y la asigna a la variable. El valor de este elemento se pasa entonces al primer elemento de la variable definida por el usuario `d(d.en)`. Cuando se cancela o se dejan de introducir datos, VBA sale del bucle `Do` y ejecuta las declaraciones finales en el procedimiento, que calculan y muestran el número total de registros del archivo. La última declaración cierra el archivo.

Al introducir una palabra en inglés y hacer clic en **Aceptar**, se te pedirá que introduzcas un equivalente en español. Si no introduces una palabra, VBA saldrá del bucle y continuará con las declaraciones que restan. Si introduces el equivalente en español, VBA lo asignará a la elección de la variable y luego lo pasará al segundo elemento de la variable definida por el usuario `d(d.es)`. A continuación, VBA escribirá todo el registro en el archivo usando la siguiente declaración:

```
Put #1, recNr, d
```

Tras escribir el primer registro, VBA aumentará el contador de registros en uno y repetirá las declaraciones de dentro del buble. El procedimiento `InglesAEspanol` te permite introducir cualquier número de registros en el diccionario. Al dejar de introducir palabras, el procedimiento utiliza las instrucciones `LOF` y `Len` para calcular el número total de registros en el archivo y muestra el mensaje:

"Este archivo contiene " & totalRec & " registros"

Después de mostrar el mensaje, VBA cierra el cierra el archivo Traductor.txt.

La creación del archivo de acceso aleatorio es solo el comienzo. A continuación, crearemos el procedimiento `EjercicioVocabulario` para ilustrar cómo trabajar con registros. Aquí aprenderemos algunas instrucciones que nos permitirán encontrar rápidamente los datos apropiados en el archivo.

4. Ejecuta el procedimiento `InglesAEspanol`. Cuando se te solicite, introduce los datos que se muestran en la Imagen 3.1. Por ejemplo, introduce la palabra “house”. Cuando se solicite el equivalente en español, escribe “casa”.

Imagen 3.1 Contenido del archivo de acceso aleatorio abierto con el Bloc de notas

5. Debajo del procedimiento `InglesAEspanol`, introduce el procedimiento `EjercicioVocabulario`:

```
Sub EjercicioVocabulario()
```

```
Dim d As Diccionario
Dim totalRec As Long
Dim recNr As Long
Dim NumAleat As Long
Dim pregunta As String
Dim respuesta As String

    ' Abre un archivo en modo aleatorio
    Open "C:\Archivos Manual VBA\Traductor.txt" _
    For Random As #1 Len = Len(d)
    ' Imprime el número total debytes del archivo
    Debug.Print "There are " & LOF(1) & " bytes en el archivo."
    ' Busca y muestra el número total de registros
    recNr = LOF(1) / Len(d)
    Debug.Print "Número total de registros: " & recNr
    Do
        ' Obtiene un número aleatorio de registro
        NumAleat = Int(recNr * Rnd) + 1
        Debug.Print NumAleat
        ' Busca el registro aleatorio
        Seek #1, NumAleat
        ' Lee el registro
        Get #1, NumAleat, d
        Debug.Print Trim(d.en); " "; Trim(d.es)
        ' Asigna la respuesta a una variable
        respuesta = InputBox("¿Qué significa en español?", d.en)
        ' finish if cancelled
        If respuesta = "" Then Close #1: Exit Sub
        Debug.Print respuesta
        ' Comprueba si la respuesta es correcta
        If respuesta = Trim(d.es) Then
            MsgBox "¡Enhorabuena!"
        Else
            MsgBox "¡Respuesta incorrecta!"
        End If
        ' Sigue haciendo preguntas hasta
        ' que se cancela el cuadro
    Loop While respuesta <> ""
    ' Se cierra el archivo
    Close #1
```

```
End Sub
```

Después de declarar las variables, el procedimiento `EjercicioVocabulario` abre un archivo en modo aleatorio y le dice a VBA la longitud de cada registro: `Len = Len(d)`. A continuación, dos declaraciones imprimen en la ventana **Inmediato** el número total de bytes y el número de registros en el archivo abierto. El número de bytes se muestra gracias a la instrucción `LOF(1)`. El número de registros se calcula dividiendo el archivo entero (`LOF`) entre la longitud de un registro (`Len(d)`). A continuación, VBA ejecuta las declaraciones dentro del bucle hasta que se pulse la tecla **Esc** o se haga clic en **Cancelar**.

La primera sentencia del bucle asigna el resultado de la función `Rnd` a la variable `NumAleat`. La siguiente declaración escribe este número en la ventana **Inmediato**. La instrucción

```
Seek #1, NumAleat, d
```

Le dice a VBA el número de registro a leer y la variable en la que se están leyendo los datos. El primer registro en un archivo de acceso aleatorio está en la posición 1, el segundo registro en la posición 2, y así sucesivamente. La omisión de un número de registro hace que VBA lea el siguiente registro. Los valores de ambos elementos del tipo Diccionario, definido por el usuario se escriben entonces en la ventana **Inmediato**. Las funciones `Trim (d.en)` y `Trim (d.es)` imprimen los valores del registro que se está leyendo sin los espacios de al principio y al final que el usuario pueda haber introducido.

A continuación, VBA muestra un cuadro de introducción de datos para introducir el equivalente en español de la palabra que aparece en el título del cuadro. La palabra se asigna a la variable `respuesta`. De lo contrario, VBA imprime tu respuesta en la ventana **Inmediato** y te notifica si es correcta. Puedes presionar la tecla **Esc** o hacer clic en el botón **Cancelar** en el cuadro de diálogo siempre que desees salir del procedimiento.

Si decides continuar y presionar **Aceptar**, se generará un nuevo número aleatorio, y el programa recuperará la palabra en inglés y te pedirá el equivalente en español.

Puedes modificar el procedimiento para que cada palabra mal traducida sea escrita en una hoja de cálculo. Además, tal vez quieras escribir todos los registros del archivo Traductor.txt en una hoja de trabajo para tener a mano el contenido de tu diccionario.

6. Ejecuta el procedimiento `EjercicioVocabulario`. Cuando se solicite, escribe la palabra en español que corresponda a la palabra en inglés que se muestra en la barra de título. Presiona **Cancelar** para salir del bucle.
7. Presiona **Alt + F11** para activar la ventana de Excel.
8. Haz clic en la ficha **Archivo** en la cinta de opciones y luego en **Abrir**.
9. Busca el archivo Traductor.txt en la carpeta C:\Archivos Manual VBA y haz doble clic sobre él. Excel mostrará el asistente para importar texto mostrado en la Imagen 3.2.

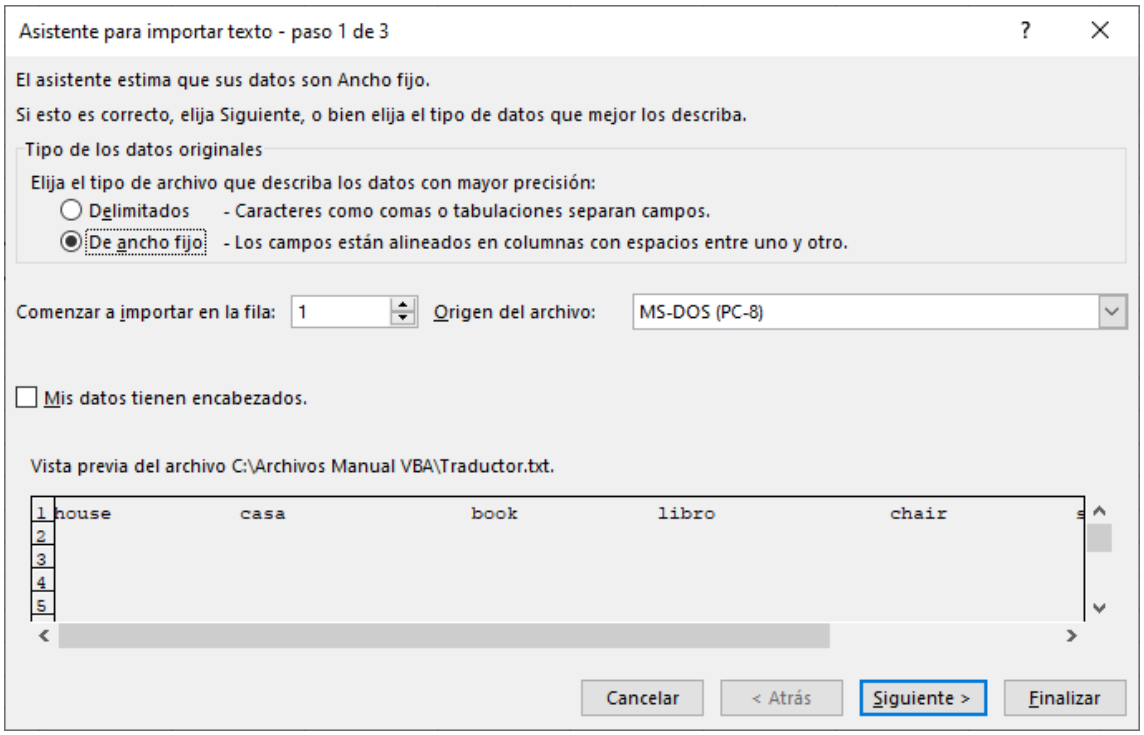


Imagen 3.2 Contenido de un archivo de acceso aleatorio al intentar abrirlo con Excel. Observa que Excel reconoce correctamente el tipo de datos original. Los datos del archivo son de ancho fijo.

10. Haz clic en **Finalizar** para cargar el archivo en Excel.
11. Ya puedes cerrar el archivo.

Ventajas e inconvenientes de los archivos de acceso aleatorio

A diferencia de los archivos secuenciales, se puede acceder a los datos almacenados en archivos de acceso aleatorio muy rápidamente. Además, estos archivos no necesitan ser cerrados cuando se cambia de una operación de lectura a una operación de escritura, y tampoco es necesario leerlos o escribirlos en orden.

Pero los archivos de acceso aleatorio también tienen algunas desventajas. Por ejemplo, a menudo almacenan los datos de forma ineficiente. Como tienen campos y registros de longitud fija, se utiliza el mismo número de bytes sin importar el número de caracteres almacenado. Así que, si algunos campos se dejan en blanco o contienen cadenas más cortas que el tamaño de campo declarado, se puede desperdiciar mucho espacio.

4 Los archivos binarios

A diferencia de los archivos de acceso aleatorio, que almacenan datos en registros de longitud fija, los archivos binarios almacenan registros de longitud variable. Por ejemplo, el primer registro puede contener 10 bytes, el segundo registro puede tener solo 5 bytes, mientras que el tercero puede tener 15 bytes. Este método de almacenamiento de datos ahorra mucho

espacio en el disco porque VBA no necesita añadir espacios adicionales a la cadena almacenada para garantizar que todos los campos tengan la misma longitud.

Al igual que los archivos de acceso aleatorio, los archivos binarios pueden abrirse para operaciones simultáneas de lectura y escritura. Sin embargo, debido a que los registros de los archivos binarios son de longitud variable, resulta más difícil manipular estos archivos. Para recuperar los datos correctamente, se debe almacenar información sobre el tamaño de cada campo y registro. Para trabajar con archivos binarios se utilizarán las siguientes cuatro declaraciones:

- La declaración **Get** se utiliza para leer datos. Tiene la siguiente sintaxis:

  ```
  Get [#]filenumber, [recnumber], varname
  ```

  El argumento **filenumber** es el número utilizado en la declaración **Open** de apertura del archivo. El argumento opcional **recnumber** es el número de registro en archivos de acceso aleatorio, o el número de bytes en los archivos de acceso binario en los que comienza la lectura. Si omitimos el número de registro, el siguiente registro o byte después de la última declaración de **Get** se leerá. Debemos incluir una coma si queremos que salte este argumento. El argumento **varname** (obligatorio) especifica el nombre de la variable que almacenará estos datos.
- La declaración **Put** permite introducir nuevos datos en un archivo binario. Tiene la siguiente sintaxis:

  ```
  Put [#]filenumber, [recnumber], varname
  ```

  Los argumentos **filenumber** y **recnumber** tienen la misma función que en el punto anterior. El argumento **varname** especifica el nombre de la variable que contiene los datos que se escribirán en el disco.
- La declaración **Loc** devuelve el número del último byte que fue leído (en los archivos de acceso aleatorio, **Loc** devuelve el número de registro que se leyó por última vez).
- La declaración **Seek** mueve el cursor a la posición adecuada dentro del archivo.

Para dominar rápidamente el uso de las declaraciones anteriores, vamos a crear un procedimiento:

1. Añade un nuevo módulo al proyecto **Acceso_Archivos** y llámalo **Archivo_Binario**.
2. Introduce en él el procedimiento que encontrarás en el archivo Capitulo_12_datos_aleatorios.txt descargado junto con el manual.
3. Ejecuta el procedimiento **DatosAleatorios**.
4. Abre el archivo Ejemplo Datos.txt creado y examina su contenido.

Cuando introduzcamos datos en un archivo binario debemos tener en cuenta las siguientes directrices:

- Antes de escribir una cadena de texto en un archivo binario debemos asignar la longitud de la cadena a una variable de tipo Integer. Podemos utilizar el siguiente bloque de instrucciones:

```
longitud_texto = Len(nombre_variable)
Put #1, , longitud_texto
Put #1, , nombre_variable
```

- Al leer los datos de un archivo binario, primero se lee la longitud de la cadena y luego el contenido de esta. Para ello, utilizamos la declaración **Get** y la función **String**:

```
Get #1, , longitud_cadena
nombre_variable = String(longitud_cadena, " ")
Get #1, , nombre_variable
```

Ventajas e inconvenientes de los archivos binarios

En comparación con los archivos secuenciales y los de acceso aleatorio, los archivos binarios son los más pequeños. Debido a que utilizan registros de longitud variable, pueden ahorrar espacio en el disco. Al igual que los archivos abiertos en modo aleatorio, se puede leer y escribir simultáneamente en un archivo abierto en modo binario.

Su gran desventaja es que debemos saber con precisión cómo se almacenan los datos en el archivo para recuperarlos o manipularlos correctamente.

5 Resumen

Este capítulo te ha dado un conocimiento práctico de la escritura y la obtención de datos de tres tipos de archivos: secuenciales, de acceso aleatorio y binarios. El siguiente capítulo te introduce a tareas más automatizadas. Aprenderás a usar VBA para controlar otras aplicaciones. También aprenderás varios métodos para iniciar aplicaciones y descubrirás cómo manipularlas directamente desde Microsoft Excel.

Capítulo 13
Interacción con otras aplicaciones

Una de las cosas más bonitas del lenguaje VBA es que puedes usarlo para lanzar y controlar otras aplicaciones de Office. Por ejemplo, puedes crear o abrir un documento de Word directamente desde un procedimiento VBA sin tener que salir de Excel o ver la interfaz de usuario de Word. También puedes iniciar y manipular una serie de programas fuera de Office utilizando las funciones VBA predefinidas. En este capítulo se muestran varios métodos para iniciar otros programas e intercambiar datos entre ellos.

1 Iniciar aplicaciones

Existen varias maneras de iniciar manualmente un programa en el sistema operativo Windows. En esta sección asumo que estás familiarizado con las técnicas manuales de apertura de aplicaciones y que estás ansioso por experimentar con otras técnicas para hacer lo mismo, pero escribiendo código.

Comencemos con la forma más simple: la función `Shell`, que nos permite iniciar cualquier programa directamente desde un procedimiento VBA.

Supongamos que un procedimiento debe abrir la aplicación Bloc de notas de Windows. Para iniciar el Bloc de notas, todo lo que necesitamos es una declaración entre las palabras `Sub` y `End Sub`. O mejor aún, puedes escribir la siguiente declaración en la ventana **Inmediato** y presionar **Intro** para ver el resultado inmediatamente.

```
Shell "notepad.exe", vbMaximizedFocus
```

Aquí, `Notepad.exe` es el nombre del programa que vamos a iniciar. Este nombre debe incluir la ruta completa (el nombre de la unidad y la carpeta en caso de que pueda haber problemas para encontrar el programa). Fijémonos que el nombre del programa está entre comillas. El segundo argumento de la función `Shell` es opcional. Especifica el estilo de la ventana. En este ejemplo, el Bloc de notas aparecerá en una ventana maximizada. Si no se especifica el estilo de la ventana, el programa se mostrará en ventana minimizada y con el foco. La siguiente tabla enumera las constantes de estilo de ventana y las opciones de apariencia:

Constante	Valor	Apariencia
vbHide	0	Oculto
vbNormalFocus	1	Normal con foco
vbMinimizedFocus	2	Minimizada con foco
vbMaximizedFocus	3	Maximizada
vbNormalNoFocus	4	Normal sin foco
vbMinimizedNoFocus	6	Minimizada sin foco

Si la función `Shell` logra lanzar el archivo ejecutable especificado, devolverá un número llamado **Task ID**, que identifica de forma exclusiva la aplicación que se ha lanzado. Si la ejecución de Shell no logra iniciar el programa especificado, VBA genera un error.

La función `Shell` funciona de forma asíncrona. Esto significa que VBA inicia el programa por la función `Shell`, e inmediatamente después de lanzarlo, vuelve al procedimiento para continuar con la ejecución de las instrucciones restantes sin darnos la oportunidad de trabajar con la aplicación. Si queremos trabajar con el programa lanzado por la función `Shell`, no debemos introducir ninguna otra instrucción en el procedimiento después de la función `Shell`. Veamos cómo usar la función `Shell` para lanzar el **Panel de control**.

1. Crea un nuevo archivo y llámalo "Capítulo 13 – Otras aplicaciones.xlsm". Guárdalo en la carpeta C:\Archivos Manual VBA.
2. Dirígete al editor de VBA y renombra el proyecto de VBA a `Int_Apps`.
3. Agrega un nuevo módulo y llámalo `Funcion_Shell`.
4. En la ventana **Código** del módulo, introduce el siguiente procedimiento:

```
Sub IniciarPanel()
    Shell "Control.exe", vbNormalFocus
End Sub
```

5. Ejecuta el procedimiento.

   Cuando se ejecuta el procedimiento `IniciarPanel`, la ventana del **Panel de control** se abre automáticamente encima de cualquier otra ventana. El Panel de control contiene varias herramientas representada por iconos. Como ya sabemos, existe un programa detrás de cada icono que se activa cuando el usuario hace doble clic sobre él o lo selecciona con las teclas de flecha y luego presiona **Intro**. Cada una de las aplicaciones del Panel de control tiene un nombre de archivo ejecutable que se puede visualizar desde las propiedades de estas.

Lamentablemente, los iconos del Panel de control tienen la opción Propiedades desactivada. Sin embargo, puedes encontrar el nombre del archivo del panel de Control creando un atajo. Por ejemplo, antes de crear un procedimiento que cambie la configuración regional del equipo, busquemos el nombre del archivo que activa esta herramienta.

6. En la ventana del **Panel de Control** haz clic en el icono de **Configuración regional**. Haz clic con el botón derecho del ratón en el enlace y selecciona **Crear acceso directo** en el menú contextual.
7. Si se muestra un mensaje para colocar el atajo en el escritorio, indícale que sí.
8. Cierra la ventana **Panel de Control**.
9. Dirígete al escritorio de tu equipo y haz clic en el atajo con el botón derecho del ratón. A continuación, selecciona **Propiedades**.
10. En la ventana **Propiedades**, haz clic en la pestaña **Acceso directo** y luego en el botón **Cambiar icono** para abrir la ventana de cambio de icono que se muestra en la Imagen 1.1.

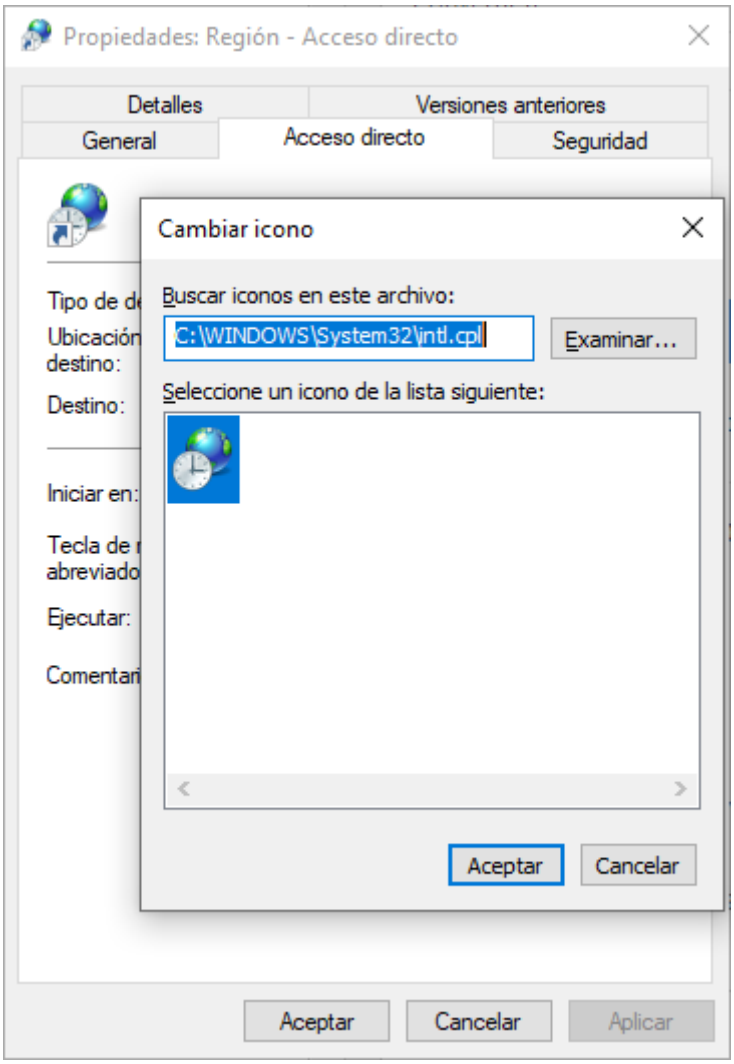


Imagen 1.1 Los iconos de las herramientas del Panel de control tienen la extensión .cpl.

11. Anota el nombre del archivo CPL (Control Panel Library) o del archivo DLL (Dynamic Link Library) que aparece en la parte superior de la ventana del icono.

Estos son algunos de los nombres de herramientas del Panel de Control:

Nombre en el Panel de Control	Nombre del archivo CPL o DLL
Centro de accesibilidad	Access.cpl
Teléfono y módem	Telephon.cpl o modem.cpl
Agregar y quitar programas	Appwiz.cpl
Cuentas de usuarios	Userpasswords, userpasswords2
Fecha y hora	Timedate.cpl
Región	Intl.cpl
Opciones de internet	Inetcpl.cpl

12. En la ventana **Código** del módulo `Funcion_Shell` introduce el procedimiento `CambioConfiguracion` como se muestra a continuación:

```
Sub CambioConfiguracion()
    Dim numTask
    numTask = Shell("Control.exe intl.cpl", vbMinimizedFocus)
    Debug.Print nrTask
End Sub
```

Este procedimiento muestra cómo lanzar el icono de configuración regional del Panel de Control utilizando la función `Shell`. Observa que los argumentos de la función `Shell` deben aparecer entre paréntesis si deseas utilizar el valor devuelto más adelante en el procedimiento. Para abrir la ventana del **Panel de control** con la pestaña de idiomas activada, modifica la siguiente línea en el procedimiento anterior:

```
numTask = Shell("Control.exe intl.cpl 0,1", vbMinimizedFocus)
```

La primera pestaña de la ventana tiene número de índice 0, la segunda el 1 y así sucesivamente.

13. Ejecuta el procedimiento anterior varias veces, cambiando cada vez el archivo según la tabla anterior. Es posible que quieras modificar el procedimiento de la siguiente manera:

```
Sub CambioConfiguracion2()
    Dim numTask
    Dim ArchivoIcono As String
    ArchivoIcono = InputBox("Introduce el nombre del " & _
    "icono CPL o DLL:")
```

```
    numTask = Shell("Control.exe " & _
      ArchivoIcono, vbMinimizedFocus)
    Debug.Print numTask
End Sub
```

Si la aplicación que deseas ejecutar es una aplicación de Microsoft, es más conveniente utilizar el método `ActivateMicrosoftApp` que la función `Shell`. Este método está disponible en el objeto `Microsoft Excel Application`. Por ejemplo, para iniciar PowerPoint desde la ventana **Inmediato** solo tienes que escribir la siguiente instrucción y pulsar **Intro**:

```
Application.ActivateMicrosoftApp xlMicrosoftPowerPoint
```

Observa que el método `ActivateMicrosoftApp` requiere una constante para indicar qué programa debe iniciarse. La declaración anterior inicia Microsoft PowerPoint si no se está ejecutando ya. Si el programa ya está abierto, esta instrucción no abre una nueva ventana del programa; simplemente activa la aplicación que ya se está ejecutando. Puedes utilizar las constantes que se muestran en la siguiente tabla:

Aplicación	Constante
Access	xlMicrosftAccess
Outlook	xlMicrosoftMail
PowerPoint	xlMicrosoftPowerPoint
Project	xlMicrosoftProject
Word	xlMicrosoftWord

2 Moverse entre aplicaciones

Debido a que una persona puede trabajar simultáneamente con varias aplicaciones en el entorno de Windows, nuestros procedimientos VBA deben saber cómo cambiar de ventana activa entre los programas abiertos. Supongamos que además de Microsoft Excel tenemos otras dos aplicaciones abiertas: Microsoft Word y el Explorador de Windows. Para activar un programa ya abierto, utilizamos la declaración `AppActivate` con la siguiente sintaxis:

```
AppActivate title [, wait]
```

Solo es obligatorio el argumento `title`. Éste es el nombre de la aplicación tal como aparece en la barra de título de la ventana de la aplicación activa o su número de identificación de tarea tal como lo devuelve la función `Shell`. El argumento opcional `wait` es un valor booleano (Verdadero/Falso) que especifica cuándo se debe activar la aplicación. El valor `False` activa inmediatamente la aplicación especificada, incluso si no tiene el foco. Si

introducimos `True`, la aplicación que llama esperará a tener el foco antes de activar la aplicación especificada.

Por ejemplo, de esta forma podemos activar Microsoft Word:

```
AppActivate "Microsoft Word"
```

Observemos que el nombre de la aplicación está encerrado entre comillas. También podemos usar el valor devuelto por la función `Shell` como argumento de la declaración `AppActivate`:

```
Sub AbrirWord()
Dim ValorDevuelto As Variant

    ValorDevuelto = Shell("C:\Archivos de Programa\Microsoft
Office\" & _
    "root\office16\WINWORD.EXE /w", 1)

    AppActivate ValorDevuelto
End Sub
```

En el procedimiento, la cláusula "`/w`" después del nombre del archivo iniciará una nueva instancia de Word con un documento en blanco.

La declaración `AppActivate` se utiliza para moverse entre aplicaciones y requiere que el programa ya esté en funcionamiento. Esta declaración simplemente cambia el foco. La aplicación especificada se convierte en la ventana activa. La instrucción `AppActivate` no iniciará la ejecución de una aplicación.

3 Controlar otra aplicación

Ahora que sabemos cómo usar las declaraciones de VBA para iniciar un programa y cambiar de aplicación, veamos cómo hacer que una aplicación se comunique con otra. La forma más simple para que una aplicación obtenga el control de otra es por medio de la declaración `SendKeys`. Esta declaración permite enviar una serie de pulsaciones de teclas a la ventana de la aplicación activa. Podemos "enviar" una tecla o una combinación de ellas y obtener el mismo resultado que si trabajáramos directamente en la ventana de la aplicación activa mediante el teclado. La instrucción `SendKeys` tiene el siguiente aspecto:

```
SendKeys string [, wait]
```

El argumento obligatorio `string` es la tecla o combinación de teclas que queremos enviar a la aplicación activa. Por ejemplo, para enviar la letra "e", utilizamos la siguiente instrucción:

```
SendKeys "e"
```

Para enviar la combinación de teclas **Alt + e** utilizamos:

```
SendKeys "%e"
```

El signo de porcentaje (%) es el símbolo utilizado para la tecla **Alt**.

Para enviar una combinación de teclas como **Mayús + Tab**, utilizamos la siguiente declaración:

```
SendKeys "+{TAB}"
```

El signo más (+) indica la tecla Mayús.

Para enviar otras teclas y combinaciones de teclas te recomiendo ver la siguiente tabla:

Tecla	Código
Retroceso	{BACKSPACE}{BS}{BKSP}
Inter	{BREAK}
Bloq Mayús	{CAPSLOCK}
Supr	{DELETE}{DEL}
Flecha hacia abajo	{DOWN}
Fin	{END}
Intro	{ENTER} o ~
Esc	{ESC}
Ayuda	{HELP}
Inicio	{HOME}
Insertar	{INSERT}{INS}
Flecha hacia la izquierda	{LEFT}
Bloq Num.	{NUMLOCK}
Av Pág.	{PGDN}
Re Pág.	{PGUP}
Impr pant	{PRTSC}
Flecha hacia la derecha	{RIGHT}
Bloq despl	{SCROLLLOCK}
Tab	{TAB}
Flecha hacia arriba	{UP}
Mayús	+
Ctrl	^
Alt	%

El segundo argumento de la declaración **SendKeys**, **wait**, es opcional. Es un valor lógico (Verdadero o Falso). Si es Falso (predeterminado), VBA vuelve el procedimiento inmediatamente después de enviar las teclas. Si **wait** es Verdadero, VBA vuelve al procedimiento solo después de ejecutar las pulsaciones de teclas enviadas.

Para enviar caracteres que no se muestran al pulsar una tecla, utilizamos los códigos de la tabla anterior. Recuerda que los códigos deben ir entre comillas.

Solo podemos enviar pulsaciones de teclas a aplicaciones diseñadas para Windows. Podemos utilizar la instrucción **SendKeys** para activar una ficha de la cinta de opciones.

Por ejemplo, para activar la ficha **Insertar** y seleccionar el comando **Insertar imagen**, utilizamos la siguiente instrucción:

```
SendKeys "%b2i1d"
```

Para ver la lista de los atajos de teclado, debemos pulsar la tecla **Alt**.

Se mostrarán las teclas asignadas a cada comando de la Cinta de opciones como se muestra en la Imagen 3.1. Después de presionar la tecla de la pestaña que deseamos activar, la Cinta mostrará las teclas de acceso asignadas a comandos individuales, como se muestra en la Imagen 3.2.

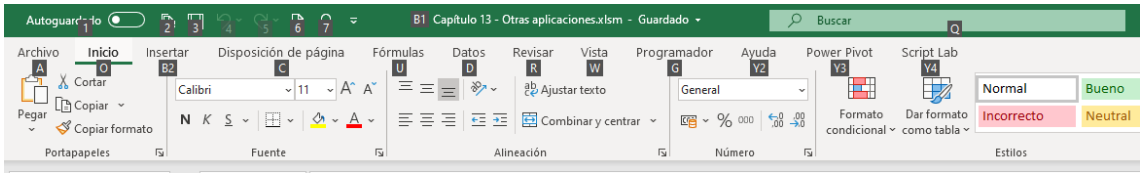


Imagen 3.1 Las fichas de la Cinta de opciones tiene una clave de acceso para poder utilizarla con la declaración SendKeys

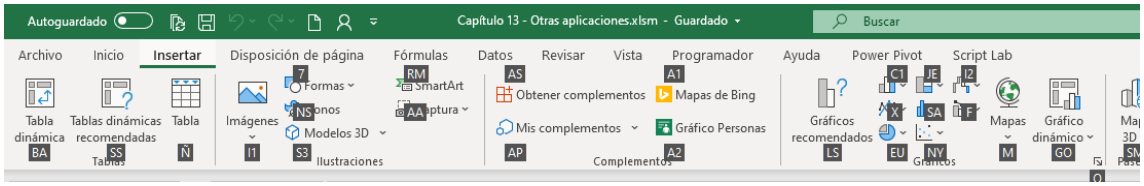


Imagen 3.2 Cada comando de la Cinta también tiene una clave de acceso.

SendKeys y los caracteres reservados

Algunos caracteres tienen un significado especial cuando se usan con SendKeys. Estos caracteres son: el signo más (+), el acento circunflejo (^), la virgulilla (~) y los paréntesis (). Para enviar estos caracteres a otra aplicación es necesario introducirlos entre llaves. Por ejemplo, para enviar llaves, debemos introducir {{} y {}}.

1. Inserta un módulo nuevo en el proyecto **Int_Apps** y llámalo **Decl_SendKeys**.

2. Introduce el siguiente procedimiento:

```
Sub BuscaArchivosXLSM()
    ' El procedimiento funciona a partir de Windows 8
    Shell "Explorer", vbMaximizedFocus
    ' Espera 5 segundos para que se ejecute
    Application.Wait (Now + TimeValue("0:00:05"))
    ' Activa la ventana de búsqueda
    SendKeys "{F3}", True
    ' Espera otros 5 segundos
    Application.Wait (Now + TimeValue("0:00:05"))
    ' Cambia la localización de la búsqueda
    ' para buscar en todas las carpetas
    ' en la unidad C
    SendKeys "%js", True
    SendKeys "%c", True
    SendKeys "%js", True
    SendKeys "%a", True
    ' Activa el cuadro de búsqueda
    SendKeys "{F3}", True
    ' Introduce el texto a buscar
    SendKeys "*.xlsm", True
    ' Ejecuta la búsqueda
    SendKeys "{ENTER}", True
End Sub
```

3. Activa la ventana de Excel y ejecuta el procedimiento `BuscaArchivosXLSM` (usa **Alt + F8** para abrir el cuadro de diálogo **Macro**, selecciona el nombre del procedimiento y haz clic en **Ejecutar**).
   Observa lo que sucede en la ventana de resultados de búsqueda, ya que el procedimiento envía pulsaciones de tecla que activan la función de búsqueda.

SendKeys y las mayúsculas

Cuando envíes pulsaciones de teclas con la declaración `SendKeys`, ten en cuenta que debes distinguir entre mayúsculas y minúsculas. Por lo tanto para enviar la combinación **Ctrl + d**, debes utilizar `^d` y para enviar **Ctrl+Mayús+d** debes utilizar `^+d`.

4 Otros métodos para controlar aplicaciones

Aunque puedes pasar comandos a otros programas utilizando la declaración `SendKeys`, para obtener el control de otra aplicación debemos recurrir a otros métodos. Hay dos formas estándar en las que las aplicaciones pueden comunicarse entre sí. Usando la **Automatización** (Automation) puedes escribir procedimientos VBA que controlen otras aplicaciones haciendo referencia a los objetos, propiedades y métodos de dicha aplicación. También existe otra tecnología de intercambio de datos bastante más antigua llamada **DDE** (Dynamic Data Exchange), sin embargo, es un método lento y complicado. DDE es un protocolo que permite enviar datos dinámicamente entre dos programas creando un canal especial para enviar y recibir información. Este método se suele utilizar en aplicaciones antiguas que no son compatibles con la automatización.

4.1 ¿Qué es Automation?

Automation

La Automatización (anteriormente Automatización OLE) es una característica del Modelo de Objetos de Componentes (COM), una tecnología estándar de la industria que las aplicaciones utilizan para exponer sus objetos a herramientas de desarrollo, lenguajes de macro y otras aplicaciones que soportan la Automatización.

Cuando nos comunicamos con otra aplicación es posible que necesitemos más funciones que un simple envío de teclas. Por ejemplo, puede que queramos crear y manipular objetos dentro de esa aplicación o incrustar un documento entero de Word en una hoja de trabajo de Microsoft Excel. Dado que tanto Excel como Word admiten la automatización, podemos crear un procedimiento VBA en Excel para manipular objetos de Word, como documentos o párrafos. Las aplicaciones que soportan la automatización se llaman “servidores de automatización” u “objetos de automatización”. Las aplicaciones que pueden manipular los objetos de un servidor se denominan “controladores de automatización”. Algunas aplicaciones pueden ser solo servidor o solo controlador, y otras pueden actuar como ambos roles. A partir del lanzamiento de Office 2000, todas las aplicaciones de Microsoft Office pueden actuar como servidores y controladores. Los controladores pueden ser todo tipo de objetos ActiveX instalados en el ordenador.

4.2 Vinculación e incrustación de objetos

La vinculación e incrustación de objetos (OLE por sus siglas en inglés) permite crear “documentos compuestos”.

Un documento compuesto contiene objetos creados por otras aplicaciones. Por ejemplo, si se incrusta un documento de Word en una hoja de Excel, Excel solo necesita saber el nombre de la aplicación que se utilizó para crear este objeto y el método de visualización del objeto en la pantalla. Los documentos compuestos se crean enlazando o incrustando objetos. Cuando se utiliza el método manual para incrustar un objeto, primero debemos copiarlo desde una aplicación y luego pegarlo en otra. La diferencia principal entre un objeto enlazado y un objeto incrustado está en la forma en que el objeto se almacena y actualiza. El objeto incrustado pasa

a formar parte del archivo de destino. Debido a que el objeto incrustado no está conectado con los datos originales, la información es estática. Cuando los datos cambian en el archivo de origen, el objeto incrustado no se actualiza. Para cambiar los datos incrustados debemos hacer doble clic en él. Esto abrirá el objeto para editarlo en el programa fuente. Por supuesto, el programa fuente debe estar instalado en el ordenador. Al incrustar objetos, todos los datos se almacenan en el archivo de destino. Esto hace que el tamaño del archivo aumente considerablemente. Cuando incrustamos un objeto, en la barra de fórmulas se muestra lo siguiente:

```
=INCRUSTAR("Word.Document.12";"")
```

El número que va a continuación de `Word.Document` indica la versión que estamos utilizando. `12` significa que estamos incrustando un objeto de Word 2019.

Cuando hacemos doble clic en un objeto vinculado, se lanza la aplicación de origen. La vinculación de objetos es una operación dinámica. Esto significa que los datos enlazados se actualizan automáticamente cuando los datos del archivo fuente cambian. Dado que el documento de destino solo contiene información sobre cómo se enlaza el objeto con el documento fuente, la vinculación de objetos no aumenta el tamaño del archivo de destino. En Excel se mostrará la siguiente fórmula, que se utiliza para enlazar un objeto:

=Word.Document.12|'C:\Archivos Manual VBA\Funciones.docx'!''''

El siguiente procedimiento muestra cómo incrustar un documento de Word en una hoja de trabajo de Excel mediante programación. Debes reemplazar la referencia a C:\Archivos Manual VBA\Hola.docx con tu propio nombre de documento.

1. Inserta un módulo nuevo en el proyecto y llámalo OLE.
2. Introduce el siguiente procedimiento:

```
Sub InsertarCarta()
    Workbooks.Add
    ActiveSheet.Shapes.AddOLEObject _
    Filename:="C:\Archivos Manual VBA\Hola.docx"
End Sub
```

   El procedimiento `InsertarCarta` utiliza el método `AddOLEObject`. Este método crea un objeto OLE y devuelve el objeto `Shape` que representa.

3. Ejecuta el procedimiento
   El procedimiento abre un nuevo libro de trabajo e incrusta en él el documento de Word indicado. Si prefieres enlazar un documento, debes especificar un argumento adicional, `Link` como se muestra a continuación:

```
ActiveSheet.Shapes.AddOLEObject _
FileName:="C:\Archivos Manual VBA\Hola.docx", _
Link:=True
```

Objetos: enlazar o incrustar

Utiliza la incrustación de objetos en lugar de la vinculación cuando:

- No te importa si el tamaño del documento aumenta o si tienes suficiente espacio en el disco y memoria para manejar archivos grandes.
- No necesitas el archivo fuente o usas el texto fuente en otro documento compuesto.
- Quieres enviar el documento a otras personas por correo electrónico y quieres asegurarte de que pueden leer los datos sin problemas.

5 Vinculaciones

Antes de poder trabajar con un objeto externo, debemos crear una instancia del mismo (una copia). Esto puede hacerse de dos formas: mediante vinculación temprana o vinculación tardía. La vinculación implica que las llamadas de funciones escritas por el programador se hacen coincidir con el código que implementa la función.

5.1 Vinculación temprana

Para utilizar la vinculación temprana debemos crear una referencia a la biblioteca de objetos mediante los comandos **Herramientas – Referencias** del editor de VBA para abrir un cuadro de diálogo como el que se muestra en la Imagen 5.1. Después activamos la casilla correspondiente a la biblioteca de objetos con la que necesitamos establecer la referencia:

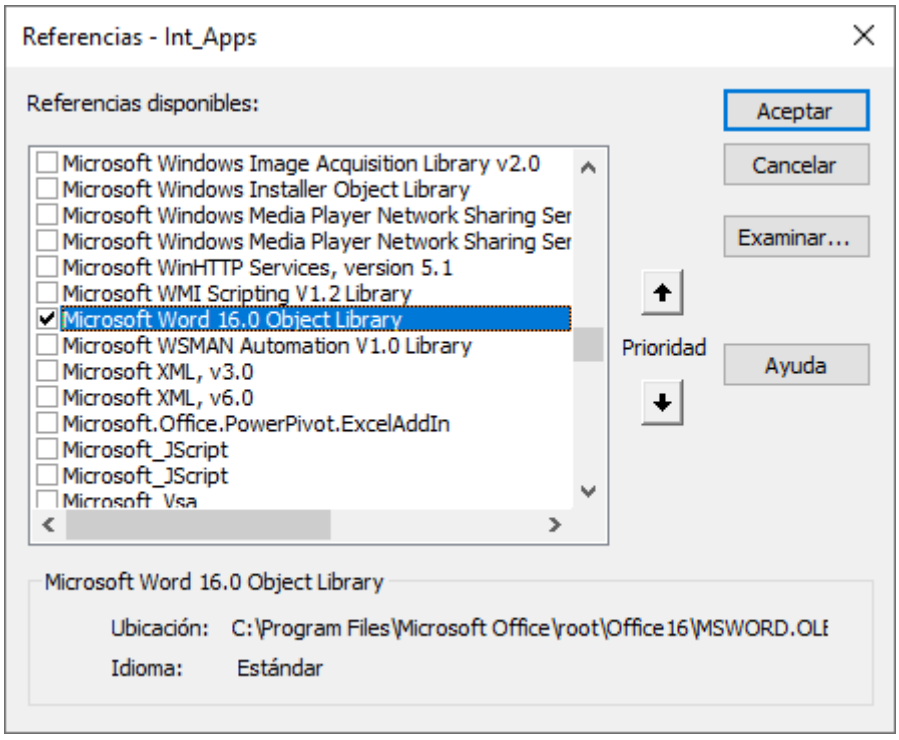


Imagen 5.1 En el cuadro Referencias podemos activar las bibliotecas que vayamos a utilizar.

Una vez establecida la referencia a la biblioteca de objetos, podemos utilizar el **Examinador de objetos** para ver los nombres, los métodos y las propiedades de los objetos.

Al utilizar la vinculación temprana, debemos establecer una referencia a una versión específica de la biblioteca de objetos. Por ejemplo, podemos especificar Microsoft Word 16.0 Object Library (para Word 2019). Tras ello utilizamos una instrucción como la siguiente para crear el objeto:

```
Dim WordApp as New Word.Application
```

El uso de la vinculación temprana para crear un objeto estableciendo una referencia a la biblioteca de objetos suele ser más eficaz y también ofrece un mejor rendimiento. La vinculación temprana es una opción válida solo si el objeto que estamos controlando tiene un tipo de biblioteca diferente o un archivo de biblioteca de objetos distinto. También será necesario asegurarse de que el usuario de la aplicación dispone de una copia de la biblioteca específica instalada.

Otra ventaja de la vinculación temprana es que se pueden utilizar constantes definidas en la biblioteca de objetos. Por ejemplo, Word (como Excel) cuenta con numerosas constantes predefinidas que puedes utilizar en tu código VBA. Si optas por la vinculación temprana puedes usar las constantes en tu código. Si te decantas por la vinculación tardía, tendrás que utilizar el valor real en lugar de la constante.

Otra ventaja es que puedes aprovechar el **Examinador de objetos** del editor de Visual Basic y la opción **Miembros de autolista** para que sea más fácil acceder a propiedades y métodos; esta función no funciona con la vinculación tardía, ya que el tipo de objeto solo se conoce en el tiempo de ejecución.

5.2 Vinculación tardía

En el tiempo de ejecución se puede utilizar la función `CreateObject` para crear el objeto, o la función `GetObject` para obtener una instancia guardada del mismo. Este objeto se declara de tipo Object (genérico) y la referencia al mismo se resuelve durante el tiempo de ejecución.

Se puede utilizar la vinculación tardía incluso cuando se desconoce qué versión de la aplicación está instalada en el sistema del usuario. Por ejemplo, el siguiente código, que funciona con Word 97 y versiones posteriores, crea un objeto Word:

```
Dim WordApp As Object
Set WordApp = CreateObject("Word.Application")
```

Si existen varias versiones de Word instaladas, puedes crear un objeto para una versión específica. La siguiente instrucción, por ejemplo, utiliza Word 2003:

```
Set WordApp = CreateObject("Word.Application.16")
```

La clave de registro para la automatización del objeto de Word y la referencia para el objeto `Application` en VBA es la misma: `Word.Application`. Sin embargo, no hacen referencia a lo mismo. Al declarar un objeto como `As Word.Application` o `As NewWord.Application`, el término se refiere al objeto `Application` de la biblioteca de Word. Pero cuando se invoca la función `CreateObject ("Word.Application")`, el

término se refiere al nombre por el que se conoce a la última versión de Word en el registro del sistema de Windows. Este no es el caso para todos los objetos de **Automatización**, aunque sí para los componentes principales de Office.

El siguiente ejemplo muestra cómo crear un objeto Word mediante vinculación tardía. Este procedimiento crea el objeto, muestra el número de versión, cierra Word y, tras ello, destruye el objeto (liberando la memoria que utilizaba):

```
Sub ObtenerVersionWord()
    Dim WordApp As Object

    Set WordApp = CreateObject("Word.Application")
    MsgBox WordApp.Version
    WordApp.Quit
    Set WordApp = Nothing
End Sub
```

Este ejemplo también se puede programar mediante vinculación temprana. Antes de hacerlo, ejecutamos **Herramientas - Referencias** para establecer una referencia a la biblioteca de objetos Word. Tras ello, utilizamos este código:

```
Sub ObtenerVersionWord()
    Dim WordApp As New Word.Application

    MsgBox WordApp.Version
    WordApp.Quit
    Set WordApp = Nothing
End Sub
```

6 Enviar un correo electrónico personalizado a través de Outlook

En la Imagen 6.1 se muestra una lista de varios correos electrónicos con sus respectivos nombres y una cantidad de comisión asignada.

	A	B	C	D	E
1	**Nombre**	**Correo**	**Comisión**		
2	Sergio Propergol	sergio@ayudaexcel.com	100,00 €		
3	Yolanda Mimética	ymiimetyca@gmail.com	150,00 €		
4	Manuel Jimeno	mjimeno@gmail.com	75,00 €		
5					
6					
7					
8					

Imagen 6.1 Datos para el envío de correos.

El siguiente ejemplo realiza un bucle por las filas de la hoja de trabajo, recupera los datos y crea un mensaje personalizado (que se almacena en la variable **Msg**):

```
Sub EnviaEmail()
    'Utiliza vinculación temprana
    'Requiere una referencia a la Biblioteca Outlook Object
    Dim OutlookApp As Outlook.Application
    Dim Correo As Outlook.MailItem
    Dim Celda As Range
    Dim Asunto As String
    Dim CorreoDes As String
    Dim Recipient As String
    Dim Comision As String
    Dim Msg As String

    'Crea el objeto Outlook
    Set OutlookApp = New Outlook.Application
    'Realiza un bucle por las filas
        For Each Celda In Columns("B").Cells.SpecialCells _
        (xlCellTypeConstants)
        If Celda.Value Like "*@*" Then
            'Obtiene los datos
            Asunto = "Tu comisión anual"
            Recipient = Celda.Offset(0, -1).Value
            CorreoDes = Celda.Value
            Comision = Format(Celda.Offset(0, 1).Value, "0.0 €")
            'Crea el mensaje
            Msg = "Estimado/a " & Recipient & vbCrLf _
            & vbCrLf
            Msg = Msg & "Me complace informarle de que su
comisión anual es "
            Msg = Msg & Comision & vbCrLf & vbCrLf
            Msg = Msg & "José Luis López" & vbCrLf
            Msg = Msg & "Presidente"
            'Crea el objeto Correo y lo envía
            Set Correo = OutlookApp.CreateItem _
            (olMailItem)
            With Correo
                .To = CorreoDes
                .Subject = Asunto
                .Body = Msg
```

```
                .Send
            End With
        End If
    Next
    Set OutlookApp = Nothing
End Sub
```

Este ejemplo utiliza vinculación temprana, por lo que necesitas una referencia a la biblioteca `Outlook Object`. Fíjate en que hay dos objetos: un objeto Outlook y un objeto `MailItem`. El objeto `Outlook` se crea con la siguiente instrucción:

```
Set OutlookApp = New Outlook.Application
```

El objeto `MailItem` se crea con esta otra instrucción:

```
Set MItem = OutlookApp.CreateItem (olMailItem)
```

El código establece las propiedades `To`, `Subject` y `Body`, y, tras ello, utiliza el método `Send` para enviar cada uno de los mensajes. A menos que hayas modificado la configuración de seguridad, probablemente verás un cuadro de diálogo que te pregunta si deseas que se envíe un correo en tu nombre. Para que este cuadro de diálogo no vuelva a aparecer, activa Outlook y selecciona **Herramientas>Centro de confianza**. En el cuadro de diálogo que aparecerá, haz clic en **Acceso** mediante programación y selecciona la opción **No avisarme nunca sobre la actividad sospechosa (no recomendado)**. Pero haz esto bajo tu responsabilidad.

6.1 Enviar correos con datos adjuntos

Como ya sabemos, Excel dispone de comandos para enviar hojas de trabajo o libros por correo electrónico. Y, evidentemente, también es posible utilizar VBA para automatizar este tipo de tareas. El siguiente procedimiento envía el libro de trabajo activo (como datos adjuntos a midireccion@midominio.com). El mensaje de correo electrónico tiene como asunto "Mi Libro".

```
Sub EnviarLibro()
    ActiveWorkbook.SendMail "midireccion@midominio.com", _
    "Mi Libro"
End Sub
```

Si únicamente queremos enviar por correo electrónico una hoja de un determinado libro de trabajo, tendremos que copiarla a un nuevo libro (temporal), enviar dicho libro como datos adjuntos de correo y, tras ello, cerrar el libro temporal. El siguiente ejemplo envía la Hoja1 del libro de trabajo activo:

```
Sub EnviarHoja()
    ActiveWorkbook.Worksheets("Hoja1").Copy
    ActiveWorkbook.SendMail "midireccion@midominio.com", _
    "Mi Hoja"
    ActiveWorkbook.Close False
End Sub
```

En el ejemplo anterior, el archivo tendrá el nombre predeterminado del libro de trabajo (por ejemplo Libro2.xlsx). Si queremos asignar un nombre más descriptivo a nuestro libro de trabajo de una sola hoja, debes guardar el libro temporal y, tras ello, eliminarlo una vez enviado. El siguiente procedimiento guarda Hoja1 en el archivo Mi Archivo.xlsx. Tras enviar este libro de trabajo temporal como datos adjuntos del correo, el código utiliza la instrucción `Kill` de VBA para eliminar el archivo:

```
Sub EnviarHoja()
    Dim NombreArchivo As String

    NombreArchivo = "Mi archivo.xlsx"
    ActiveWorkbook.Worksheets("Hoja1").Copy
    ActiveWorkbook.SaveAs NombreArchivo
    ActiveWorkbook.SendMail "midireccion@midominio.com", _
    "Mi Hoja"
    ActiveWorkbook.Close False
    Kill NombreArchivo
End Su
```

Excel incluye un comando para enviar un libro de trabajo como archivo PDF.

7 Resumen

En este capítulo has aprendido a lanzar, abrir, activar y controlar otras aplicaciones desde procedimientos. Has aprendido a enviar pulsaciones de teclas a otra aplicación utilizando el método `SendKeys` y a enlazar e incrustar objetos de forma manual y con programación. También has aprendido diversas formas de enviar correos electrónicos a través de Outlook.

En el siguiente capítulo aprenderás varios métodos para controlar Microsoft Access desde Excel.

Capítulo 14
Uso de Access a través de Excel

En el capítulo anterior aprendiste a controlar Microsoft Word y Outlook desde Excel a través de la automatización. Este capítulo te muestra cómo usar Access desde Excel mediante programación y cómo recuperar los datos de Access en una hoja de trabajo de Excel utilizando los siguientes métodos:

- Automatizacón
- DAO (Data Access Objects)
- ADO (ActiveX Data Objects)

Antes de aprender a usar VBA para realizar varias tareas en una base de datos como Access, examinemos brevemente los métodos de acceso a los datos que utiliza Access para obtener acceder a sus objetos mediante programación.

1 Bibliotecas de objetos

Una base de datos de Microsoft Access consiste en varios tipos de objetos almacenados en diferentes bibliotecas de objetos. En este capítulo accederemos a los objetos, propiedades y métodos de varias bibliotecas que se enumeran a continuación:

- La biblioteca de objetos de Microsoft Access 16.0.

  Esta biblioteca, que se muestra en la Imagen 1.1, proporciona objetos que se utilizan para mostrar datos y trabajar con Microsoft Access. La biblioteca está almacenada en el archivo MSACC.OLB y se puede encontrar en la carpeta C:\Archivos de programa (x86)\Microsoft Office\root\office16. Después de establecer una referencia a esta biblioteca en el cuadro de diálogo **Referencias** (lo cual se trata en la siguiente sección), podemos ser capaces de buscar los objetos, propiedades y métodos de esta biblioteca en el **Examinador de objetos**.
- La biblioteca de objetos de Microsoft DAO 3.6.

Los Objetos de Acceso a Datos (DAO por sus siglas en inglés) que proporciona esta biblioteca nos permiten determinar la estructura de nuestras bases de datos y manipular los datos utilizando VBA. Esta biblioteca se almacena en el archivo DAO360.dll y se puede encontrar en la carpeta C:\Archivos de programa\Common Files\Microsoft Shared\DAO. Después de configurar la referencia a esta biblioteca en el cuadro de diálogo **Referencias**, podremos buscar los objetos, propiedades y métodos de la biblioteca en el **Explorador de Objetos** (ver Imagen 1.2).

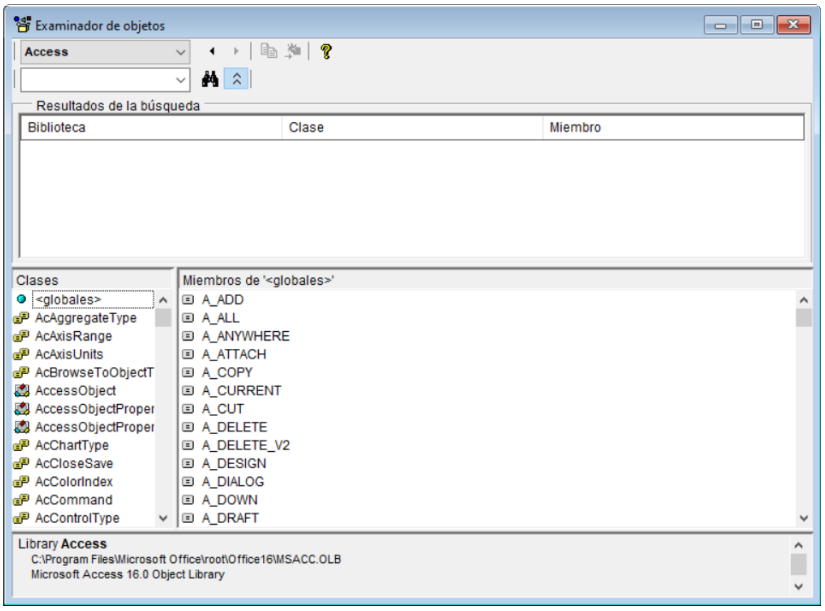


Imagen 1.1 Biblioteca de objetos de Access.

- La biblioteca de objetos de Microsoft ActiveX 6.1 (ADO)

  Los objetos de datos ActiveX (ADO por sus siglas en inglés) proporcionados por esta biblioteca nos permiten acceder y manipular los datos objetos usando el proveedor OLEDB. Los objetos ADO permiten establecer una conexión con una fuente de datos para leer, insertar, modificar y borrar datos en una base de datos Access. Después de configurar la referencia a esta biblioteca en el cuadro de diálogo **Referencias**, podremos acceder a los objetos, propiedades y métodos desde el Explorador de objetos (ver Imagen 1.3).

Importante

En sistemas de 32 bits, debemos buscar los archivos mencionados en la carpeta **Archivos de programa (x86)** en lugar de **Archivos de programa**.

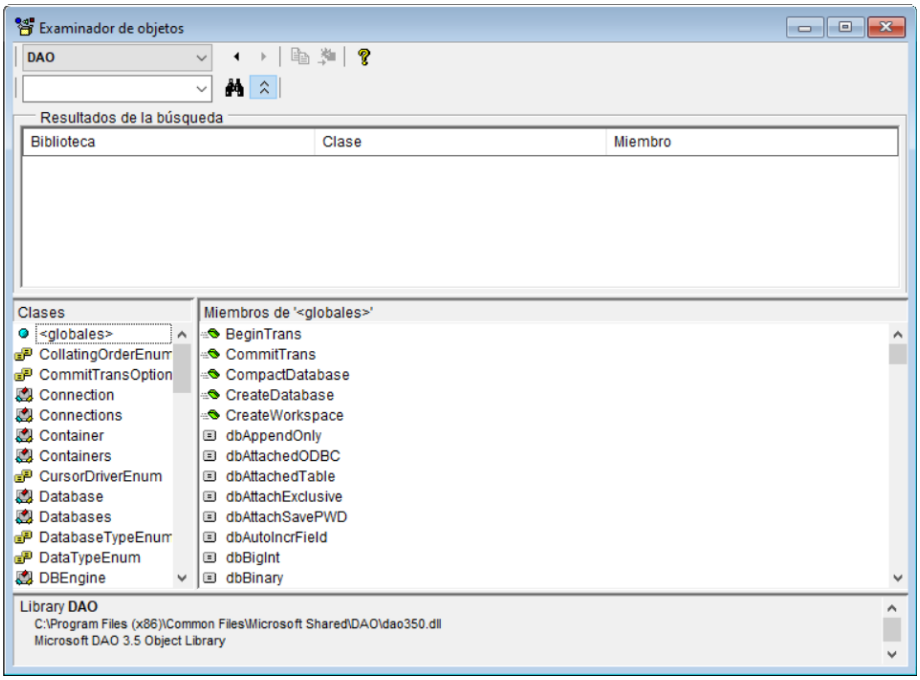


Imagen 1.2 La biblioteca DAO 3.6

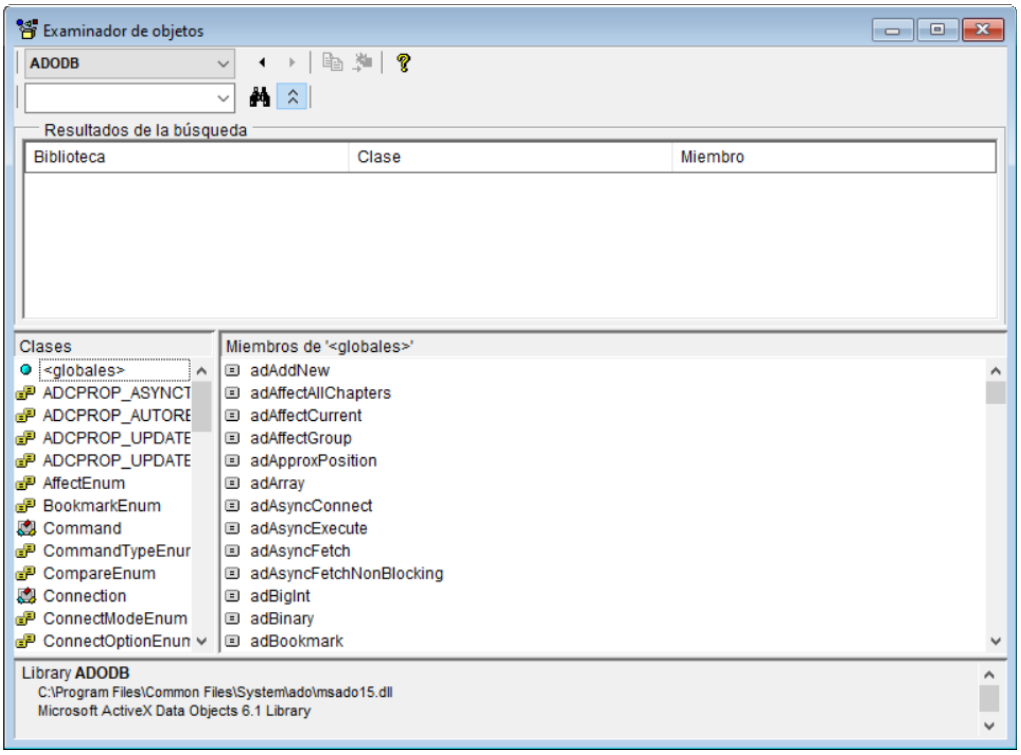


Imagen 1.3 La biblioteca ADODB.

- La biblioteca de Microsoft ADO Ext. 6.0 para DDL y Seguridad (ADOX)

  Los objetos que se almacenan en esta biblioteca permiten definir la base de datos, estructura y seguridad. Por ejemplo, se pueden definir tablas, índices y relaciones, así como crear cuentas de usuario y de grupo. Después de configurar la referencia a esta

biblioteca encontraremos los objetos relacionados con ella en el Explorador de objetos.

- La biblioteca de Microsoft Jet and Replication Objects (JIRO)

  Los objetos contenidos en esta biblioteca se utilizan en la réplica de una base de datos. Tras configurar la referencia a la biblioteca podremos buscar sus objetos en el Explorador de objetos.
- La biblioteca de objetos de Visual Basic for Applications (VBA).

  Los objetos contenidos en esta biblioteca nos permiten acceder al sistema de archivos de nuestro ordenador, trabajar con funciones de fecha y hora, realizar cálculos matemáticos y financieros, interactuar con los usuarios, convertir datos y leer archivos de texto. La referencia a esta biblioteca se establece automáticamente cuando se instala Excel. Esta biblioteca se comparte entre todas las aplicaciones de Office 2019.

1.1 Establecer las referencias a las bibliotecas de objetos

Para comenzar a trabajar con los objetos de Microsoft Access, crearemos una referencia a la biblioteca de Microsoft Access 16.0 (para versiones anteriores de Access también hay versiones anteriores de la biblioteca, 15.0 14.0 12.0, etc.):

1. Crea un nuevo libro de trabajo y guárdalo como C:\Archivos Manual VBA\ Capítulo 14 – Trabajar con Access.xlsm.
2. Activa el editor de VBA y haz clic en **Herramientas – Referencias** para abrir el cuadro de diálogo **Referencias**. Este cuadro de diálogo muestra una lista de todas las bibliotecas disponibles en tu equipo basadas en las aplicaciones que has instalado.
3. Localiza la biblioteca de objetos de Microsoft Access 16.0 en la lista y selecciona su casilla de verificación.
4. Cierra el cuadro de diálogo **Referencias**.

   Una vez que hayas creado la referencia, puedes usar el **Examinador de objetos** para ver una lista de los objetos, propiedades y métodos de la aplicación.
5. Utiliza el cuadro de diálogo **Referencias** para configurar las referencias a otras bibliotecas de objetos a las que se accederá en los ejercicios de este capítulo. Encontrarás la lista de bibliotecas al principio del capítulo. Puedes omitir la configuración de la referencia a la biblioteca JRO, ya que no se utilizará aquí.

2 Conectar con Access

En los ejemplos de procedimientos de este capítulo se utilizan diversos métodos para conectarse a Access. Cada método se examina en detalle. Se puede establecer una conexión con Access utilizando uno de los tres métodos siguientes:

- Automatización
- Data Access Object (DAO)
- ActiveX Data Objects (ADO)

Ventajas de crear una referencia a la biblioteca de objetos de Access

Cuando se establece una referencia a la biblioteca de objetos de Microsoft Access obtenemos tenemos las siguientes ventajas:

- Puedes buscar objetos, propiedades y métodos de Microsoft Access en el Examinador de objetos.
- Puedes ejecutar las funciones de Access directamente en tus procedimientos.
- Puedes declarar la variable de objeto de tipo **`Application`** en lugar del tipo de objeto genérico. Declarando la variable de objeto como **`Dim objAccess As Access.Application`** (vinculación temprana) es más rápido que declararla como **`Dim objAccess As Object`** (vinculación tardía).
- Puedes usar las constantes de Microsoft Access en VBA.
- El procedimiento se ejecutará más rápido.

3 Abrir una base de datos Access con Automatización

Cuando trabajamos con Microsoft Access desde Excel (u otra aplicación) utilizando la automatización se deben seguir los siguientes pasos:

1. Establecer una referencia a la biblioteca de objetos de Microsoft Access 16.0 (ver sección anterior en este capítulo).
2. Declarar una referencia de objeto para representar al objeto **`Application`** de Access:
   Dim **`objAccess`** As **`Access.Application`**
   En esta declaración, **`objAccess`** es el nombre de la variable de objeto y **`Access.Application`** califica la variable de objeto con el nombre de la biblioteca de objetos.
   Devolver la referencia al objeto **`Application`** utilizando la función **`CreateObject`**, la función **`GetObject`** o la palabra clave **`New`** como se muestra a continuación. Observa que debemos asignar la referencia a la variable de objeto mediante la palabra **`Set`**.
   - Se utiliza la función **`CreateObject`** para devolver una referencia a la aplicación cuando no hay una instancia actual del objeto. Si Access ya está funcionando, se inicia una nueva instancia y se crea el objeto especificado.

```
Dim objAccess As Object
Set objAccess = CreateObject("Access.Application.16")
```

   - Se utiliza la función **`GetObject`** para devolver una referencia al objeto **`Application`**, para utilizar la instancia actual de Microsoft Access o para iniciar la aplicación y hacer que cargue un archivo.

```
Dim objAccess As Object
Set objAccess = GetObject(,"Access.Application.16")
```

- Se utiliza la palabra **New** para devolver una referencia al objeto **Application** y asignar la referencia a la variable de objeto, todo en un solo paso.

```
Dim objAccess As New Access.Application
```

También es posible declarar una variable de objeto usando el método de dos pasos que da más control sobre el objeto:

```
Dim objAccess As Access.Application
Set objAccess = New Access.Application
```

Argumentos de la función GetObject

El primer argumento, **pathname**, es opcional. Se utiliza cuando quieres trabajar con un objeto en un archivo específico. El segundo argumento, **class** especifica qué aplicación crea el objeto y qué tipo de objeto es. Cuando el primer argumento es opcional y el segundo es obligatorio, debemos colocar una coma en la posición del primer argumento como se muestra a continuación:

```
Dim objAccess As Object
Set objAccess = GetObject(, "Access.Application.14")
```

Como se omite el primer argumento, se devuelve una referencia a la instancia actual de Microsoft Excel.

```
Dim objAccess As Object
Set objAccess = GetObject("C:\VBAExcel2019_ByExample\" & _
"Northwind 2007.accdb")
```

Cuando el primer argumento de la función **GetObject** es el nombre del archivo de la base de datos, se activa o se crea una nueva instancia de Access con la base de datos específica.

La palabra New

- Cuando declaras la variable de objeto con la palabra **New**, Access no se inicia hasta que comienzas a trabajar con esa variable en el código del procedimiento.
- Cuando usas la palabra **New** para declarar la variable del objeto **Application**, se crea automáticamente una nueva instancia de Access y no necesitarás usar la función **CreateObject**.
- Usar la palabra **New** para crear una nueva instancia de la aplicación es más rápido que usar la función **CreateObject**.

A veces es habitual tener instalada más de una versión de Office. Los números de versiones más recientes de Access son los siguientes:

Versión	Referencia
Microsoft Access 2016/2019	Access.Application.16
Microsoft Access 2013	Access.Application.15
Microsoft Access 2010	Access.Application.14
Microsoft Access 2007	Access.Application.12
Microsoft Access 2003	Access.Application.11
Microsoft Access 2002	Access.Application.10
Microsoft Access 2000	Access.Application.9
Microsoft Access 97	Access.Application.8
Microsoft Access 95	Access.Application.7

Una vez creada una nueva instancia de `Application` utilizando uno de los métodos anteriores, podemos abrir una base de datos o crear una nueva con la ayuda de `OpenCurrentDatabase`. Podemos cerrar la base de datos de Access que abrimos con la automatización utilizando el método `CloseCurrentDatabase`.

Ahora que sabemos cómo crear una variable de objeto que representa al objeto `Application`, veamos un procedimiento de ejemplo que abre una base de datos Access directamente desde un procedimiento de Excel.

1. Antes de comenzar, asegúrate de que en la carpeta C:\Archivos Manual VBA existe una base de datos llamada Northwind 2007.accdb.
2. Accede al editor de VBA del archivo Capítulo 14 – Trabajar con Access.xlsm y cambia el nombre del proyecto VBA a `AccessDesdeExcel`.
3. Inserta un nuevo módulo en el proyecto y llámalo Automatización. A continuación introduce el siguiente procedimiento en la ventana **Código**:

```
Sub AutomatizacionAccess()
    Dim objAccess As Access.Application
    Dim ruta As String

    On Error Resume Next
    Set objAccess = GetObject(, " Access.Application.16")
    If objAccess Is Nothing Then
        ' Obtiene una referencia al objeto Application de Access
        Set objAccess = New Access.Application
    End If
```

```
    ruta = "C:\Archivos Manual VBA\Northwind 2007.accdb"
    ' Abre la tabla Employees de la base de datos Northwind
    With objAccess
        .OpenCurrentDatabase ruta
        .DoCmd.OpenTable "Employees", acViewNormal, acReadOnly
        If MsgBox("¿Quieres hacer visible la " & vbCrLf _
            & "aplicación Access?", vbYesNo, _
            "Mostrar Access") = vbYes Then
            .Visible = True
            MsgBox "Observa que el icono de la aplicación " _
            & "de Access aparece en la barra de tareas"
        End If
        ' Cierra la base de datos y Access
        .CloseCurrentDatabase
        .Quit
    End With
    Set objAccess = Nothing
End Sub
```

Este procedimiento utiliza la instancia actual de Access si está disponible. Si Access no se está ejecutando, se producirá un error en tiempo de ejecución y la variable de objeto se establecerá en `Nothing`. Colocando la instrucción `On Error Resume Next` dentro del procedimiento puedes atrapar el error. Por lo tanto, si Access no se está ejecutando, se iniciará una nueva instancia. Este ejemplo en particular utiliza la palabra `New` para iniciar una nueva instancia de Access.

Como se ha mencionado anteriormente, en lugar de crear una nueva instancia de objeto con la palabra `New`, puedes utilizar la función `CreateObject` para iniciar una nueva instancia de un servidor de automatización como se muestra a continuación:

```
Set objAccess = GetObject(, "Access.Application.16")
If objAccess Is Nothing Then
Set objAccess = CreateObject(, "Access.Application.16")
End If
```

Una vez que se abre Access y se carga la base de datos Northwind con el método `OpenCurrentDatabase`, se crea un comando para abrir la tabla *Employees* en modo de solo lectura. El procedimiento entonces pregunta al usuario si debe hacer visible la ventana de la aplicación Access. Si el usuario hace clic en Sí, la propiedad `Visible` del objeto `Application` se establece en True y se pide que busques el icono en la barra de tareas. Después de hacer clic en **Aceptar**, la base de datos se cierra con el método `CloseCurrentDatabase` y el objeto `Application` de Access se cierra con el método `Quit`.

Después de cerrar el objeto, el valor de la variable de objeto se establece en `Nothing` para liberar los recursos de memoria utilizados. Se puede evitar que se cierre una instancia de Access haciendo que una variable de objeto sea una variable a nivel de módulo en lugar de declararla a nivel de procedimiento. En este caso, la conexión a la base de datos permanecerá abierta hasta que cierres la aplicación controladora (Excel) o utilices el método `Quit` en el código.

4. Ejecuta el procedimiento anterior con la tecla **F8** (paso a paso por instrucciones). Tanto Excel como Access deshabilitan automáticamente todo el contenido potencialmente dañino mostrando una advertencia debajo de la cinta de opciones. Para que Access sepa que confías en la base de datos, haz clic en el botón **Habilitar contenido**.

Abrir una base de datos con contraseña

Si la base de datos Access está protegida con una contraseña, se le pedirá al usuario que la introduzca. Debes utilizar DAO o ADO para abrirla mediante programación. El siguiente ejemplo utiliza la propiedad `DBEngine` del objeto Microsoft Access para especificar la contraseña de la base de datos. Para que este procedimiento funcione, debes establecer una referencia al DAO 3.5 o 3.6 de Microsoft como se explica al principio del capítulo. También debes sustituir el nombre del archivo de la base de datos por el que se encuentra protegido. En este ejemplo se ha utilizado la contraseña “test".

```
Sub BBDDPass()
    Static objAccess As Access.Application
    Dim db As DAO.Database
    Dim strDb As String

    strDb = "C:\Archivos Manual VBA\BBDD Pass.mdb"
    Set objAccess = New Access.Application
    Set db = objAccess.DBEngine.OpenDatabase(Name:=strDb, _
    Options:=False, _
    ReadOnly:=False, _
    Connect:=";PWD=test")
    With objAccess
        .Visible = True
        .OpenCurrentDatabase strDb
    End With
    db.Close
    Set db = Nothing
End Sub
```

3.1 Usar DAO para conectar con Access

Para conectarnos a una base de datos Access mediante DAO, primero debemos configurar una biblioteca de objetos de Microsoft DAO 3.6 en el cuadro de diálogo **Referencias** (ver sección anterior en este capítulo). El procedimiento de ejemplo que se muestra a continuación utiliza el método `OpenDatabase` del objeto `DBEngine` para abrir la base de datos Northwind y luego procede a leer los nombres de las tablas:

1. Inserta un nuevo módulo en el proyecto y llámalo `Ejemplos_DAO`
2. Introduce el siguiente procedimiento:

```
Sub DAO_AbrirBBDD(dbRuta As String)
    Dim db As DAO.Database
    Dim tbl As Variant

    Set db = DBEngine.OpenDatabase(dbRuta)
    MsgBox "Hay " & db.TableDefs.Count & _
    " tablas en " & dbRuta & "." & vbCrLf & _
    " Ver los nombres en la ventana Inmediato."
    For Each tbl In db.TableDefs
        Debug.Print tbl.Name
    Next
    db.Close
    Set db = Nothing
    MsgBox "La base de datos se ha cerrado."
End Sub
```

El objeto `DBEngine` te permite iniciar el motor de base de datos estándar de Access (conocido como (Jet/ACE) y abrir un archivo de base de datos. Puedes abrir un archivo en el formato .accdb o en un formato más antiguo (.mdb). Una vez que la base de datos está abierta, el procedimiento recupera el número total de tablas de la colección `TableDefs`. Una colección `TableDefs` contiene todos los objetos `TableDef` almacenados en una base de datos Access. A continuación, el procedimiento recorre la colección `TableDefs` leyendo los nombres de las tablas e imprimiéndolas en la ventana **Inmediato**. Todas estas operaciones se realizan de forma oculta; fíjate que la ventana de Access no está visible en ningún momento. Finalmente, el procedimiento utiliza el método `Close` para cerrar el archivo.

3. Para ejecutar el procedimiento, escribe cualquiera de las siguientes declaraciones en la ventana **Inmediato** y presiona **Intro**:

```
DAO_AbrirBBDD "C:\Archivos Manual VBA\Northwind 2007.accdb"
DAO_AbrirBBDD "C:\Archivos Manual VBA\Northwind.mdb"
```

   Observa que cuando el procedimiento termina, la ventana **Inmediato** contiene la lista de todas las tablas de la base de datos.

3.2 Usar ADO para conectar con Access

Otro método para establecer una conexión con Access es ADO. Debemos comenzar por establecer una referencia a la biblioteca de Microsoft ActiveX Data Objects 6.1 o una versión inferior. El siguiente procedimiento de ejemplo se conecta a la base de datos Northwind mediante el objeto **Connection**.

1. Inserta un nuevo módulo y llámalo **EjemplosADO**.
2. A continuación, introduce el siguiente procedimiento:

```
Sub ADO_AbrirBBDD(dbRuta)
    Dim con As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim fld As ADODB.Field
    Dim iCol As Integer
    Dim wks As Worksheet

    ' Conecta con la base de datos
    If Right(dbRuta, 3) = "mdb" Then
        con.Open _
        "Provider=Microsoft.Jet.OLEDB.4.0;" _
        & "Data Source=" & dbRuta
    ElseIf Right(dbRuta, 5) = "accdb" Then
        con.Open _
        "Provider = Microsoft.ACE.OLEDB.12.0;" _
        & "Data Source=" & dbRuta
    Else
        MsgBox "Extensión del archivo incorrecta"
        Exit Sub
    End If
    ' Abre el conjunto de registros desde una consulta SQL
    rst.Open "SELECT * FROM Employees " & _
    "WHERE City = 'Redmond'", con, _
    adOpenForwardOnly, adLockReadOnly
    ' Introduce el resultado de la consulta en un libro nuevo
    Workbooks.Add
    Set wks = ActiveWorkbook.Sheets(1)
    wks.Activate
    ' Escribe los nombres de las columnas en la primera fila
    For iCol = 0 To rst.Fields.Count - 1
        wks.Cells(1, iCol + 1).Value = rst.Fields(iCol).Name
    Next
    ' Copia los registros en la hoja
```

```
    wks.Range("A2").CopyFromRecordset rst
    'Autoajusta el ancho de las columnas
    wks.Columns.AutoFit
    ' Vacía las variables de objeto
    Set wks = Nothing
    ' Cierra el conjunto de registros y la conexión con Access
    rst.Close
    con.Close
    ' Elimina las variables de objetos para recuperar
    ' los recursos
    Set rst = Nothing
    Set con = Nothing
End Sub
```

En el procedimiento anterior se abre la base de datos Access con el método `Open`. El método `Open` requiere un argumento de cadena de conexión que contiene el nombre del proveedor de datos. El proveedor **Microsoft.Jet.OLEDB.4.0** se utiliza para las bases de datos Access en formato .mdb y **Microsoft.Jet.OLEDB.12.0** para las bases de datos .accdb. El nombre de la fuente de datos es el nombre completo de la ruta hasta el archivo que deseas abrir.

Después de establecer la conexión con la base de datos Northwind, puedes utilizar el objeto `Recordset` para acceder a sus datos. Los objetos `Recordset` se utilizan para manipular los datos a nivel de registro. El objeto `Recordset` está compuesto por registros (filas) y campos (columnas). Para obtener un conjunto de registros es necesario utilizar el método `Open` y especificar algo de información. La fuente de los registros puede ser el nombre de una tabla de la base de datos, una consulta o declaración SQL que devuelva esos registros. Después de especificar la fuente de registros debes indicar la conexión con la base de datos (`con`) y dos constantes, una de las cuales define el tipo de cursor (`adLockForwardOnly`) y la otra el tipo de bloqueo (`adReadOnly`). La constante `adOpenForwardOnly` le indica a VBA que cree el conjunto de registros y que se desplace hacia adelante en el conjunto de registros devueltos.

La segunda constante, `adLockReadOnly`, especifica el tipo de bloqueo utilizado en los registros durante la edición. Durante la edición los registros de solo lectura por lo que no se podrán alterar los datos. A continuación, el procedimiento recorre todo el conjunto de registros y su colección de campos para imprimir el contenido en una hoja de cálculo de Excel. Después de obtener los datos, el método `Close` cierra el conjunto de registro y otro método `Close` cierra la conexión con la base de datos.

3. Para ejecutar el procedimiento escribe cualquiera de las siguientes instrucciones en la ventana **Inmediato** y presiona **Intro**:

```
ADO_AbrirBBDD "C:\Archivos Manual VBA\Northwind 2007.accdb"
ADO_AbrirBBDD "C:\Archivos Manual VBA\Northwind.mdb"
```

4 Algunas tareas de Access desde Excel

Tras conectarnos a Microsoft Access desde Excel podemos realizar diferentes taras dentro de Access. Esta sección muestra cómo utilizar el código VBA para:

- Crear una nueva base de datos Access.
- Abrir un formulario de una base de datos existente.
- Crear un nuevo formulario en una base de datos.
- Abrir un informe de la base de datos.
- Ejecutar una función de Access.

4.1 Crear una nueva base de datos con DAO

Si deseamos transferir datos de Excel a una nueva base de datos de Access mediante programación, es posible que tengamos que crear la base de datos desde cero. El siguiente procedimiento muestra cómo se hace esto utilizando DAO:

1. En la ventana **Código** del módulo `EjemplosDAO` introduce el siguiente procedimiento:

```
Sub DAO_NuevaBBDD()
    Dim db As DAO.Database
    Dim tbl As DAO.TableDef
    Dim strDb As String
    Dim strTbl As String

  '  On Error GoTo Error_DAO_CrearBBDD
    strDb = "C:\Archivos Manual VBA\DatosExcel.mdb"
    strTbl = "tblStates"
    ' Crea una nueva base de datos llamada DatosExcel
    Set db = CreateDatabase(strDb, dbLangGeneral)
    ' Crea una nueva tabla llamada tblStates
    Set tbl = db.CreateTableDef(strTbl)
    ' Crea los campso y los agrega a la colección de campos
    With tbl
    .Fields.Append .CreateField("StateID", dbText, 2)
    .Fields.Append .CreateField("StateName", dbText, 25)
    .Fields.Append .CreateField("StateCapital", dbText, 25)
    End With
    ' Agrega el objeto tbl a la colección TableDefs
    db.TableDefs.Append tbl
    ' Cierra la base de datos
    db.Close
    Set db = Nothing
    MsgBox "Hay una nueva base de datos en tu disco duro. " _
```

```
        & Chr(13) & "La base de datos contiene una tabla " _
        & "llamada " & strTbl & "."
Exit_DAO_CrearBBDD:
    Exit Sub
Error_DAO_CrearBBDD:
    If Err.Number = 3204 Then
    ' Elimina la base de datos si ya existe
    Kill strDb
    Resume
    Else
    MsgBox Err.Number & ": " & Err.Description
    Resume Exit_DAO_CrearBBDD
    End If
End Sub
```

El método **CreateDatabase** se utiliza para crear la nueva base de datos DatosExcel.mdb. El método **CreateTableDef** del objeto **Database** se utiliza entonces para crear la tabla llamada **TblStates**.

Antes de que se pueda añadir una tabla nueva a una base de datos, se deben crear y añadir los campos a dicha tabla. El procedimiento crea tres campos de texto (**dbText**) que pueden almacenar 2, 25 y 25 caracteres respectivamente. A medida que se crea cada campo, se agregan a la colección de campos del objeto **TableDef** utilizando el método **Append**.

Una vez que se han creado los campos y añadido a la tabla, la propia tabla se añade a la base de datos con el método **Append**. Como es posible que el archivo de base de datos ya exista en la ubicación especificada, el procedimiento incluye la correspondiente rutina de tratamiento de errores que eliminará el archivo existente para que el proceso de creación de la base de datos pueda continuar. Dado que podrían producirse otros errores, la cláusula **Else** incluye declaraciones que mostrarán el error y su descripción y permitirán salir del procedimiento.

2. Ejecuta el procedimiento **DAO_NuevaBBDD**.
3. Abre el archivo DatosExcel.mdb. A continuación, abre la tabla llamada tblStates.
4. Cierra la aplicación de Access.

4.2 Abrir un formulario de Access

Es posible abrir un formulario de Access desde Excel. También podemos crear un formulario nuevo. El siguiente ejemplo utiliza la automatización para conectarse a Access:

1. Inserta un módulo nuevo y llámalo **Formularios**.
2. En la ventana **Código** introduce la siguiente declaración de variable a nivel de módulo y, a continuación, el procedimiento:

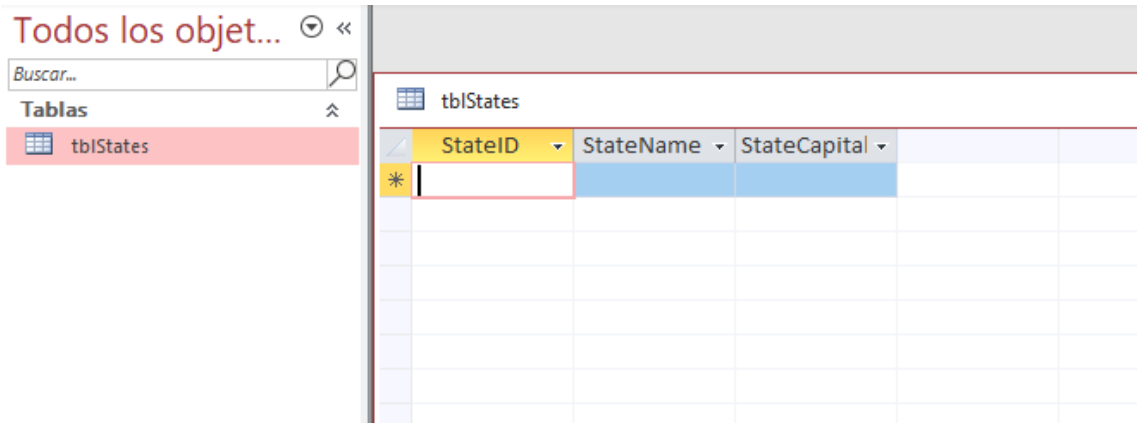


Ilustración 1 Esta base de datos se ha creado mediante un procedimiento desde Excel.

```
Dim objAccess As Access.Application

Sub FormularioAccess()
    Dim strDb As String
    Dim strFrm As String

    strDb = "C:\Archivos Manual VBA\Northwind.mdb"
    strFrm = "Customers"
    Set objAccess = New Access.Application
    With objAccess
    .OpenCurrentDatabase strDb
    .DoCmd.OpenForm strFrm, acNormal
    .DoCmd.Restore
    .Visible = True
    End With
End Sub
```

En el procedimiento anterior se utiliza el método `OpenCurrentDatabase` para abrir la base de datos Northwind. El formulario Customers se abre en vista normal (`acNormal`) con el método `OpenForm` del objeto `DoCmd`. Para mostrar el formulario en la vista de diseño se utiliza la constante `acDesign`. El método `Restore` del objeto `DoCmd` asegura que el formulario se muestra en la pantalla de una ventana y no se minimiza. La propiedad `Visible` del objeto `Application` (`objAccess`) debe ser igual a True para que el formulario sea visible.

Observa que la variable de objeto `Access.Application` (`objAcces`) se declara en la parte superior del módulo. Para que este procedimiento funcione correctamente debes establecer una referencia a la biblioteca de objetos de Microsoft Access.

3. Cambia a la ventana de Excel y presiona **Alt + F8** para mostrar el cuadro de diálogo **Macro**. Resalta el nombre de la macro `FormularioAccess` y haz clic en **Ejecutar**. La Imagen 4.1 muestra el formulario Customers después de haberlo abierto.

4.3 Abrir un informe de Access

El siguiente procedimiento muestra cómo ejecutar un informe de Access directamente en Excel:

1. Inserta un módulo nuevo en el proyecto **AccessDesdeExcel** que se llame **Informes**.
2. Introduce el siguiente procedimiento:

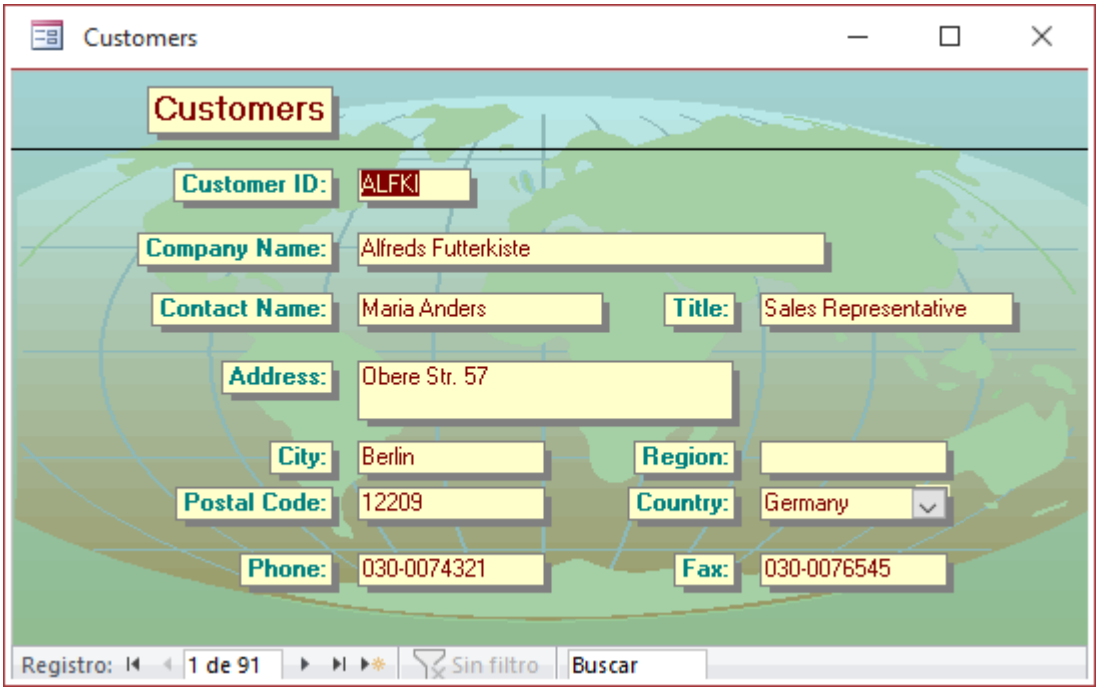


Imagen 4.1 Podemos abrir un formulario de Microsoft Access usando un procedimiento de VBA Excel

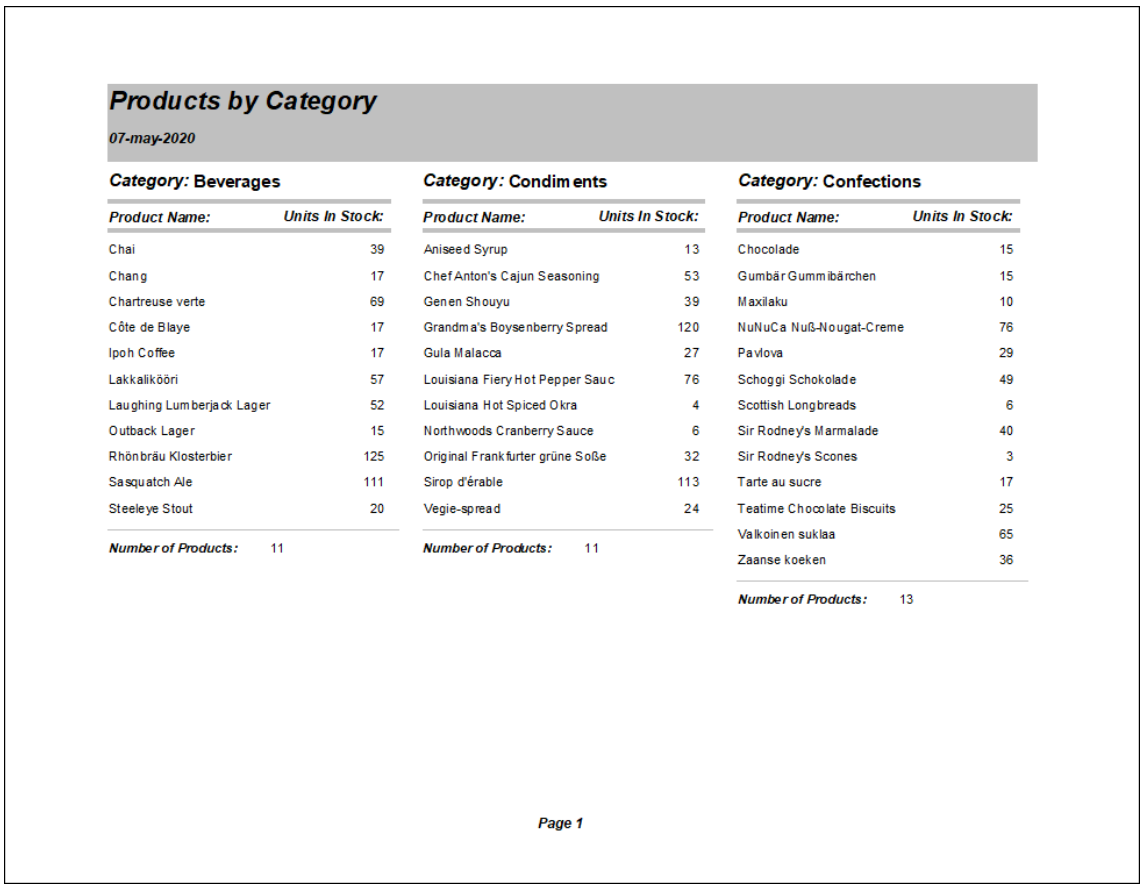
```
Sub InformeAccess()
    Dim strDb As String
    Dim strRpt As String
    strDb = "C:\Archivos Manual VBA\Northwind.mdb"
    strRpt = "Products by Category"
    Set objAccess = New Access.Application
    With objAccess
    .OpenCurrentDatabase (strDb)
    .DoCmd.OpenReport strRpt, acViewPreview
    .DoCmd.Maximize
    .Visible = True
    End With
End Sub
```

En este procedimiento se utiliza el método **OpenCurrentDatabase** para abrir la base de datos Northwind. El informe de productos por categorías se abre en modo vista previa con el método **OpenReport** del objeto **DoCmd**. El método **Maximiza** del objeto

`DoCmd` asegura que el formulario se muestre en la pantalla de una ventana de tamaño completo.

Observa que la variable de objeto `Access.Application` (`objAccess`) se declara en la parte superior del módulo. Para que este procedimiento funcione correctamente debes establecer una referencia a la biblioteca de objetos de Microsoft Access.

3. Cambia a la ventana de Excel y presiona **Alt + F8** para mostrar el cuadro de diálogo **Macro**. A continuación selecciona la macro `InformeAccess` y pulsa **Ejecutar**. La Imagen 4.2 muestra el informe que aparecerá en la pantalla.



Products by Category

07-may-2020

Category: Beverages

Product Name:	Units In Stock:
Chai	39
Chang	17
Chartreuse verte	69
Côte de Blaye	17
Ipoh Coffee	17
Lakkalikööri	57
Laughing Lumberjack Lager	52
Outback Lager	15
Rhönbräu Klosterbier	125
Sasquatch Ale	111
Steeleye Stout	20

Number of Products: 11

Category: Condiments

Product Name:	Units In Stock:
Aniseed Syrup	13
Chef Anton's Cajun Seasoning	53
Genen Shouyu	39
Grandma's Boysenberry Spread	120
Gula Malacca	27
Louisiana Fiery Hot Pepper Sauc	76
Louisiana Hot Spiced Okra	4
Northwoods Cranberry Sauce	6
Original Frankfurter grüne Soße	32
Sirop d'érable	113
Vegie-spread	24

Number of Products: 11

Category: Confections

Product Name:	Units In Stock:
Chocolade	15
Gumbär Gummibärchen	15
Maxilaku	10
NuNuCa Nuß-Nougat-Creme	76
Pavlova	29
Schoggi Schokolade	49
Scottish Longbreads	6
Sir Rodney's Marmalade	40
Sir Rodney's Scones	3
Tarte au sucre	17
Teatime Chocolate Biscuits	25
Valkoinen suklaa	65
Zaanse koeken	36

Number of Products: 13

Page 1

Imagen 4.2 Es posible abrir un informe de Access con un procedimiento de VBA Excel.

4. Cierra y el informe y sal de Access.

El procedimiento que se muestra a continuación es más versátil, ya que permite mostrar cualquier informe de Access en cualquier base de datos de Access. Observa que este procedimiento toma dos argumentos de cadena de texto: el nombre de la base de datos y el nombre del informe.
Asegúrate de que la primera línea siempre aparezca en la parte superior del módulo:

```
Dim objAccess As Access.Application
Sub InformeAccess2(strDb As String, strRpt As String)
    Set objAccess = New Access.Application
    With objAccess
```

```
        .OpenCurrentDatabase (strDb)
        .DoCmd.OpenReport strRpt, acViewPreview
        .DoCmd.Maximize
        .Visible = True
        End With
    End Sub
```

Puedes ejecutar el procedimiento `InformeAccess2` desde la ventana **Inmediato** mediante una subrutina, como se muestra a continuación:

- Desde la ventana **Inmediato** puedes introducir la siguiente instrucción:

```
Call InformeAccess2("C:\Archivos Manual VBA\Northwind.mdb",
"Invoice")
```

- También puedes llamar al procedimiento desde otra macro:

```
Sub MostrarInforme()
    Dim strDb As String
    Dim strRpt As String

    strDb = InputBox("Introduce el nombre de " & _
    "la base de datos (ruta completa): ")
    strRpt = InputBox("Introduce el nombre del " & _
    "Informe:")
    Call InformeAccess2(strDb, strRpt)
End Sub
```

4.4 Crear una base de datos con ADO

Anteriormente creamos una base de datos llamada DatosExcel usando DAO. Crear una base de datos desde un procedimiento VBA también es posible e igual de fácil utilizando ADO.

Todo lo que necesitas es el objeto `Catalog` y la biblioteca de objetos ADOX con su método `Create`. El objeto `Catalog` representa la base de datos completa. Este objeto contiene los elementos de la base de datos como tablas, campos, índices vistas y procedimientos almacenados.

A continuación crearemos una base de datos llamada DatosExcel2.accdb.

1. En el editor de VBA haz clic en el menú **Herramientas – Referencias**. Selecciona la biblioteca **Microsoft ADO Ext. 6.0 for DDL and Security Library** en caso de que no esté activada.
2. Acepta el cuadro de diálogo **Referencias**.
3. Haz clic en el módulo `EjemplosADO` e introduce el siguiente procedimiento en la ventana **Código**:

```
Sub ADO_NuevaBBDD()
    Dim cat As ADOX.Catalog
```

```
    Set cat = New ADOX.Catalog
    cat.Create "Provider=Microsoft.ACE.OLEDB.12.0;" & _
    "Data Source=C:\Archivos Manual VBA\DatosExcel2.accdb;"
    Set cat = Nothing
End Sub
```

El procedimiento anterior utiliza el método `Create` del objeto `Catalog` de ADOX para crear una nueva base de datos Access. Observa que la versión es Access 2019. Para crear un archivo de Access en formato .mdb, asegúrate de cambiar el nombre del proveedor y el del archivo.

4. Ejecuta el procedimiento y navega con el Explorador de Windows hasta la carpeta "Archivo Manual VBA" para comprobar que el archivo ha sido creado.

4.5 Ejecutar consultas Select de Access

Las consultas que más se suelen utilizar para extraer subconjuntos de datos de una o varias tablas son las consultas de selección y de parámetros (Select). Estas consultas se pueden ejecutar fácilmente desde un procedimiento VBA de Excel. Para colocar los datos devueltos por la consulta en una hoja de Excel, utilizamos el método `CopyFromRecordset` del objeto `Range`. Trabajaremos con un procedimiento que ejecute una consulta de selección de Access:

1. Inserta un nuevo módulo en el proyecto y llámalo `Consultas`.
2. Asegúrate de que la biblioteca Microsoft ADO Ext. 6.0 for DDL and Security se encuentra activa en el cuadro de diálogo **Referencias**.
3. En la ventana **Código** del módulo `Consultas` introduce el siguiente procedimiento:

```
Sub ConsultaAccess(strQryName As String)
    Dim cat As ADOX.Catalog
    Dim cmd As ADODB.Command
    Dim rst As ADODB.Recordset
    Dim i As Integer
    Dim strPath As String

    strPath = "C:\Archivos Manual VBA\Northwind.mdb"
    Set cat = New ADOX.Catalog
    cat.ActiveConnection = _
    "Provider=Microsoft.Jet.OLEDB.4.0;" & _
    "Data Source=" & strPath
    Set cmd = cat.Views(strQryName).Command
    Set rst = cmd.Execute
    Sheets.Add
    For i = 0 To rst.Fields.Count - 1
    Cells(1, i + 1).Value = rst.Fields(i).Name
    Next
    With ActiveSheet
```

```
    .Range("A2").CopyFromRecordset rst
    .Range(Cells(1, 1), _
    Cells(1, rst.Fields.Count)).Font.Bold = True
    .Range("A1").Select
    End With
    Selection.CurrentRegion.Columns.AutoFit
    rst.Close
    Set cmd = Nothing
    Set cat = Nothing
End Sub
```

El procedimiento comienza creando una variable de objeto que apunta al objeto **Catalog**. A continuación, la propiedad **ActiveConnection** del objeto **Catalog** define el método para establecer la conexión con la base de datos:

```
Set cat = New ADOX.Catalog
cat.ActiveConnection = "Provider=Microsoft.Jet.OLEDB.4.0;" & _
      "Data Source=" & strPath
```

El objeto **Command** de la biblioteca de objetos ADODB especifica el comando que quieres ejecutar para obtener datos de la fuente de datos. Este procedimiento intenta acceder a una consulta específica en una base de datos cuyo nombre se proporcionará en tiempo de ejecución.

```
Set cmd = cat.Views(strQryName).Command
```

La colección **Views**, que forma parte de la biblioteca de objetos ADOX, contiene todos los objetos **View** de un catálogo específico. Una vista es un conjunto filtrado de registros o una tabla virtual creada a partir de otras tablas o vistas.

Después de obtener acceso a la consulta requerida en la base de datos, se puede ejecutar la consulta de la siguiente manera:

```
Set rst = cmd.Execute
```

El método **Execute** del objeto **Command** permite activar una consulta específica, una instrucción SQL o un procedimiento almacenado. El conjunto de registros devuelto se asigna entonces a la variable de objeto del tipo **Recordset** mediante la palabra **Set**. Después de crear el conjunto de registros, éstos se colocan en una hoja de trabajo de Excel utilizando el método **CopyFromRecordset** (este método se trata más adelante).

4. Para ejecutar el procedimiento anterior, escribe la siguiente declaración en la ventana **Inmediato** y presiona **Intro**:

```
ConsultaAccess("Current Product List")
```

5. Dirígete a la ventana de Excel y observa los resultados obtenidos (ver Imagen 4.3).

	A	B
1	**Product ID**	**Product Name**
2	3	Aniseed Syrup
3	40	Boston Crab Meat
4	60	Camembert Pierrot
5	18	Carnarvon Tigers
6	1	Chai
7	2	Chang
8	39	Chartreuse verte
9	4	Chef Anton's Cajun Seasoning
10	48	Chocolade
11	38	Côte de Blaye
12	58	Escargots de Bourgogne
13	52	Filo Mix
14	71	Fløtemysost
15	33	Geitost
16	15	Genen Shouyu
17	56	Gnocchi di nonna Alice
18	31	Gorgonzola Telino
19	6	Grandma's Boysenberry Spread
20	37	Gravad lax
21	69	Gudbrandsdalsost
22	44	Gula Malacca
23	26	Gumbär Gummibärchen
24	22	Gustaf's Knäckebröd
25	10	Ikura

Imagen 4.3 El resultado de la ejecución de una consulta de Access desde un procedimiento de Excel se coloca en una hoja de trabajo.

4.6 Llamar a una función de Access

Podemos ejecutar una función predeterminada de Access desde Excel a través de automatización. El siguiente procedimiento llama a la función **`EuroConvert`** para convertir 1.000 pesetas españolas a Euros. La función **`EuroConvert`** utiliza las tasas de conversión fijadas por la Unión Europea:

```
Sub FuncionAccess()
    Dim objAccess As Object

    On Error Resume Next
    Set objAccess = GetObject(, "Access.Application")
    ' Si no hay abierta ninguna instancia de Access, se crea una
    If objAccess Is Nothing Then
        Set objAccess = CreateObject("Access.Application")
    End If
    MsgBox "Por 1000 pesetas españolas obtendrás " & _
```

```
        objAccess.EuroConvert(1000, "ESP", "EUR") & _
        " euros. "
        Set objAccess = Nothing
    End Sub
```

5 Introducir datos de Access en una hoja de Excel

Existen muchas formas de llevar datos externos a Excel. Esta sección muestra diferentes técnicas para poner los datos de Microsoft Access en una hoja de trabajo de Excel. Aunque ya hemos visto algunos de estos métodos, a continuación, profundizaremos en ellos con más detalle:

- Usando el método `GetRows`.
- Usando el método `CopyFromRecordset`.
- Usando el método `TransferSpreadsheet`.
- Usando el método `OpenDatabase`
- Creando un archivo de texto.
- Creando una consulta de tabla.
- Creando un gráfico incrustado a partir de los datos de Access.

5.1 Obtener datos con el método GetRows

Para introducir o colocar datos de Access en una hoja de Excel, podemos utilizar el método `GetRows`. Este método devuelve una matriz bidimensional en la que el primer subíndice es el número que representa el campo, y el segundo es el número que representa el registro. La numeración de los registros y campos comienza por 0.

El siguiente ejemplo muestra cómo utilizar el método `GetRows` en un procedimiento VBA. Ejecutaremos la consulta “Facturas” en la base de datos Northwind y mostraremos los registros en una hoja de cálculo.

1. Inserta un módulo nuevo y llámalo `Metodo_GetRows`.
2. Activa la biblioteca Microsoft DAO 3.6 Object Library en caso de que no se encuentre activa desde el cuadro de diálogo **Referencias**.
3. Haz clic en **Aceptar** para cerrar el cuadro.
4. Introduce el siguiente procedimiento:

```
Sub ObtenerDatos_GetRows()
    Dim db As DAO.Database
    Dim qdf As DAO.QueryDef
    Dim rst As DAO.Recordset
    Dim recArray As Variant
    Dim i As Integer
    Dim j As Integer
    Dim strPath As String
```

```
    Dim a As Variant
    Dim countR As Long
    Dim strShtName As String

    strPath = "C:\Archivos Manual VBA\Northwind.mdb"
    strShtName = "Returned records"
    Set db = OpenDatabase(strPath)
    Set qdf = db.QueryDefs("Invoices")
    Set rst = qdf.OpenRecordset
    rst.MoveLast
    countR = rst.RecordCount
    a = InputBox("El conjunto contiene  " & _
    countR & " registros." & vbCrLf _
    & "Introduce el número de registros a devolver: ", _
    "Número de registros")
    If a = "" Or a = 0 Then Exit Sub
        If a > countR Then
        a = countR
        MsgBox "El número introducido es muy alto." & vbCrLf _
        & "Se devolverán todos los datos."
        End If
    Workbooks.Add
    ActiveWorkbook.Worksheets(1).Name = strShtName
    rst.MoveFirst
    With Worksheets(strShtName).Range("A1")
        .CurrentRegion.Clear
        recArray = rst.GetRows(a)
        For i = 0 To UBound(recArray, 2)
            For j = 0 To UBound(recArray, 1)
                .Offset(i + 1, j) = recArray(j, i)
            Next j
        Next i
        For j = 0 To rst.Fields.Count - 1
            .Offset(0, j) = rst.Fields(j).Name
            .Offset(0, j).EntireColumn.AutoFit
        Next j
    End With
    db.Close
End Sub
```

Después de abrir la base de datos Access con el método **OpenDatabase**, el procedimiento ejecuta la consulta "Invoices" con la siguiente instrucción:

```
Set qdf = db.QueryDefs("Invoices")
```

En la biblioteca de objetos de Microsoft DAO 3.6, el objeto **QueryDefs** representa una consulta de selección o de acción. Las consultas de selección devuelven datos de una o más tablas, mientras que las consultas de acción permiten añadir, modificar o eliminar registros.

Tras ejecutar la consulta, el procedimiento coloca los registros devueltos por la consulta en la variable de objeto del tipo **Recordset** mediante el método **OpenRecordset**, como se muestra a continuación:

```
Set rst = qdf.OpenRecordset
```

A continuación, el número de registros se recupera mediante el método **RecordCount** y se coloca en la variable **countR**. Observa que, para obtener el número de registros correctos, en primer lugar el puntero del registro debe moverse al último registro del conjunto mediante el método **MoveLast**.

```
rst.MoveLast
countR = rst.RecordCount
```

A continuación, el procedimiento pide al usuario que introduzca el número de registros a devolver a la hoja de cálculo. En este punto, puedes cancelar haciendo clic en el botón Cancelar del cuadro de diálogo o introducir un número de registros. Si introduces un número mayor que el número de registros, el procedimiento recuperará todos los registros.

Antes de recuperar los registros, debes mover el puntero al primer registro utilizando el método **MoveFirst**. Si se te olvida hacerlo, el puntero permanecerá en el último registro y solo se recuperará un registro.

A continuación, el procedimiento activa la hoja de trabajo de registros devueltos y borra la región actual. Los registros se devuelven primero a la variable **Variant** que contiene una matriz bidimensional utilizando el método **GetRows** del objeto **RecordSet**. A continuación, el procedimiento hace un bucle en ambas dimensiones de la matriz para colocar los registros en la hoja de trabajo comenzando en la celda A2. Cuando finaliza, otro bucle rellenará la primera fila de la hoja con los nombres de los campos y ajustará automáticamente cada columna para que los datos se muestren correctamente.

5. Ejecuta el procedimiento. Cuando se indique el número de registros, escribe "10" y haz clic en **Aceptar**. A continuación, dirígete a la ventana de Excel para ver los resultados.

5.2 Obtener datos usando el método CopyFromRecordset

Para obtener un conjunto de registros completo en una hoja de trabajo, utilizaremos el método **CopyFromRecordset** del objeto **Range**. Este método puede tener hasta tres argumentos: **Data**, **MaxRows** y **MaxColumns**. Solo se requiere el primero. Este argumento

puede ser el objeto **Recordset**. Los argumentos opcionales, **MaxRows** y **MaxColumns**, permiten especificar el número de registros y el número de campos que deben devolverse. Si se omite el argumento **MaxRows**, todos los registros devueltos se copiarán en la hoja de cálculo. Si se omite el argumento **MaxColumns**, se recuperarán todos los campos.

El siguiente procedimiento utiliza los objetos ADO y el método **CopyFromRecordset** para recuperar todos los registros de la tabla "Products" de la base de datos Northwind.

1. Inserta un módulo nuevo y llámalo **Metodo_CopyFromRecordset**.
2. Introduce el siguiente procedimiento:

```
Sub ObtenerProductos()
    Dim conn As New ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim strPath As String

    strPath = "C:\Archivos Manual VBA\Northwind.mdb"
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strPath & ";"
    conn.CursorLocation = adUseClient
    ' Crea un Recordset de todos los registros
    ' de la tabla Products
    Set rst = conn.Execute(CommandText:="Products", _
    Options:=adCmdTable)
    rst.MoveFirst
    ' Transfiere los datos a Excel
    ' incluyendo los nombres de los campos
    With Worksheets("Hoja1").Range("A1")
        .CurrentRegion.Clear
        For j = 0 To rst.Fields.Count - 1
            .Offset(0, j) = rst.Fields(j).Name
        Next j
        .Offset(1, 0).CopyFromRecordset rst
        .CurrentRegion.Columns.AutoFit
    End With
    rst.Close
    conn.Close
    Set rst = Nothing
    Set conn = Nothing
End Sub
```

El procedimiento anterior copia todos los registros de la tabla de productos de la base de datos en una hoja de trabajo de Excel. Si deseas copiar menos registros, utiliza el argumento **MaxRows** de la siguiente forma:

```
.Offset(1, 0).CopyFromRecordset rst, 5
```

Esta declaración le dice a VBA que copie solo cinco registros. El método **Offset** hace que los registros se introduzcan en una hoja de cálculo empezando por la segunda fila. Para enviar todos los registros a la hoja utilizando los datos de solo dos filas de la tabla, utiliza la siguiente declaración:

```
.Offset(1, 0).CopyFromRecordset rst, , 2
```

La declaración anterior le dice a VBA que copie todos los datos de las dos primeras columnas. La segunda coma de la instrucción es un marcador de posición para el argumento omitido.

3. Ejecuta el procedimiento **ObtenerProductos** y cambia el foco a la ventana de Excel para ver los resultados.

5.3 Obtener datos con el método TransferSpreadsheet

Otra técnica para obtener datos desde Access la tenemos con el método **TransferSpreadsheet** del objeto **DoCmd** de Access para importar o exportar datos entre la base de datos actual de Access (.mdb) o el proyecto Access (.adp) y un archivo de hoja de cálculo. Utilizando este método también se pueden vincular los datos de una hoja de cálculo Excel a una base de datos Access. Con una hoja de cálculo vinculada, podemos ver y editar los datos de la hoja de cálculo Access y, al mismo tiempo, permitir el acceso completo a los datos desde la Excel. El método **TransferSpreadsheet** tiene la siguiente sintaxis:

```
DoCmd.TransferSpreadsheet [transfertype][, spreadsheettype],
tablename, filename [, hasfieldnames][, range]
```

El argumento **transfertype** puede ser una de las siguientes constantes: **acImport** (valor predeterminado), **acExport** o **acLink**. Estas constantes definen si los datos deben ser importados, exportados o vinculados a la base de datos. El argumento **Spreadsheettype** puede ser una de las constantes que se muestran en la siguiente tabla:

Nombre constante	Valor
acSpreadsheetTypeExcel3 (valor predet.)	0
acSpreadsheetTypeExcel4	6
acSpreadsheetTypeExcel5	5
acSpreadsheetTypeExcel7	5
acSpreadsheetTypeExcel8	8
acSpreadsheetTypeExcel9	8

Nombre constante	Valor
acSpreadsheetTypeExcel12	9
acSpreadsheetTypeExcel12XML	10
acSpreadsheetTypeLotusWK1	2
acSpreadsheetTypeLotusWK3	3
acSpreadsheetTypeLotusWK4	7

No es difícil adivinar que el argumento **spreadsheettype** especifica el nombre de la hoja de cálculo y el número de versión.

El argumento **tablename** es una expresión de texto que especifica el nombre de la tabla de Access a la que deseamos importar datos de la hoja de cálculo o exportarlos. En lugar del nombre de la tabla, también podemos especificar el nombre de la consulta de selección cuyos resultados queremos exportar a la hoja de cálculo.

El argumento **filename** es una expresión de texto que especifica el nombre de archivo y la ruta de la hoja de cálculo de la que deseamos importar, exportar o enlazar datos.

El argumento **hasfieldnames** es un valor lógico. Establecido en True indica que la primera fila de la hoja de cálculo contiene los nombres de los campos. False indica que la primera contiene datos normales. El ajuste predeterminado es False (sin encabezados).

El argumento **rango** es una expresión de texto que especifica el rango de celdas o el nombre del rango en la hoja de cálculo. Este argumento se aplica solo a la importación. Si lo omitimos, se importará toda la hoja de cálculo.

El siguiente procedimiento exporta los datos de la tabla Shippers de la tabla de datos Northwind a la hoja de cálculo Shippers.xls mediante el método **TransferSpreadsheet**.

1. Crea un nuevo módulo llamado **Método_TransferSpreadsheet**.
2. Introduce el siguiente procedimiento:

```
Sub ExportarDatos()
    Dim objAccess As Access.Application

    Set objAccess = CreateObject("Access.Application")
    objAccess.OpenCurrentDatabase filepath:= _
    "C:\Archivos Manual VBA\Northwind 2007.accdb"
    objAccess.DoCmd.TransferSpreadsheet _
    TransferType:=acExport, _
    SpreadsheetType:=acSpreadsheetTypeExcel12, _
    TableName:="Shippers", _
```

```
        Filename:="C:\Archivos Manual VBA\Shippers.xls", _
        HasFieldNames:=True, _
        Range:="Hoja1"
        objAccess.Quit
        Set objAccess = Nothing
    End Sub
```

El procedimiento `ExportarDatos` utiliza la automatización para establecer una conexión con Access. La base de datos se abre usando el método `OpenCurrentDatabase`. El método `TransferSpreadsheet` del objeto `DoCmd` se utiliza para especificar que los datos de la tabla Shippers deben ser exportados a un libro llamado Shippers.xls y colocados en la Hoja1

3. Cambia a la ventana de Excel y selecciona la ficha **Vista > Macros > Ver macros**. En el cuadro de diálogo **Macros** selecciona el procedimiento `ExportarDatos` y haz clic en **Ejecutar**.
4. Abre el archivo Shippers.xls creado por el procedimiento para ver los datos recuperados. Si se muestra un mensaje para verificar que el archivo no está dañado o que proviene de una fuente de confianza, haz clic en **Aceptar**.

5.4 Obtener datos con el método OpenDatabase

El método `OpenDatabase` es la forma más fácil de introducir datos de una base de datos Access en una hoja de Excel. Este método, que se aplica a los libros de trabajo, requiere que se especifique el nombre del archivo de la base de datos que se quiere abrir.

El siguiente procedimiento muestra cómo abrir la base de datos Northwind utilizando el método `OpenDatabase` de la colección `Workbooks`:

```
Sub MetodoOpenDatabase()
    On Error Resume Next
    Workbooks.OpenDatabase _
    Filename:="C:\Archivos Manual Excel\Northwind.mdb"
    Exit Sub
End Sub
```

Al ejecutar el procedimiento anterior, Excel mostrará un cuadro de diálogo con una lista de todas las tablas y consultas de la base de datos. Después de seleccionar una de la lista, se abrirá un nuevo libro de trabajo con los datos de la tabla o consulta seleccionada.

El método `OpenDatabase` tiene cuatro argumentos opcionales (ver la siguiente tabla) que se pueden utilizar para calificar mejor los datos que se desean recuperar:

Argumento	Tipo de dato	Descripción
CommandText	Variant	Cadena de consulta SQL. Ver siguiente ejercicio.

Argumento	Tipo de dato	Descripción
CommandType	Variant	Tipo de comando de la consulta. Puede ser: `xlCmdCube`, `xlCmdList`, `xlCmdSql`, `xlCmdTable` o `xoCmdDefault`
BackgroundQuery	Variant	`True` para que Excel realice consultas de forma asíncrona (sin que se vea). El valor por defecto es `False`.
ImportDataAs	Variant	Especifica el formato de la consulta. Se usa `xlQueryTable` o `xlPivotTableReport` para generar una tabla o una tabla dinámica con los datos recibidos.

Escribamos un procedimiento que cree un informe de tabla dinámica de los registros de clientes:

1. Crea un módulo nuevo y llámalo `Metodo_OpenDatabase`.
2. Introduce el siguiente procedimiento:

```
Sub ClientesPorPais()
    On Error Resume Next
    Workbooks.OpenDatabase _
    Filename:="C:\Archivos Manual VBA\Northwind.mdb", _
    CommandText:="Select * from Customers", _
    CommandType:=xlCmdSql, _
    BackgroundQuery:=True, _
    ImportDataAs:=xlPivotTableReport
    Exit Sub
End Sub
```

3. Activa la ventana de Excel y haz clic en la **ficha Vista > Macros > Ver macros**. A continuación, selecciona el procedimiento `ClientesporPais` y haz clic en **Ejecutar**. Cuando lo hagas, Excel creará un nuevo libro de trabajo y mostrará la ventana con la lista de campos de la tabla dinámica donde se enumeran los campos disponibles en la tabla Customers de Access (ver Imagen 5.1).
4. Arrastra los campos Country y CustomerID de la lista de campos hasta el área de filas. Arrastra de nuevo el campo CustomerID y suéltalo en el área de valores. En la Imagen 5.2 se muestra el informe de tabla dinámica completo.

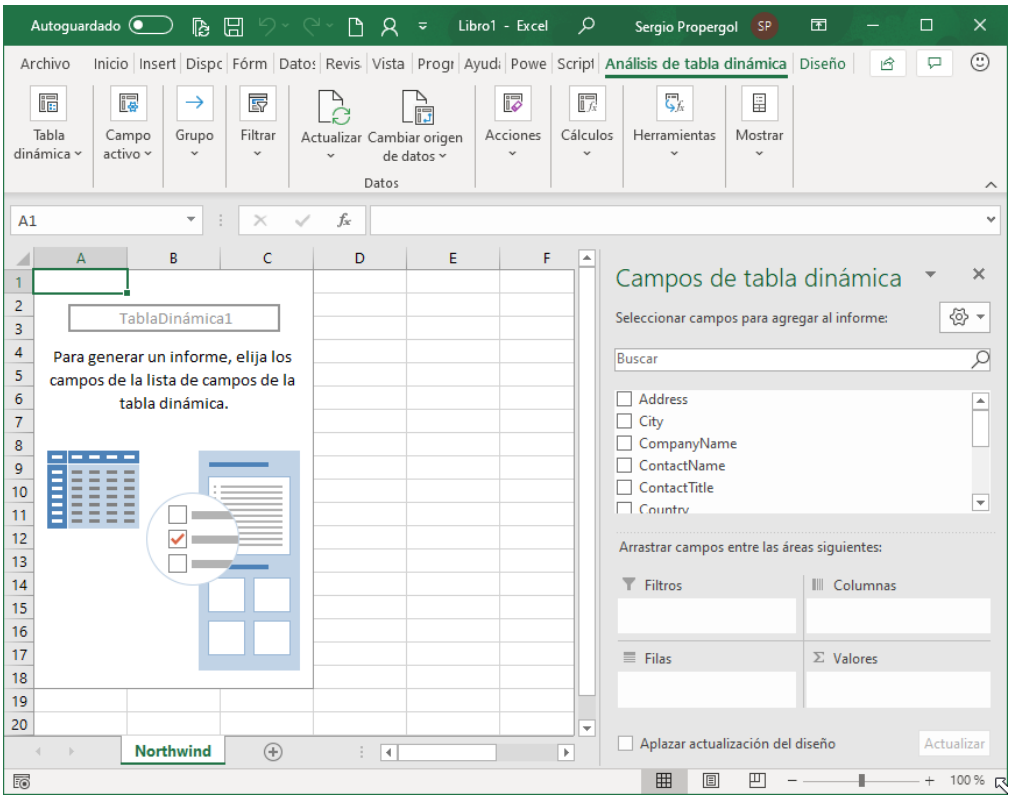


Imagen 5.1 Utilizando los argumentos opcionales del método OpenDatabase, se puede especificar que los datos de la base de datos se recuperen en un formato específico como una tabla o una tabla dinámica.

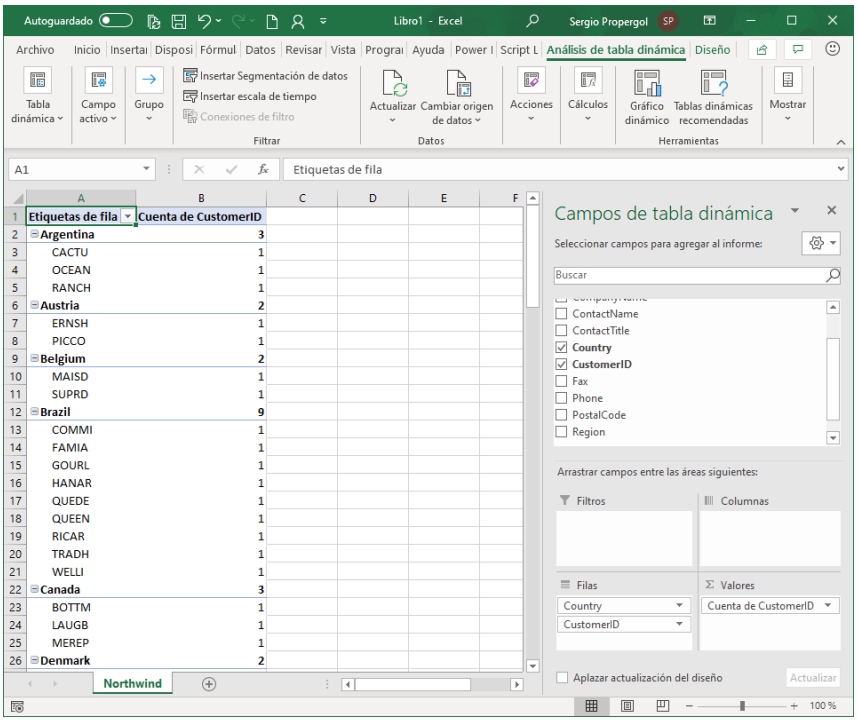


Imagen 5.2 Un informe de tabla dinámica basado en los datos obtenidos de la tabla de clientes de una base de datos Access.

5. Cierra el libro con la tabla dinámica.

5.5 Crear un archivo de texto con datos de Access

Podemos crear un archivo de texto delimitado por comas o tabulaciones con los datos de Access a partir de un procedimiento VBA. Los archivos de texto son especialmente útiles para transferir grandes cantidades de datos a una hoja de cálculo.

El siguiente procedimiento ilustra cómo utilizar el **Recordset** de ADO para crear un archivo de texto delimitado por tabulaciones:

1. Inserta un módulo nuevo en el proyecto **AccessDesdeExcel** y llámalo **ArchivosTexto**
2. En la ventana **Código** introduce el siguiente procedimiento:

> ### Importante
>
> Para que el procedimiento funcione correctamente es necesario crear una referencia a la biblioteca de Microsoft ActiveX Data Objects 6.1 (consulta en una sección anterior de este capítulo las instrucciones sobre cómo cargar una referencia).

```
Sub CrearArchivoTexto()
    Dim strPath As String
    Dim conn As New ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim strData As String
    Dim strHeader As String
    Dim strSQL As String
    Dim fld As Variant

    strPath = "C:\Archivos Manual VBA\Northwind.mdb"
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strPath & ";"
    conn.CursorLocation = adUseClient
    strSQL = "SELECT * FROM Products WHERE UnitPrice > 50"
    Set rst = conn.Execute(CommandText:=strSQL, _
    Options:=adCmdText)
    ' save the recordset as a tab-delimited file
    strData = rst.GetString(StringFormat:=adClipString, _
    ColumnDelimeter:=vbTab, RowDelimeter:=vbCr, _
    nullExpr:=vbNullString)
    For Each fld In rst.Fields
        strHeader = strHeader + fld.Name & vbTab
    Next
    Open "C:\Archivos Manual VBA\Productos más de 50.txt" _
```

```
    For Output As #1
    Print #1, strHeader
    Print #1, strData
    Close #1
    rst.Close
    conn.Close
    Set rst = Nothing
    Set conn = Nothing
End Sub
```

Antes de crear el archivo de texto hay que obtener los datos necesarios de Access utilizando el método **GetString** del objeto **Recordset**. Este método devuelve un conjunto de registros en una cadena y es más rápido que hacer un bucle en el conjunto de registros. El método **GetString** tiene la siguiente sintaxis:

```
variant = recordset.GetString(StringFormat, NumRows, _
ColumnDelimiter, RowDelimiter, NullExpr)
```

El primer argumento (**StringFormat**) determina el formato para representar el conjunto de registros como string. Usa la constante **adClipString** para este argumento. El segundo argumento (**NumRows**) especifica el número de filas del conjunto de registros que se deben devolver. Si está en blanco **GetString** devolverá todas las filas. El tercer argumento (**ColumnDelimiter**) especifica el delimitador de las columnas dentro de la fila (el valor predeterminado es una tabulación (**vbTab**)). El cuarto argumento (**RowDelimiter**) especifica un delimitador de fila (el valor por defecto es un retorno de carro (**vbCr**)). El quinto argumento (**NullExpr**) especifica una expresión para representar valores NULL (vacíos).

Una vez que tenemos todos los datos en una variable string, el procedimiento hace un bucle recorriendo los campos del conjunto de registros para recuperar los nombres de las columnas. El método **GetString** solo puede manejar las solicitudes de los datos, por lo que, si necesitas los datos con los encabezados, debes obtenerlos por separado. Se almacenan los nombres de los campos en una variable string separada.

A continuación, el procedimiento crea un archivo de texto con la siguiente declaración:

```
Open "C:\Archivos Manual VBA\Productos más de 50.txt" _
For Output As #1
```

Ya deberías estar familiarizado con este método de creación de archivos, pues se trató con detalle en el Capítulo 12.

A continuación, utilizamos la declaración **Print** para escribir tanto el encabezado como la cadena de datos en el archivo de texto. Ahora que el archivo tiene los datos podemos cerrarlo con la declaración **Close**.

3. Ejecuta el procedimiento **CrearArchivoTexto**. Este procedimiento crea el archivo Productos más de 50.txt en el directorio indicado.

4. Ahora abre el archivo. En el cuadro de diálogo **Abrir**, selecciona "Todos los archivos (*.*)" en la lista desplegable. A continuación, selecciona el archivo Productos más de 50.txt y haz clic en **Abrir**. En el cuadro de diálogo **Asistente para la importación de texto (Paso 1 de 3)** selecciona la opción **Delimitado**. Haz clic en **Siguiente** para obtener una vista previa de la estructura del archivo **(Paso 2 de 3)**. Haz clic en **Finalizar** para mostrar los datos en una hoja de trabajo.

5.6 Crear una consulta a partir de datos de Access

Si deseamos trabajar en Excel con datos que provienen de fuentes externas y sabemos que estos datos a menudo sufren cambios, podemos crear una tabla de consulta. Una tabla de consulta es una tabla especial en una hoja de Excel, que está conectada a una fuente de datos externa, como bases de datos de Access o SQL Server, una página web o un archivo de texto. Para obtener la información más actualizada, el usuario puede actualizar fácilmente la tabla de consulta. Excel cuenta con una gran herramienta (Power Query) para obtener datos de fuentes de datos externas: simplemente haz clic en la ficha datos y selecciona la fuente de datos adecuada.

Al crear una consulta en la base de datos externa, podemos introducir datos que se ajusten exactamente a nuestras necesidades. Por ejemplo, en lugar de obtener toda la información sobre productos, podemos especificar los criterios que deben cumplir los datos antes de su obtención. De este modo, en lugar de introducir todos los productos de una tabla de Access, podemos recuperar solo los productos con un precio unitario superior a 20 euros / dólares.

En VBA podemos utilizar el objeto `QueryTable` para acceder a los datos externos. Cada objeto `QueryTable` representa una tabla de hoja de cálculo creada a partir de datos devueltos desde una fuente de datos externa. Para crear una consulta mediante programación, utilizamos el método `Add` de la colección de objetos `QueryTables`. Este método requiere tres argumentos que se explican en el siguiente ejercicio práctico:

1. Crea un módulo nuevo y llámalo `Consultas`.
2. Introduce el siguiente procedimiento en la ventana **Código**:

```
Sub CrearTablaConsulta()
    Dim myQryTable As Object
    Dim myDb As String
    Dim strConn As String
    Dim Dest As Range
    Dim strSQL As String

    myDb = "C:\Archivos Manual VBA\Northwind.mdb"
    strConn = "OLEDB;Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & myDb & ";"
    Workbooks.Add
    Set Dest = Worksheets(1).Range("A1")
    Sheets(1).Select
```

```
    strSQL = "SELECT * FROM Products WHERE UnitPrice > 20"
    Set myQryTable = ActiveSheet.QueryTables.Add(strConn, _
    Dest, strSQL)
    With myQryTable
        .RefreshStyle = xlInsertEntireRows
        .Refresh False
    End With
End Sub
```

El procedimiento utiliza la siguiente declaración para crear una tabla de consulta en la hoja activa:

```
Set myQryTable = ActiveSheet.QueryTables.Add(strConn, _
    Dest, strSQL)
```

strConn es una variable que proporciona un valor para el primer argumento del método **QueryTables** (**Connection**). Se trata de un argumento obligatorio de tipo variant que especifica la fuente de datos de la consulta.

Dest es una variable que proporciona un valor para **Destination**. Es un argumento de tipo **Range** que especifica la celda donde se colocará la tabla de consulta resultante.

strSQL es una variable que le da el valor al tercer argumento (SQL). El argumento es obligatorio, de tipo string y define los datos que debe devolver la consulta.

Cuando se crea una consulta con el método **Add**, la consulta no se ejecuta hasta que se llama al método **Refresh**. Este método acepta un argumento: **BackgroundQuery**. Es un argumento opcional de tipo variant que permite determinar si el control debe volver al procedimiento cuando se ha establecido la conexión con la base de datos.

El procedimiento **CrearTablaConsulta** solo obtiene de la tabla de productos de la base de datos aquellos cuyo precio unitario es superior a 20. Observa que el control se devuelve al procedimiento solo después de que todos los registros que cumplen el criterio hayan sido recuperados. El método **RefreshStyle** determina cómo se insertan los datos en la hoja de trabajo. Se pueden utilizar las siguientes constantes:

- **xlOverwriteCells**. Las celdas con contenido se sobrescriben con los datos obtenidos.
- **xlInsertDeleteCells**. Las celdas se insertan o borran para acomodar los datos obtenidos.
- **xlInsertEntireRows**. Se insertan filas para acomodar los datos obtenidos.

3. Ejecuta el procedimiento **CrearTablaConsulta**. Cuando finalice, deberías ver un nuevo libro de Excel. La primera hoja del libro contendrá los datos que especificaste en la consulta.

5.7 Crear un gráfico incrustado desde Access

Desde VBA podemos crear fácilmente un gráfico basado en los datos obtenidos de una base de datos Access. Los gráficos se crean utilizando el método **Add** de la colección **Charts**. Vamos a crear un procedimiento que recoja datos de la base de datos Northwind y cree un gráfico incrustado.

1. Crea un nuevo módulo y llámalo **GráficosAccess**.
2. Introduce el siguiente procedimiento:

```
Sub Graficos_ADO()
    Dim conn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim mySheet As Worksheet
    Dim recArray As Variant
    Dim strQueryName As String
    Dim i As Integer
    Dim j As Integer

    strQueryName = "Category Sales for 1997"
    ' Conecta con la base de datos
    conn.Open _
    "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=C:\Archivos Manual VBA\Northwind.mdb;"
    ' Open Recordset based on the SQL statement
    rst.Open "SELECT * FROM [" & strQueryName & "]", conn, _
    adOpenForwardOnly, adLockReadOnly
    Workbooks.Add
    Set mySheet = Worksheets("Hoja1")
    With mySheet.Range("A1")
        recArray = rst.GetRows()
        For i = 0 To UBound(recArray, 2)
            For j = 0 To UBound(recArray, 1)
                .Offset(i + 1, j) = recArray(j, i)
            Next j
        Next i
        For j = 0 To rst.Fields.Count - 1
            .Offset(0, j) = rst.Fields(j).Name
            .Offset(0, j).EntireColumn.AutoFit
        Next j
    End With
    rst.Close
    conn.Close
```

```
    Set rst = Nothing
    Set conn = Nothing
    mySheet.Activate
    Charts.Add
    ActiveChart.ChartType = xl3DColumnClustered
    ActiveChart.SetSourceData _
    Source:=mySheet.Cells(1, 1).CurrentRegion, _
    PlotBy:=xlRows
    ActiveChart.Location Where:=xlLocationAsObject, _
    Name:=mySheet.Name
    With ActiveChart
        .HasTitle = True
        .ChartTitle.Characters.Text = _
        "Ventas por categoría en 2018"
        .Axes(xlCategory).HasTitle = True
        .Axes(xlCategory).AxisTitle.Characters.Text = ""
        .Axes(xlValue).HasTitle = True
        .Axes(xlValue).AxisTitle. _
        Characters.Text = mySheet.Range("B1") & "($)"
        .Axes(xlValue).AxisTitle.Orientation = xlUpward
    End With
End Sub
```

El procedimiento anterior utiliza objetos de datos ActiveX (ADO) para obtener datos de una consulta de Access. Las primera filas de datos se obtienen utilizando el método **GetRows** del objeto **Recordset**. Como ya sabes por los ejemplos anteriores de este capítulo, el método **GetRows** obtiene los datos en una matriz bidimensional. Una vez que los datos se colocan en una hoja de cálculo de Excel, se añade un gráfico con el método **Add**. El método **SetSourceData** del objeto **Chart** establece el rango de datos de origen para el gráfico de esta forma:

```
ActiveChart.SetSourceData Source:=mySheet.Cells(1, 1). _
CurrentRegion, PlotBy:=xlRows
```

Source es el rango que contiene los datos de la fuente que acabas de colocar en la hoja de trabajo que comienza en la celda A1. **PlotBy** hará que el gráfico incrustado trace los datos por filas.

A continuación, el método **Location** del objeto **Chart** especifica dónde debe colocarse el gráfico. Este método requiere dos argumentos: **Where** y **Name**. El argumento **Where** es obligatorio, pudiendo utilizar una de las siguientes constantes: **xlLocationAsNewSheet**, **xlLocationAsObject** o **xlLocationAutomatic**. El argumento **Name** solo es obligatorio si **Where** se establece como **xlLocationAsObject**. En este procedimiento, el método **Location** especifica que el gráfico debe incrustarse en la hoja de trabajo activa:

```
ActiveChart.Location Where:=xlLocationAsObject, _
Name:=mySheet.Name
```

A continuación, varias declaraciones dan forma al gráfico estableciendo varias propiedades.

3. Ejecuta el procedimiento.

6 Resumen

En este capítulo se han presentado varios ejemplos sobre cómo importar datos en Excel provenientes de una base de datos de Access. Aprendiste a controlar una aplicación Access desde un procedimiento VBA, realizando tareas como abrir formularios e informes de Access, crear nuevos formularios, ejecutar consultas de selección y parámetros y llamar a las funciones integradas en Access. Además, este capítulo te ha mostrado técnicas para crear archivos de texto, tablas de consulta y gráficos a partir de datos de Access.

En el próximo capítulo aprenderás cómo el manejo de eventos puede ayudarte a construir aplicaciones de hojas de cálculo que respondan a acciones del usuario.

Capítulo 15
Los eventos

¿Cómo se desactiva un menú contextual cuando un usuario hace clic en una celda de la hoja de cálculo? ¿Cómo se muestra un mensaje personalizado antes de abrir o cerrar un libro? ¿Cómo se pueden validar datos introducidos en una celda o rango de celdas a medida que se escriben? Para tener un control total sobre Excel debes aprender a responder ante los eventos. Aprender a programar eventos te permitirá implementar nuevas funcionalidades en una aplicación de Excel.

Lo primero que necesitas saber es qué es un evento. Un evento es una acción reconocida por un objeto. Un evento es algo que les sucede a los objetos que forman parte de Excel. Por ejemplo, hacer clic en el botón Guardar, escribir una palabra en una celda o simplemente, abrir un libro de Excel. Una vez que interiorices este concepto, podrás comprender de forma más sencilla qué les puede suceder a los objetos de Excel, Word u otras aplicaciones de Microsoft Office.

Los eventos pueden ser desencadenados por un usuario de la aplicación, por otro programa o por el propio sistema. Entonces, ¿cómo se puede desencadenar un evento? Supongamos que haces clic con el botón derecho del ratón de una celda de la hoja de cálculo. Esta acción en particular mostraría un menú contextual, lo que te permitiría acceder rápidamente a los comandos más utilizados relacionados con las celdas de la hoja de cálculo. ¿Pero qué pasa si esta respuesta no es lo que necesitas en ese momento? Es posible que quieras impedir que se haga clic con el botón derecho del ratón en una determinada hoja o quizá asegurarte de que aparezca un menú personalizado en vez del menú contextual habitual. Afortunadamente puedes usar VBA para escribir unas líneas que puedan reaccionar a los eventos cuando ocurren.

Los siguientes objetos de Excel pueden responder a los eventos:

- Hoja de cálculo.
- Hoja de gráficos.

- Tabla de consulta.
- Libro.
- Aplicación.

Puedes decidir qué debe suceder cuando ocurre un evento en particular, escribiendo un procedimiento de evento.

1 Introducción a los procedimientos de evento

Un procedimiento de evento es un procedimiento especial utilizado para reaccionar a eventos específicos. Este procedimiento contiene el código VBA necesario para manejar un evento en particular. Algunos eventos pueden requerir una sola línea de código, mientras que otros pueden ser más complejos. Los procedimientos de evento tienen nombres creados de forma automática de la siguiente forma:

```
NombreObjeto_NombreEvento()
```

Después del nombre del evento, y entre paréntesis, se pueden colocar los parámetros que serán enviados al procedimiento. El programador no puede cambiar el nombre del evento.

Antes de escribir un procedimiento de eventos necesitamos saber:

- **El nombre del objeto y del evento al que queremos responder**.
  La Imagen 1.1 ilustra cómo los objetos que responden a eventos muestran una lista de eventos en la lista desplegable **Procedimiento** en la ventana **Código**. También se puede utilizar el **Explorador de objetos** para encontrar los nombres de los eventos, como se muestra en la Imagen 1.2.
- **El lugar donde se debe poner el código del evento**.
  Algunos eventos se pueden crear en un módulo estándar (como los que hemos visto hasta ahora); otros deben ser introducidos en un módulo de clase.
  Los eventos de libro, hoja o gráfico están disponibles para cualquier hoja o libro de trabajo abierto. Para crear procedimientos de eventos para un gráfico o una tabla, primero se debe crear un nuevo objeto utilizando la palabra clave `WithEvents` en el módulo de clase.

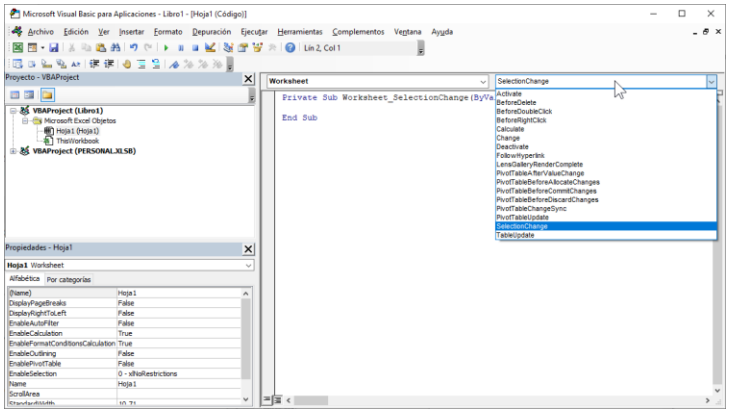


Imagen 1.1 Podemos encontrar los nombres de los eventos en la ventana Código.

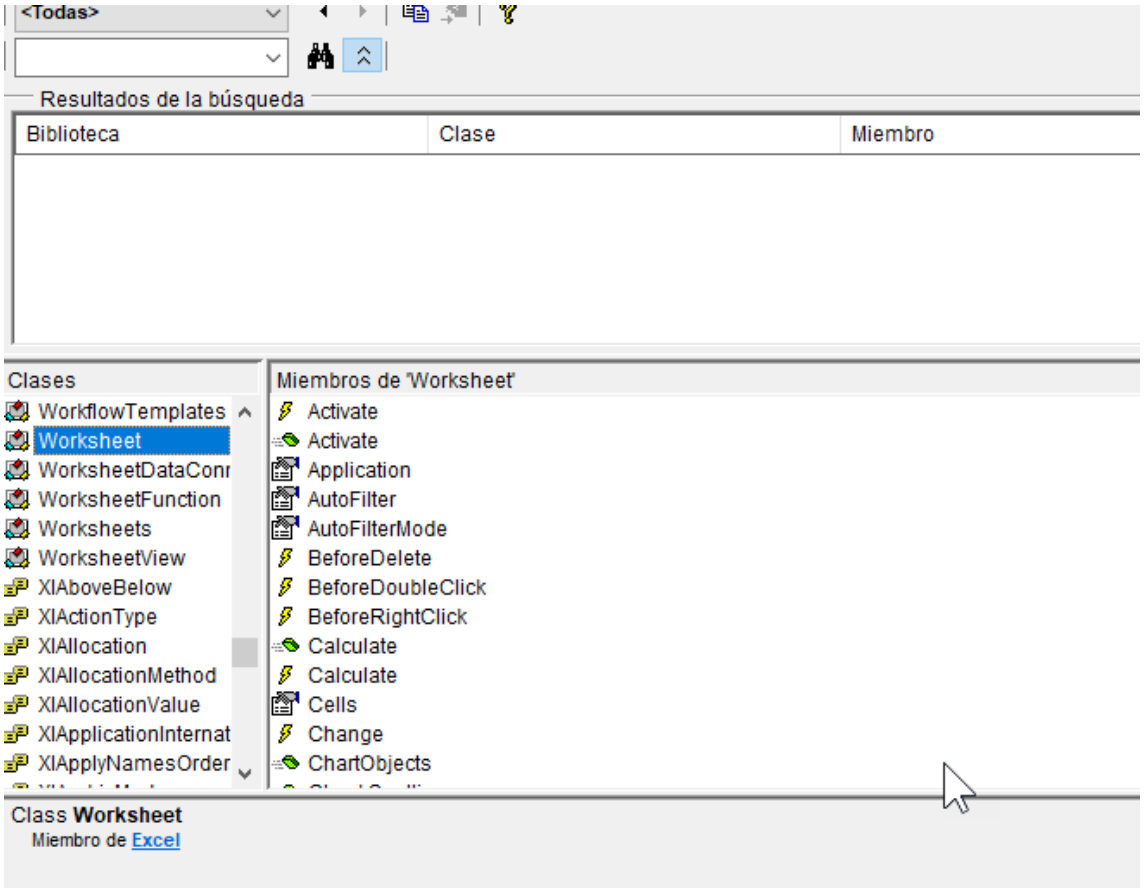


Imagen 1.2 También podemos encontrar los eventos en el Examinador de objetos.

2 El primer procedimiento de evento

A veces querremos activar una determinada operación cuando un usuario utilice un comando de Excel. Por ejemplo, cuando intente guardar un libro de Excel, es posible que queramos mostrarle la posibilidad de copiar la hoja de trabajo activa en otro libro. El primer procedimiento de eventos de este capítulo aborda esta acción.

Una vez escrito el procedimiento de evento, el código se ejecutará automáticamente cuando el usuario intente guardar el archivo del libro de trabajo en el que se encuentra el procedimiento:

1. Crea un nuevo libro de Excel y guárdalo como C:\Archivos Manual VBA\Capítulo 15 – Eventos.xlsm.
2. Cambia el nombre de la Hoja1 por `Test`.
3. Escribe alguna palabra en la celda A1 y presiona **Intro**.
4. Abre el editor de VBA.
5. En la ventana del **Explorador de proyectos** haz doble clic en el objeto `ThisWorkbook`.
6. En la ventana **Código**, introduce el siguiente procedimiento:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
Cancel As Boolean)
    If MsgBox("¿Te gustaría copiar " & vbCrLf _
        & "esta hoja en " & vbCrLf _
```

```
        & "un libro nuevo?", vbQuestion + vbYesNo) = vbYes Then
        Sheets(ActiveSheet.Name).Copy
    End If
End Sub
```

El procedimiento anterior utiliza una función `MsgBox` para mostrar un cuadro de diálogo de dos botones que pregunta al usuario si la hoja de trabajo actual debe ser copiada a otro libro de trabajo. Si el usuario hace clic en el botón Sí, VBA abrirá un nuevo libro de trabajo y copiará la hoja de trabajo activa en él. La hoja de trabajo original no se guardará. Sin embargo, si el usuario hace clic en No, se activará el evento `Save` integrado en Excel. Si el libro de trabajo nunca ha sido guardado antes, se mostrará el cuadro de diálogo **Guardar como** donde podemos especificar tanto el nombre del archivo como su formato y ubicación.

7. Cambia a la ventana de Excel, haz clic en la ficha **Archivo** y selecciona **Guardar**.
   El procedimiento del evento `Workbook_BeforeSave` del paso anterior se activará en ese momento. Haz clic en **Sí** en el cuadro de mensaje. Excel abrirá un nuevo libro de trabajo con la copia de la hoja de trabajo actual.
8. Cierra el libro de trabajo creado por Excel sin guardar los cambios. No cierres el libro **Capítulo 15 – Eventos.xlsm**.
9. Haz clic en la ficha **Archivo** y selecciona **Guardar**.
10. Observa nuevamente que se solicita una respuesta en el cuadro de mensaje. Haz clic en No y observa que el libro se ha guardado.
    Pero ¿qué pasa si quieres copiar la hoja del archivo en otro libro y también guardar el archivo original? Modifica el procedimiento para asegurarte de que el libro se guarde independientemente de si el usuario ha respondido Sí o No al mensaje.
    Cambia el procedimiento `Workbook_BeforeSave` de la siguiente forma:

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
Cancel As Boolean)
    Dim wkb As Workbook

    Set wkb = ActiveWorkbook
    Cancel = False
    If MsgBox("¿Te gustaría copiar " & vbCrLf _
        & "esta hoja en " & vbCrLf _
        & "un libro nuevo?", vbQuestion + vbYesNo) = vbYes Then
        Sheets(ActiveSheet.Name).Copy
        wkb.Activate
    End If
End Sub
```

Para continuar el proceso de guardado debes establecer el argumento `Cancel` en `False`. Esto activará el evento `Save` integrado en Excel. Como al copiar se moverá el foco al nuevo libro de trabajo que no contiene el procedimiento de evento, necesitarás activar el libro original después de realizar la copia. Podemos volver fácilmente al libro de trabajo original

estableciendo una referencia al principio del procedimiento de eventos y luego introduciendo la declaración `wkb.Activate`.

11. Escribe cualquier contenido en la hoja Test del archivo Capítulo 15 – Eventos.xlsm y haz clic en el botón **Guardar** de la barra de herramientas de acceso rápido. Al hacer clic en Sí o No en respuesta al `MsgBox`, Excel procede a guardar el archivo. Si has hecho clic en Sí, Excel también copia la hoja de trabajo Test en otro libro. Después de completarse todas estas tareas, Excel activa el archivo Capítulo 15 – Eventos.xlsm.
12. Si respondiste sí en el paso anterior, cierra el archivo creado automáticamente por Excel sin guardar los cambios. No cierres el archivo original. Ahora que ya sabes cómo usar el argumento `Cancel`, veamos el otro argumento del libro (`SaveAsUI`) antes de guardar el procedimiento. Este argumento te permite gestionar la posibilidad de que el usuario elija la opción **Guardar como**. Supón que, en el ejemplo del procedimiento, quieres pedir al usuario que copie la hoja en otro libro solo cuando se selecciona la opción **Guardar**. En los casos en los que el libro aún no ha sido guardado o el usuario quiere guardarlo con un nombre diferente, aparecerá el cuadro de diálogo **Guardar como** y no se mostrará el MsgBox que se ha creado.
13. Modifica el procedimiento del evento teniendo en cuenta el punto anterior.

```
Private Sub Workbook_BeforeSave(ByVal SaveAsUI As Boolean, _
Cancel As Boolean)
    If SaveAsUI = True Then Exit Sub
    Dim wkb As Workbook
    Set wkb = ActiveWorkbook
    Cancel = False
    If MsgBox("¿Te gustaría guardar una copia " & vbCrLf _
        & "de esta hoja en " & vbCrLf _
        & "un libro nuevo?", vbQuestion + vbYesNo) = vbYes Then
        Sheets(ActiveSheet.Name).Copy
        wkb.Activate
    End If
End Sub
```

14. Cambia a la ventana de Excel, haz clic en la ficha **Archivo** y selecciona **Guardar como – Libro de trabajo habilitado para macros de Excel**.
    Observa que no se pide que copies la hoja actual en otro libro. En su lugar, Excel procede a ejecutar su propio procedimiento de Guardar como.
    Cuando aparezca el cuadro de diálogo, haz clic en **Cancelar**.

3 Activar y desactivar eventos

Podemos utilizar la propiedad `EnableEvents` del objeto `Application` para activar o desactivar eventos. Si estamos escribiendo un procedimiento y no queremos que ocurra un evento en particular, debemos establecer la propiedad `EnableEvents` en False.

Para mostrar cómo se puede evitar que se ejecute un procedimiento de eventos, escribiremos un procedimiento en un módulo estándar que guardará el libro después de realizar algunos cambios en la hoja activa. Seguiremos trabajando con el archivo Capítulo 15 – Eventos.xlsm porque ya contiene el procedimiento del evento `Worksheet_BeforeSave` que queremos bloquear en el ejemplo.

1. Inserta un módulo nuevo y llámalo `ProcedimientosEstandar`
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub IntroduccionDatos()
    With ActiveSheet.Range("A1:B1")
        .Font.Color = vbRed
        .Value = 15
    End With
    Application.EnableEvents = False
    ActiveWorkbook.Save
    Application.EnableEvents = True
End Sub
```

Observa que antes de llamar al método `Save` de la propiedad `ActiveWorkbook`, se han desactivado los eventos poniendo la propiedad `EnableEvents` en False. Eso evitará que se ejecute el procedimiento del evento `Workbook_BeforeSave` cuando VBA encuentre la instrucción para guardar el libro. No queremos que se le solicite al usuario que copie la hoja de trabajo mientras se ejecuta el procedimiento `IntroduccionDatos`. Cuando VBA haya completado el proceso de guardado, queremos que el sistema vuelva a su situación original, es decir, que responda ante eventos, por lo que los habilitamos con la instrucción `Application.EnableEvents = True`.

3. Cambia a la ventana de la aplicación y haz clic en la ficha **Vista > Macros > Ver macros**. En el cuadro de diálogo **Macro**, selecciona `IntroduccionDatos` y presiona en **Ejecutar**.
   Observa que cuando ejecutas el procedimiento, no se solicita que copies la hoja antes de guardarla. Esto indica que el código que introdujiste anteriormente (`Workbook_BeforeSave`) no se está ejecutando.
4. Ya puedes cerrar el libro.

4 Secuencias de eventos

Los eventos se producen en respuesta a acciones específicas. También ocurren en una secuencia predefinida. En la siguiente tabla se muestra la secuencia de eventos que suceden cuando abrimos un nuevo libro de trabajo, añadimos una nueva hoja a un libro y cerramos un libro.

Acción	Objeto	Secuencia de eventos
Crear un libro nuevo	Workbook	NewWorkbook ↓ WindowDeactivate ↓ WorkbookDeativate ↓ WorkbookActivate ↓ WindowActivate
Insertar una hoja den un libro	Workbook	WorkbookNewSheet ↓ SheetDeactivate ↓ SheetActivate
Cerrar un libro	Workbook	WorkbookBeforeClose ↓ WindowDeactivate ↓ WorkbookDeactivate ↓ WorkbookActivate ↓ WindowActivate

4.1 Eventos de hoja

El objeto **`Worksheet`** responde a eventos como activar y desactivar una hoja de cálculo, calcular datos en una hoja, hacer un cambio en una celda o hacer clic con el botón derecho. En la siguiente tabla se enumeran algunos de los eventos asociados al objeto **`Worksheet`**.

Evento	Descripción
Activate	Ocurre cuando se activa la hoja
Deactivate	Ocurre cuando se desactiva la hoja
SelectionChange	Ocurre cuando se selecciona una celda
Change	Ocurre cuando cambia una celda
Calculate	Ocurre cuando se recalcula la hoja
BeforeDoubleClick	Ocurre cuando se hace doble clic sobre una celda
BeforeRightClick	Ocurre cuando se hace clic con el botón derecho en una celda

Vamos a probar estos eventos para familiarizarnos con ellos.

4.1.1 Worksheet_Activate()

Este evento ocurre cuando se activa una hoja.

1. Abre el libro Capítulo15 – Eventos.xlsm.
2. Inserta una hoja nueva en el libro.
3. Abre el editor de VBA.
4. En el **Explorador de proyectos** selecciona el objeto correspondiente a la hoja que acabas de crear y haz doble clic sobre él.
5. En la ventana **Código** introduce el siguiente procedimiento:

```
Dim shtNombre As String
Private Sub Worksheet_Activate()
    shtNombre = ActiveSheet.Name
    Range("B2").Select
End Sub
```

El procedimiento selecciona la celda B2 cada vez que se activa la hoja. Fíjate que la variable `shtNombre` se declara fuera del procedimiento.

6. Cambia a la ventana de Excel y activa la Hoja2.
7. Observa que cuando se activa la hoja 2 la celda activa se mueve a la celda B2. Excel también almacena el nombre de la hoja en la variable `shtNombre` que fue declarada en la parte superior del módulo. Necesitarás este valor cuando trabajes con otros procedimientos de eventos de esta sección.

4.1.2 Worksheet_Deactivate()

Este evento ocurre cuando el usuario activa una hoja diferente del libro.

1. Dirígete al editor de VBA e introduce el siguiente procedimiento en la ventana **Código** de la hoja del ejemplo anterior:

```
Private Sub Worksheet_Deactivate()
    MsgBox "Has desactivado la hoja " & _
    shtNombre & "." & vbCrLf & _
    "Has activado la hoja " & _
    ActiveSheet.Name & "."
End Sub
```

Este ejemplo muestra un mensaje cuando se desactiva la Hoja 2.

2. Cambia a la ventana de Excel y activa la Hoja 2. Se activará el procedimiento `Worksheet_Activate` que creaste en la sección anterior activando la celda B2 y guardando el nombre de la hoja en la `variable shtNombre`.
3. Ahora haz clic en cualquier otra hoja. Observa que Excel muestra el nombre de la hoja que has desactivado y el nombre de la hoja actual.

4.1.3 Worksheet_SelectionChange()

Este evento ocurre cuando se selecciona una celda de la hoja.

1. En el libro actual crea una nueva hoja.
2. En el **Explorador de proyectos** del editor de VBA selecciona el objeto correspondiente a la hoja nueva y haz doble clic sobre él.
3. Introduce el siguiente procedimiento:

```
Private Sub Worksheet_SelectionChange _
(ByVal Target As Excel.Range)
    Dim miRango As Range

    On Error Resume Next
    Set miRango = Intersect(Range("A1:A10"), Target)
    If Not miRango Is Nothing Then
        MsgBox _
"No está permitida la introducción de datos en el rango A1:A10."
    End If
End Sub
```

Este ejemplo muestra un mensaje si el usuario selecciona cualquier celda de `MiRango`

4. Activa la ventana de Excel y activa la Hoja 3. A continuación haz clic en cualquier celda dentro del rango especificado (A1:A10).
   Observa que Excel muestra un mensaje cada vez que haces clic en la zona restringida.

4.1.4 Worksheet_Change()

Este evento ocurre cuando un usuario modifica el contenido de una celda.

1. En el editor de VBA haz doble clic en el objeto Hoja 1 (Sheet1).
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Worksheet_Change(ByVal Target As Excel.Range)
```

```
        Application.EnableEvents = False
        Target = UCase(Target)
        Columns(Target.Column).AutoFit
        Application.EnableEvents = True
    End Sub
```

El procedimiento cambia el contenido de lo que se escribe a letras mayúsculas. La columna en la que se encuentra la celda modificada ajusta su tamaño al ancho del contenido.

3. Cambia a la ventana de Excel y activa la Hoja 1. Introduce cualquier texto en cualquier celda.
   Observa que en cuanto presionas **Intro**, Excel cambia el texto escrito a mayúsculas y ajusta automáticamente el tamaño de la columna.

4.1.5 Worksheet_Calculate()

Este evento ocurre cuando se calcula o recalcula la hoja.

1. Agrega una nueva hoja al libro. En la celda A2 de la nueva hoja introduce el número 1. En la celda B2 introduce el número 2. En la celda C2 introduce la fórmula =A2+B2.
2. Activa el editor de VBA y haz doble clic sobre el objeto de la nueva hoja.
3. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Worksheet_Calculate()
    MsgBox "La hoja ha sido recalculada."
End Sub
```

4. Activa la ventana de Excel y modifica el contenido de la celda B2 introduciendo cualquier número. Observa que al salir del modo edición se activa el procedimiento `Worksheet_Calculate`, que muestra un mensaje personalizado.

4.1.6 Worksheet_BeforeDoubleClick(ByVal Target As Range, Cancel As Boolean)

Este evento ocurre cuando el usuario hace doble clic en una celda de la hoja.

1. Introduce cualquier dato en la celda C9 de la Hoja 2 del libro.
2. En el editor de VBA, haz doble clic en el objeto correspondiente a la hoja 2.
3. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Worksheet_BeforeDoubleClick(ByVal _
    Target As Range, Cancel As Boolean)
    If Target.Address = "$C$9" Then
        MsgBox "No se puede hacer doble clic sobre esta celda."
        Cancel = True
    Else
        MsgBox "Puedes editar la celda."
    End If
End Sub
```

El procedimiento no permite la edición en la celda cuando se hace doble clic sobre C9.

4. Activa la ventana de Excel y haz clic sobre la celda C9 de la Hoja 2.
   El procedimiento cancela el comportamiento habitual de Excel y el usuario no puede editar los datos de la celda. Sin embargo, se puede sortear esta restricción haciendo clic en la barra de fórmulas o presionando **F2**. Cuando escribas procedimientos de evento que restrinjan el acceso a ciertas características del programa, escribe también un código que impida cualquier solución.

4.1.7 Worksheet_BeforeRightClick(ByVal Target As Range, Cancel As Boolean)

Este evento ocurre cuando el usuario hace clic con el botón derecho en la hoja.

1. Activa el editor de VBA y haz doble clic sobre el objeto Hoja 2.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Worksheet_BeforeRightClick(ByVal _
    Target As Range, Cancel As Boolean)
    With Application.CommandBars("Cell")
        .Reset
        If Target.Rows.Count > 1 Or _
            Target.Columns.Count > 1 Then
            With .Controls.Add(Type:=msoControlButton, _
                before:=1, temporary:=True)
                .Caption = "Imprimir..."
                .OnAction = "PrintMe"
            End With
        End If
    End With
End Sub
```

El procedimiento añade una opción de impresión al menú contextual cuando se selecciona más de una celda en la hoja.

3. Inserta un nuevo módulo en el proyecto e introduce el siguiente procedimiento en él.

```
Sub Imprime()
   Application.Dialogs(xlDialogPrint).Show arg12:=1
End Sub
```

   El procedimiento `Imprime` es llamado por la hoja antes de que se haga clic con el botón derecho. Observa que el método `Show` de la colección `Dialogs` va seguido de un argumento (`arg12:=1`). Este argumento mostrará el cuadro de diálogo **Imprimir** con el botón de opción preseleccionado. Los cuadros de diálogo de Excel se tratan en el siguiente capítulo.

4. Activa la ventana de Excel y haz clic con el botón derecho del ratón en cualquier celda de la Hoja 4.
   Observa que el menú contextual aparece con las opciones predeterminadas.

5. Selecciona dos o más celdas y vuelve a hacer clic con el botón derecho del ratón en el área seleccionada. Deberías ver la opción **Imprimir** en la parte superior del menú. Haz clic en la opción **Imprimir** y observa que en lugar de aparecer la opción "Imprimir hoja activa", el cuadro muestra "Imprimir selección".
6. Guarda y cierra el archivo.

5 Eventos de libro (Workbook)

Los eventos del objeto `Workbook` ocurren cuando el usuario realiza tareas como abrir, activar, desactivar, imprimir, guardar o cerrar un libro de Excel. Estos eventos de libro no se crean en un módulo VBA estándar. Para escribir código que responda a un libro de trabajo en particular podemos:

- Hacer doble clic en el objeto `ThisWorkbook` en el **Explorador de proyectos** del editor de VBA.
- En la ventana **Código** que aparece, hacer clic en la lista desplegable **Objeto** (la de la izquierda) y seleccionar el objeto `Workbook`.
- En la lista desplegable **Procedimiento** (la de la derecha) seleccionar el evento que deseamos. En la ventana **Código** aparecerá el encabezado del evento seleccionado como se muestra a continuación:

```
Private Sub Workbook_Open()
'Introduce aquí tu código
End Sub
```

A continuación se muestra una lista de los eventos ante los que puede responder el objeto `Workbook`:

Evento	Descripción
Activate	Ocurre cuando el usuario activa el libro. Este evento no se producirá cuando el usuario active el libro de trabajo al cambiar de aplicación.
Deactivate	Ocurre cuando el usuario activa un libro de trabajo diferente dentro de Excel. No ocurrirá cuando el usuario cambia a una aplicación diferente.
Open	Ocurre cuando el usuario abre un libro.
BeforeSave	Ocurre antes de que el libro se guarde. El argumento `SaveAsUI` es de solo lectura y se refiere al cuadro de diálogo Guardar como. Si el libro no se ha guardado, el valor de

Evento	Descripción
	`SaveAsUI` es `True`; de lo contrario, es `False`.
BeforePrint	Ocurre antes de que se imprima el libro y antes de que aparezca el cuadro de diálogo de impresión.
BeforeClose	Ocurre antes de que se cierre el libro y antes de que se le pida al usuario que guarde los cambios.
NewSheet	Ocurre cuando el usuario crea una nueva hoja.
WindowActivate	Ocurre cuando el usuario cambia el foco a cualquier ventana que muestre el libro.
WindowDeactivate	Ocurre cuando el usuario quita el foro de cualquier ventana que muestre el libro.
WindowResize	Ocurre cuando el usuario abre, redimensiona, maximiza o minimiza cualquier ventana que muestre el libro.

Vamos a probar los eventos anteriores para familiarizarnos con ellos.

5.1.1 Workbook_Activate()

Este evento ocurre cuando el usuario activa el libro pero no lo hace cuando el libro se activa al cambiar de aplicación.

1. Abre el archivo Capítulo 15 – Eventos.xlsm.
2. Activa el editor de VBA y haz doble clic sobre el objeto `ThisWorkbook`.
3. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Workbook_Activate()
    MsgBox "Este libro contiene " & _
    ThisWorkbook.Sheets.Count & " hojas."
End Sub
```

El procedimiento muestra el número total de hojas del libro cuando el usuario activa el libro que contiene el evento `Workbook_Activate`.

4. Activa la ventana de Excel y abre un nuevo libro.
5. Activa el Capítulo 15 – Eventos.xlsm. Excel debería mostrar el número total de hojas del libro.

5.1.2 Workbook_Deactivate()

Este evento ocurre cuando el usuario activa un libro de Excel diferente desde otro libro. El evento no se produce cuando el usuario cambia de aplicación.

1. En el editor de VBA, haz doble clic en el objeto **ThisWorkbook**.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Workbook_Deactivate()
    Dim cell As Range

    For Each cell In ActiveSheet.UsedRange
        If Not IsEmpty(cell) Then
            Debug.Print cell.Address & ":" & cell.Value
        End If
    Next
End Sub
```

El procedimiento imprimirá en la ventana **Inmediato** las direcciones y valores de las celdas que estén vacías en el libro cuando el usuario active un libro de trabajo diferente.

3. Activa la ventana de Excel e introduce algunos datos en la hoja activa. A continuación, activa un libro diferente. Esta acción activará el procedimiento **Workbook_Deactivate**.
4. Activa el editor de VBA y abre la ventana **Inmediato** para ver los datos que se han impreso en ella.

5.1.3 Workbook_Open()

Este evento ocurre cuando el usuario abre un libro.

1. Haz doble clic en el objeto **ThisWorkbook** del libro Capítulo 15 – Eventos.xlsm.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Private Sub Workbook_Open()
    ActiveSheet.Range("A1").Value = Format(Now(), "dd/mm/yyyy")
    Columns("A").AutoFit
End Sub
```

El procedimiento introduce la fecha actual en la celda A1 cuando el libro se abre.

3. Guarda el libro y ciérralo. A continuación, vuélvelo a abrir. Al abrirlo se activa el evento **Workbook_Open** y se ejecuta el procedimiento, mostrando la fecha de hoy en la celda A1 de la hoja activa.

5.1.4 Workbook_BeforeSave(ByVal SaveAsUI As Boolean, Cancel As Boolean)

Este evento ocurre antes de que el libro sea guardado. El argumento **SaveAsUI** es de solo lectura y hace referencia al cuadro de diálogo **Guardar como**. Si el libro no se ha guardado, el valor de **SaveAsUI** es **True**; de lo contrario, es **False**.

1. En el editor de VBA haz doble clic en el objeto **ThisWorkbook**.

2. Introduce le siguiente procedimiento en la ventana **Código**:

```
Private Sub Workbook_BeforeSave(ByVal _
SaveAsUI As Boolean, Cancel As Boolean)
    If SaveAsUI = True And _
            ThisWorkbook.Path = vbNullString Then
            MsgBox "Este documento todavía " _
            & "No se ha guardado." & vbCrLf _
            & "Ahora se mostrará el cuadro Guardar como."
        ElseIf SaveAsUI = True Then
            MsgBox "No puedes utilizar " _
            & "la opción Guardar como. "
            Cancel = True
    End If
End Sub
```

El procedimiento muestra el cuadro de diálogo **Guardar como** si el libro no se ha guardado antes. La ruta del libro es una cadena de texto nula (`vbNullString`) si el archivo no ha sido guardado nunca. El procedimiento no permitirá que el usuario guarde el libro con un nombre diferente. La operación **Guardar como** se abortará al establecer el argumento Cancel en True. El usuario deberá elegir la opción **Guardar** para poder grabar el libro.

3. Activa la ventana de Excel y activa cualquier hoja del libro.
4. Escribe algo de contenido en cualquier celda del libro. A continuación, dirígete a la ficha **Archivo**y selecciona **Guardar como > Libro habilitado para macros**. El procedimiento del evento `Workbook_BeforeSave` será activado, y la cláusula `ElseIf` será ejecutada. Observa que no se permite guardar el libro mediante la opción **Guardar como**.

5.1.5 Workbook_BeforePrint(Cancel As Boolean)

Este evento ocurre antes de que se imprima el libro, e incluso antes de que se muestre el cuadro de diálogo **Imprimir**.

1. Haz doble clic en el objeto `ThisWorkbook` para mostrar la ventana **Código**.
2. Introduce el siguiente procedimiento:

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Dim response As Integer

    response = MsgBox("¿Te gustaría " & vbCrLf & _
    "imprimir el nombre del libro al pie del documento?", _
    vbYesNo)
    If response = vbYes Then
        ActiveSheet.PageSetup.LeftFooter = _
        ThisWorkbook.FullName
```

```
        Else
            ActiveSheet.PageSetup.LeftFooter = ""
        End If
    End Sub
```

El procedimiento coloca el nombre del libro en el pie de página del documento antes de imprimirlo si el usuario hace clic en Sí en el cuadro de mensaje que aparece.

3. Activa la ventana de Excel y a continuación activa cualquier hoja del libro.
4. Introduce algo de contenido en cualquier celda.
5. Haz clic en la ficha **Archivo > Imprimir** y haz clic en el botón **Imprimir**.
6. Excel te preguntará si deseas colocar el nombre y la ruta del libro en el pie de página.

5.1.6 Workbook_BeforeClose(Cancel As Boolean)

Este evento ocurre antes de cerrar el libro de Excel y antes de que se pregunte al usuario si desea guardar los cambios.

1. En el editor de VBA haz doble clic en el objeto `ThisWorkbook`.
2. Introduce el siguiente procedimiento en la ventana **Código**:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    If MsgBox("¿Deseas modificar las propiedades " & vbCrLf _
        & " del libro antes de cerrarlo?", _
        vbYesNo) = vbYes Then
        Application.Dialogs(xlDialogProperties).Show
    End If
End Sub
```

El procedimiento muestra el cuadro de diálogo **Propiedades** si el usuario responde Sí al cuadro de mensaje.

3. Activa la ventana de Excel y cierra el libro.
   Al cerrarlo, debería aparecer un mensaje que te solicita que veas el cuadro de diálogo **Propiedades** antes de cerrarlo. Después de ver o modificar las propiedades del libro, el procedimiento lo cierra. Si hay algún cambio que aún no has guardado, se da la oportunidad de guardarlos, cancelarlos o abortar la operación de cierre del libro.

5.1.7 Workbook_NewSheet(ByVal Sh As Object)

Este evento ocurre después de que el usuario cree una nueva hoja en el libro.

1. Abre el archivo Capítulo 15 – Eventos.xlsm.
2. Activa el editor de VBA y haz clic en el objeto `ThisWorkbook` del **Explorador de proyectos**.
3. Introduce el siguiente procedimiento:

```
Private Sub Workbook_NewSheet(ByVal Sh As Object)
    If MsgBox("¿Quieres situar " & vbCrLf _
    & "la hoja nueva al principio " & vbCrLf _
```

```
        & "del libro?", vbYesNo) = vbYes Then
        Sh.Move before:=ThisWorkbook.Sheets(1)
        Else
        Sh.Move After:=ThisWorkbook.Sheets( _
        ThisWorkbook.Sheets.Count)
        MsgBox Sh.Name & _
        " is now the last sheet in the workbook."
        End If
    End Sub
```

El procedimiento coloca la nueva hoja al principio del libro si el usuario responde Sí al cuadro de mensajes; de lo contrario, la nueva hoja se coloca al final.

4. Activa la ventana de Excel y haz clic en el botón **Nueva hoja** (+) (en la parte inferior de la pantalla). Excel preguntará dónde colocar la nueva hoja.

Vamos a probar ahora algunos eventos relacionados con la ventana del libro.

5.1.8 Workbook_WindowActivate(ByVal Wn As Window)

Este evento ocurre cuando el usuario cambia el foco a cualquier ventana que muestre el libro de Excel.

1. En el editor de VBA activa el **Explorador de proyectos** y haz doble clic en el objeto `ThisWorkbook`.
2. Introduce el siguiente procedimiento en la ventana **Código**:

```
Private Sub Workbook_WindowActivate(ByVal Wn As Window)
    Wn.GridlineColor = vbYellow
End Sub
```

El procedimiento cambia el color de la cuadrícula de la hoja de cálculo a amarillo cuando el usuario activa el libro que contiene el código de `Workbook_WindowActivate`.

3. Activa la ventana de Excel y crea un libro de trabajo nuevo.
4. Organiza los libros verticalmente en la pantalla haciendo clic en la ficha **Vista > Organizar todo**, para abrir el cuadro de diálogo **Organizar ventanas**. Selecciona la opción **Vertical** y haz clic en **Aceptar**. Al activar la hoja del libro donde introdujiste el código, el color de la cuadrícula cambiará a amarillo.

5.1.9 Workbook_WindowDeactivate(ByVal Wn As Window)

Este evento ocurre cuando el usuario quita el foco de cualquier ventana de Excel.

1. Haz doble clic en el objeto `ThisWorkbook` en el **Explorador de proyectos**.
2. Introduce el siguiente procedimiento en la ventana **Código**:

```
Private Sub Workbook_WindowDeactivate(ByVal Wn As Window)
    MsgBox "Acabas de desactivar " & Wn.Caption
End Sub
```

Este ejemplo muestra el nombre del libro desactivado cuando el usuario pasa a otro libro diferente.

3. Activa la ventana de Excel y crea un libro nuevo. Excel muestra el nombre de libro desactivado en un cuadro de mensaje.

5.1.10 Workbook_WindowResize(ByVal Wn As Window)

Este evento ocurre cuando se produce un cambio en el estado de la ventana, cuando el usuario abre, redimensiona, maximiza o minimiza cualquier ventana de libro.

1. En la ventana **Código** del objeto **ThisWorkbook**, introduce el siguiente código:

```
Private Sub Workbook_WindowResize(ByVal Wn As Window)
    If Wn.WindowState <> xlMaximized Then
        Wn.Left = 0
        Wn.Top = 0
    End If
End Sub
```

El procedimiento mueve la ventana del libro a la esquina superior izquierda de la pantalla cuando el usuario la cambia de tamaño.

2. Activa el libro de Excel.
3. Haz clic en el botón **Restauración** de la ventana (en la parte derecha de la cinta de opciones.
4. Arrastra la ventana al centro de la pantalla con el ratón desde la barra de título.
5. Modifica el tamaño de la ventana activa arrastrando los bordes hacia afuera.
   Al completar la operación, la ventana del libro debería saltar automáticamente a la esquina superior izquierda de la pantalla.
   Haz clic en el botón **Maximizar** para mostrar la ventana del libro Capítulo 15 – Eventos.xlsm a su tamaño completo.

Hemos visto algunos eventos ante los que puede responder un libro de Excel, pero hay muchos más. Se muestran en la siguiente tabla:

Evento	Descripción
SheetActivate	Ocurre cuando el usuario activa cualquier hoja del libro. El evento SheetActivate ocurre también a nivel de aplicación.
SheetDeactivate	Ocurre cuando se activa una hoja diferente del libro.
SheetSelectionChange	Ocurre cuando se modifica la selección en cualquier hoja del libro.

Evento	Descripción
SheetChange	Ocurre cuando se cambia un valor o fórmula de una hoja.
SheetCalculate	Ocurre cuando la hoja se recalcula.
SheetBeforeDoubleClick	Ocurre cuando el usuario hace doble clic en una celda.
SheetBeforeRightClick	Ocurre cuando se hace clic con el botón derecho del ratón en cualquier hoja.
NewChart	Ocurre cuando se crea un gráfico nuevo.
AfterSave	Ocurre cuando el libro se guarda.
AddinInstall	Ocurre después de instalar un complemento en el libro.
AddinUninstall	Ocurre después de instalar un complemento en el libro.
SheetFollowHyperlink	Ocurre cuando se hace clic en un hipervínculo del libro.

Debemos diferenciar los objetos **`Worksheet`** y **`Sheet`**. El primero hace referencia a lo que se denomina hoja de trabajo (las hojas con celdas). El segundo se refiere a cualquier tipo de hoja de Excel (hoja de gráficos, hoja de macros u hoja de cuadro de diálogo).

6 Eventos de tabla dinámica

Todos sabemos que las tablas dinámicas proporcionan una poderosa forma de analizar y comparar grandes cantidades de información almacenada en una base de datos. Al rotar las filas y columnas de la tabla dinámica, se pueden obtener diferentes vistas de los datos de origen o ver detalles de los datos que más nos interesan.

Cuando trabajamos con informes de tabla dinámica por medio de programación, podemos determinar cuándo se abrió o cerró la conexión con la fuente de datos gracias a los eventos del libro **`PivotTableOpenConnection`** y **`PivotTableCloseConnection`** y determinar cuándo se actualizó por última vez mediante el evento **`SheetPivotTableUpdate`**.

En la siguiente tabla se enumeran los eventos relacionados con tablas dinámicas. Si nunca has trabajado con esta herramienta de forma automatizada, el Capítulo 22 te permitirá comenzar a escribir código VBA para crear y manipular tablas dinámicas. Verás que es más fácil profundizar en la programación de eventos de tablas dinámicas después de trabajar en el Capítulo 22.

Evento	Descripción
PivotTableOpenConnection	Ocurre después de que una tabla dinámica abra una conexión con la fuente de datos. Este evento requiere que se declare un objeto **Application** o **Workbook** utilizando la palabra **WithEvents** en un módulo de clase (ver ejemplos de uso más adelante).
PivotTableCloseConnection	Ocurre después de que una tabla dinámica cierre una conexión con la fuente de datos. Este evento requiere que se declare un objeto **Application** o **Workbook** utilizando la palabra **WithEvents** en un módulo de clase (ver ejemplos de uso más adelante).

SheetPivotTableUpdate
 	
El procedimiento cuenta con dos argumentos:
• **Sh**: la hoja seleccionada.
• **Target**: la tabla dinámica.
	Este evento ocurre después de actualizar la tabla dinámica.
	
	Este evento requiere que se declare un objeto **Application** o **Workbook** utilizando la palabra **WithEvents** en un módulo de clase (ver ejemplos de uso más adelante).
	
	Nota: El procedimiento que se muestra a continuación junto con los demás procedimientos sobre eventos de tablas dinámicas se pueden encontrar en los archivos del manual.
	
	`Private Sub pivTbl_SheetPivotTableUpdate(_`
	`ByVal Sh As Object, _`
	`ByVal Target As PivotTable)`
	`MsgBox Target.Name & _`
	`" ha sido actualizado." & vbCrLf _`

Evento	Descripción

	`& "La tabla dinámica se encuentra en las celdas " & _`
	
	`Target.DataBodyRange.Address`
	`End Sub`

SheetPivotTableChangeSync El procedimiento cuenta con dos argumentos: • `Sh`: la hoja seleccionada. • `Target`: la tabla dinámica.	Ocurre después de modificar una tabla dinámica. Se puede utilizar para mostrar un mensaje.
`SheetPivotTableAfterValueChange`	Ocurre después de que se editen una o varias celdas o se recalcule la tabla dinámica. Este evento no ocurrirá al ordenar, filtrar, o actualizar la tabla.
`SheetPivotTableBeforeDiscardChanges`	Ocurre antes de descartar los cambios de la tabla dinámica. Se utiliza con fuentes de datos OLAP.
`SheetPivotTableBeforeCommitChanges`	Ocurre inmediatamente antes de que se hagan cambios en la fuente de datos.
`SheetPivotTableBeforeAllocateChanges`	Ocurre antes de que los cambios se apliquen a la fuente de datos de la tabla.

7 Eventos de gráfico

Como ya sabemos, es posible crear gráficos incrustados en una hoja de trabajo o en una hoja de gráfico de Excel. En esta sección, aprenderemos a controlar los eventos de los gráficos sin importar su ubicación. Antes de probar los eventos de gráficos, vamos a realizar algunas tareas:

1. Crea un nuevo libro de Excel y llámalo Capítulo 15 – Eventos de gráfico.xlsm.
2. Introduce los datos que se muestran en la Imagen 1.1.
3. Selecciona las celdas A1:D5 y haz clic en la ficha **Insertar > Gráficos > Gráfico de columna > Columna en 2-D**.
4. Usando los mismos datos, crea un gráfico de líneas en una hoja de gráfico como se muestra en la Imagen 7.3. Para añadir una hoja de gráfico, haz clic con el botón derecho del ratón en el nombre de cualquier hoja de libro y selecciona **Insertar**. En el cuadro de diálogo **Insertar**, selecciona **Gráfico** y haz clic en **Aceptar**. En el grupo **Datos**

de la ficha contextual **Diseño de gráfico** haz clic en el botón **Seleccionar datos**. Excel mostrará el cuadro de diálogo **Seleccionar fuente de datos**.

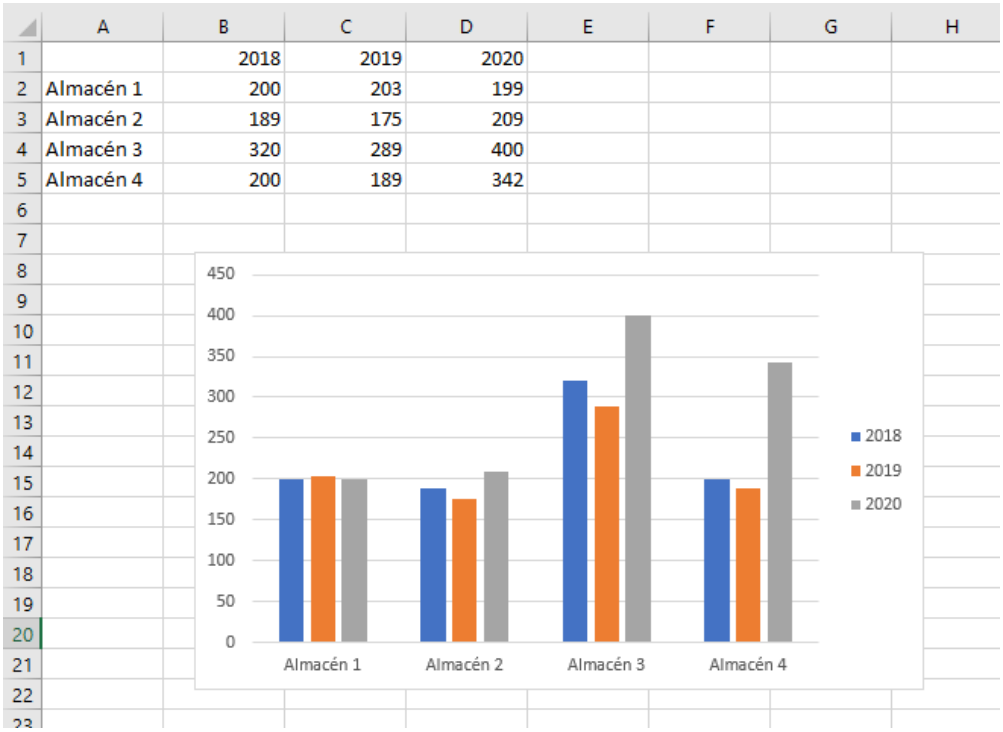


Imagen 7.1 Un gráfico de columnas incrustado en una hoja.

A continuación, haz clic en la pestaña de la Hoja1 y selecciona las celdas A1:D5. Excel rellenará el cuadro de diálogo, como en la Imagen 7.2. Haz clic en **Aceptar** para introducir el gráfico. Ahora cambia el tipo de gráfico a gráfico de líneas con marcadores seleccionando el botón **Cambiar tipo de gráfico** en la ficha **Diseño de gráfico**. En el cuadro de diálogo **Cambiar tipo de gráfico**, selecciona gráfico de líneas en el panel izquierdo y haz clic en el botón de gráfico de líneas con marcadores. Haz clic en **Aceptar** para cerrar el cuadro.

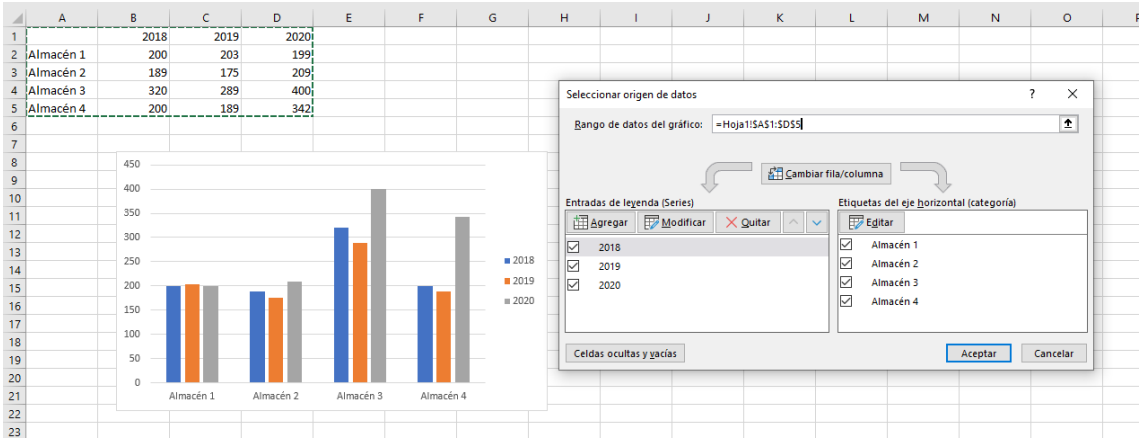


Imagen 7.2 Creando un gráfico.

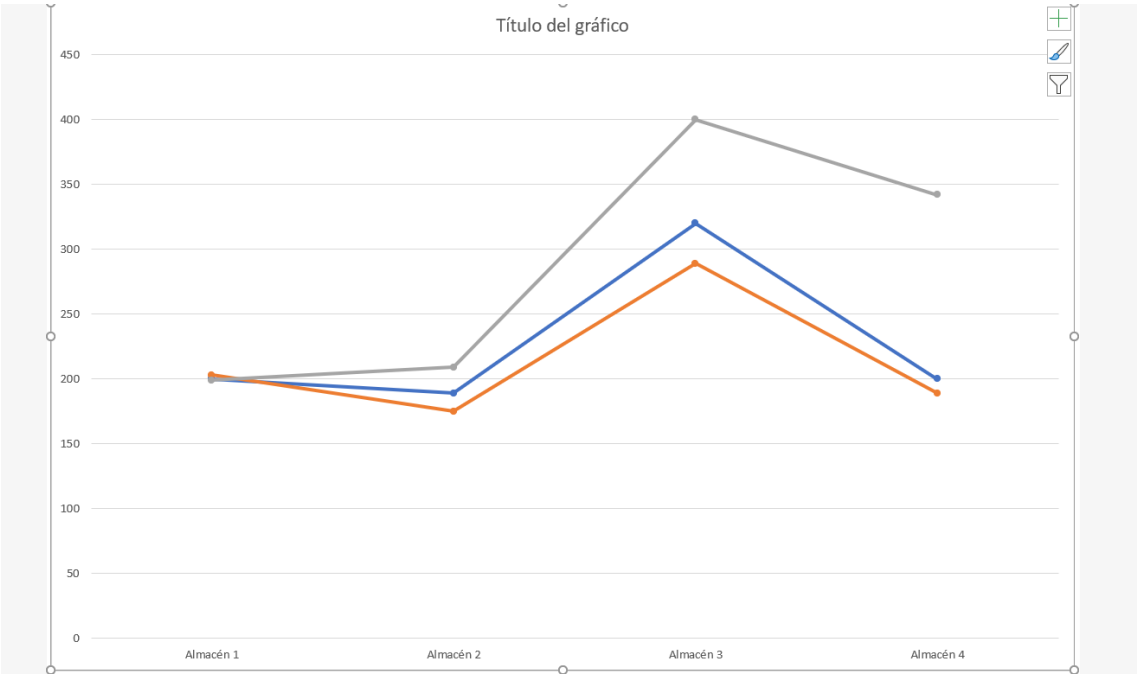


Imagen 7.3 Gráfico ubicado en una hoja de gráfico.

5. Modifica el nombre de la hoja de grafico a Análisis de ventas.

7.1 Procedimientos de eventos para gráficos localizados en una hoja de gráfico

Los gráficos de Excel pueden responder a numerosos eventos, tal y como se muestra en la siguiente tabla:

Evento	El evento ocurre cuando…
`Activate`	Se activa la hoja de gráfico.
`Deactivate`	Se desactiva la hoja de gráfico.
`Select`	Se selecciona un elemento del gráfico.
`SeriesChange`	Se cambia el valor de un dato del gráfico. El objeto Chart debe ser declarado utilizando la palabra `WithEvents`.
`Calculate`	Se trazan datos nuevos o se modifica el gráfico.
`Resize`	Se cambia el tamaño del gráfico. El objeto Chart debe ser declarado utilizando la palabra `WithEvents`.
`BeforeDoubleClick`	Se hace doble clic en un gráfico incrustado, antes de la acción predeterminada del doble clic.
`BeforeRightClick`	Se hace clic con el botón derecho del ratón en un gráfico incrustado.

Evento	El evento ocurre cuando…
`MouseDown`	Se presiona el botón del ratón encima del gráfico.
`MouseMove`	El cursor del ratón se mueve encima del gráfico.
`MouseUp`	Se suelta el botón del ratón encima del gráfico.

Comenzaremos escribiendo procedimientos de eventos que controlan un cuadro colocado en una hoja de gráfico, como se muestra en la Imagen 7.3. Los eventos relacionados con gráficos incrustados requieren el uso de la palabra `WithEvents` y se explican más adelante.

7.1.1 Chart_Activate()

El evento ocurre cuando el usuario activa la hoja de gráfico.

7.1.2 Chart_Deactivate()

El evento ocurre cuando se desactiva la hoja de gráfico.

7.1.3 Chart_Select(ByVal ElementID As Long, ByVal Arg1 As Long, ByVal Arg2 As Long)

Este evento ocurre cuando el usuario selecciona un elemento del gráfico. `ElementID` devuelve una constante que representa el tipo del gráfico seleccionado. Los argumentos `Arg1` y `Arg2` se utilizan en relación con algunos elementos del gráfico. Por ejemplo, el eje del gráfico (`Element=21`) puede especificarse como eje principal (`Arg1=0`) o eje secundario (`Arg1=1`), mientras que el tipo de eje se especifica mediante `Arg2`, que puede ser uno de los siguientes valores.

- `0` – Eje de la categoría.
- `1` – Eje del valor.
- `3` – Eje de la serie.

7.1.4 Chart_Calculate()

Este evento ocurre cuando el usuario traza datos nuevos o modificados en el gráfico.

7.1.5 Chart_BeforeRightClick()

Este evento ocurre cuando el usuario hace clic con el botón derecho del ratón sobre el gráfico.

7.1.6 Chart_MouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal x As Long, ByVal y As Long)

Este evento ocurre cuando se presiona el botón del ratón mientras el puntero está sobre un gráfico. El argumento `Button` determina qué botón del ratón fue presionado (evento `MouseDown`) o soltado (evento `MouseUp`):

- `1` – Botón izquierdo.
- `2` – Botón derecho.
- `3` – Botón central.

El argumento `Shift` especifica el estado de las teclas Mayús, Ctrl y Alt:

- `1` – Se pulsó Mayúsc.
- `2` – Se pulsó Ctrl.
- `4` – Se pulsó Alt.

La `x` y la `y` especifican las coordenadas del puntero del ratón.

1. En el editor de VBA del archivo Capítulo 15 – Eventos de gráfico.xlsm, haz clic en el objeto que representa la hoja de gráfico (Gráfico 1).
2. Introduce los siguientes procedimientos en la ventana **Código**:

```
Private Sub Chart_Activate()
    MsgBox "Has desactivado la hoja de gráfico."
End Sub
Private Sub Chart_Deactivate()
    MsgBox "Parece que quieres abandonar " _
    & "la hoja de gráfico."
End Sub
Private Sub Chart_Select(ByVal ElementID As Long, _
ByVal Arg1 As Long, ByVal Arg2 As Long)
    If Arg1 <> 0 And Arg2 <> 0 Then
        MsgBox ElementID & ", " & Arg1 & ", " & Arg2
    End If
    If ElementID = 4 Then
        MsgBox "Has seleccionado el título del gráfico."
    ElseIf ElementID = 24 Then
        MsgBox "Has seleccionado la leyenda del gráfico."
    ElseIf ElementID = 12 Then
        MsgBox "Has seleccionado la clave de la leyenda."
    ElseIf ElementID = 13 Then
        MsgBox "Has seleccionado el contenido de la leyenda."
    End If
End Sub
Private Sub Chart_Calculate()
    MsgBox "Los datos en la hoja de cálculo " & vbCrLf _
    & "Han cambiado. El gráfico se ha actualizado."
End Sub
Private Sub Chart_BeforeRightClick(Cancel As Boolean)
    Cancel = True
End Sub
Private Sub Chart_MouseDown(ByVal Button As Long, _
ByVal Shift As Long, ByVal x As Long, ByVal y As Long)
    If Button = 1 Then
```

```
            MsgBox "Has presionado el botón izquierdo del ratón."
        ElseIf Button = 2 Then
            MsgBox "Has presionado el botón derecho del ratón."
        Else
            MsgBox "Has presionado el botón central del ratón."
        End If
    End Sub
```

3. Activa la hoja de gráfico y realiza las acciones que desencadenen los procedimientos de eventos que has escrito. Por ejemplo, haz clic en la leyenda del gráfico y observa que esta acción desencadena dos eventos: `Chart_MouseDown` y `Chart_Select`.

7.2 Procedimientos de eventos para gráficos incrustados

Para capturar los eventos provocados por un gráfico incrustado en una hoja de trabajo, primero debemos crear un nuevo objeto en el módulo utilizando la palabra `WithEvents`.

La palabra `WithEvents` permite especificar una variable de objeto que se utilizará para responder a los eventos desencadenados por un objeto ActiveX. Esta palabra clave solo se puede utilizar en módulos de clase. En el siguiente procedimiento de ejemplo, aprenderemos a utilizar `WithEvents` para capturar el evento `Chart_Activate` para el gráfico incrustado que creamos al principio del capítulo.

1. Activa el editor de VBA del libro Capítulo 15 – Eventos de gráfico.xlsm.
2. Haz clic en el menú **Insertar – Módulo de clase**.
3. Se creará una nueva carpeta que contendrá el nuevo módulo creado. Llámalo `clsGrafico`.
4. Introduce la siguiente línea en la ventana **Código**.

   ```
   Public WithEvents xlChart As Excel.Chart
   ```

   La declaración anterior declara una variable de objeto que representará los eventos generados por el objeto `Chart`.

   La palabra clave `Public` hará que la variable de objeto `xlChart` esté disponible para todos los módulos del proyecto actual. Declarar una variable de objeto usando la palabra clave `WithEvents` expone todos los eventos definidos para ese tipo de objeto en particular. Después de escribir la declaración anterior, la variable objeto `xlChart` se agrega a la lista desplegable **Objeto** en la esquina superior izquierda de la ventana **Código**.

5. Abre el cuadro desplegable **Objeto** y selecciona la variable `xlChart`.
   La ventana **Código** debería mostrar ahora el esqueleto del procedimiento del evento `xlChart_Activate`:

   ```
   Private Sub xlChart_Activate()

   End Sub
   ```

6. Añade tu código VBA al procedimiento de eventos. En este ejemplo añadiremos una declaración para mostrar un cuadro de mensaje. Después de agregar esta declaración, el procedimiento debe verse como el siguiente:

```
Private Sub xlChart_Activate()
    MsgBox "Has activado un gráfico incrustado en la hoja " & _
    ActiveSheet.Name & "."
End Sub
```

   Después de introducir el código del procedimiento de eventos, debemos informar a VBA que vas a utilizarlo (ver paso 7).

7. En la ventana del **Explorador de proyectos** haz doble clic en el objeto **ThisWorkbook**, e introduce en la primera línea de la ventana **Código** la instrucción para crear una nueva instancia de la clase llamada **clsGrafico**:

```
Dim miGrafico As New clsGrafico
```

   Esta instrucción declara una variable de objeto llamada **miGrafico**. Esta variable se referirá al objeto **xlChart** que se encuentra en el módulo de clase **clsGrafico**. La palabra clave **New** le dice a VBA que cree una nueva instancia del objeto especificado. Antes de poder utilizar la variable **miGrafico**, debes escribir un procedimiento que la inicie (ver paso 8).

8. Introduce el siguiente procedimiento en la ventana **Código** del objeto **ThisWorkbook** para iniciar la variable de objeto **miGrafico**.
9. Ejecuta el procedimiento **IniciamiGrafico**.
   Después de ejecutarlo, los procedimientos de eventos introducidos en el módulo de clase **clsGrafico** se activarán en respuesta a un evento. Recuerda que en este momento el módulo de clase **clsGrafico** contiene el procedimiento de eventos **Chart_Activate**. Más adelante puedes escribir en el módulo de clase **clsGrafico** otros procedimientos de eventos para capturar otros eventos del gráfico.
10. Activa la ventana de Excel y haz clic en el gráfico de la Hoja 1. En este momento el procedimiento **xlChart_Activate** que introdujiste en el paso 6 será activado.
11. Guarda y cierra el archivo Capítulo 15 – Eventos de gráfico.xlsm.

8 Eventos reconocidos por el objeto Application

Si queremos que nuestros procedimientos de eventos se ejecuten independientemente del libro que esté activo en ese momento, debemos crear un procedimiento de eventos para el objeto **Application**. Estos procedimientos tienen un ámbito global. Esto significa que el código del procedimiento se ejecutará en respuesta a un determinado evento mientras la aplicación Excel permanezca abierta.

Los eventos para el objeto **Application** se enumeran en la siguiente tabla. De forma similar a un gráfico incrustado, los procedimientos de eventos para un objeto **Application**

requieren que se cree un nuevo objeto utilizando la palabra `WithEvents` en un módulo de clase.

Evento	Acción que desencadena el evento
`AfterCalculate`	Se ha completado un cálculo y no quedan consultas pendientes.
`NewWorkbook`	Se crea un nuevo libro de trabajo.
`ProtectedViewWindowActivate`	Se activa la vista protegida.
`ProtectedViewWindowBeforeClose`	Ocurre antes de abrir una ventana en vista protegida.
`ProtectedViewWindowBeforeEdit`	Ocurre antes de editar una ventana en vista protegida.
`ProtectedViewWindowDeactivate`	Se desactiva una ventana de vista protegida.
`ProtectedViewWindowOpen`	Se abre una ventana de vista protegida.
`ProtectedViewWindowResize`	Se cambia el tamaño de una ventana de vista protegida.
`WorkbookOpen`	Se abre un libro de trabajo.
`WorkbookActivate`	Se activa cualquier libro de trabajo.
`WorkbookDeactivate`	Se desactiva cualquier libro de trabajo abierto.
`WorkbookNewSheet`	Se crea una nueva hoja en un libro abierto.
`WorkbookNewChart`	Se crea un nuevo gráfico en cualquier libro abierto.
`WorkbookBeforeSave`	Se guarda cualquier libro de trabajo abierto.
`WorkbookBeforePrint`	Se imprime cualquier libro de trabajo abierto.
`WorkbookBeforeClose`	Se cierra cualquier libro de trabajo abierto.
`WorkbookAddInInstall`	Se instala un libro de trabajo como complemento.

Evento	Acción que desencadena el evento
WorkbookAddInUninstall	Se desinstala un libro de trabajo que sea un complemento.
WorkbookAfterSave	El libro se guarda.
WorkbookRowsetComplete	El usuario llega hasta el conjunto de registros o invoca el conjunto de registros de una tabla dinámica OLAP.
SheetActivate	Se activa cualquier hoja.
SheetDeactivate	Se desactiva cualquier hoja.
SheetFollowHyperlink	Se hace clic sobre un hipervínculo.
SheetPivotTableAfterValueChanged	Una o varias celdas que forman parte de la tabla dinámica se editan o recalculan.
SheetPivotTableBeforeAllocateChanges	Ocurre antes de aplicar cambios a la tabla dinámica.
SheetPivotTableBeforeCommitChanges	Se hacen cambios en la fuente de datos OLAP.
SheetPivotTableBeforeDiscardChanges	Ocurre antes de descartar los cambios en una tabla dinámica.
SheetPivotTableUpdate	Se actualiza cualquier tabla dinámica.
SheetSelectionChange	Se cambia la selección en cualquier hoja de trabajo excepto en una hoja de gráfico.
SheetChange	Las celdas de cualquier hoja son cambiadas por el usuario o por un enlace externo.
SheetCalculate	Se calcula cualquier hoja de trabajo (o se vuelve a calcular).
SheetBeforeDoubleClick	Se hace doble clic en cualquier hoja de trabajo. Este evento tiene lugar antes de la acción de doble clic predeterminada.
SheetBeforeRightClick	Se hace clic con el botón derecho del ratón sobre cualquier hoja de trabajo. Este evento tiene lugar antes de la acción

Evento	Acción que desencadena el evento
	predeterminada de hacer clic con el botón derecho.
`WindowActivate`	Se activa cualquier ventana del libro de trabajo.
`WindowDeactivate`	Se desactiva cualquier ventana del libro de trabajo.
`WindowResize`	Se cambia el tamaño de cualquier ventana del libro de trabajo.
`WorkbookPivotTableCloseConnection`	Se cierra una conexión de origen de datos externa para cualquier tabla dinámica.
`WorkbookPivotTableOpenConnection`	Se abre una conexión de origen de datos externa para cualquier tabla dinámica.
`WorkbookAfterXmlExport`	Se exporta un archivo XML.
`WorkbookAfterXmlImport`	Se importa un archivo XML o se actualiza una conexión de datos XML.
`WorkbookBeforeXmlExport`	Se va a exportar un archivo XML o se va a actualizar una conexión de datos XML.
`WorkbookBeforeXmlImport`	Se va a importar un archivo XML.
`WorkbookSync`	Se sincroniza con la copia del servidor un libro de trabajo que forme parte del espacio de trabajo Documento.

Probemos algunos de estos eventos a nivel de aplicación:

1. Crea un nuevo libro de Excel y llámalo Capítulo 15 – Eventos de aplicación.xlsm. Guárdalo en la carpeta C:\Archivos Manual VBA.
2. Activa el editor de VBA y haz clic en el proyecto VBA del libro.
3. Haz clic en el menú **Insertar – Módulo de clase**. Llámalo `clsAplicacion`.
4. En el módulo introduce la siguiente declaración:

```
Public WithEvents App As Application
```

5. A continuación, introduce los siguientes procedimientos de eventos:

```
Private Sub App_WorkbookOpen(ByVal Wb As Workbook)
    If Wb.FileFormat = xlCSV Then
        If MsgBox("¿Deseas guardar este archivo " & vbCrLf _
```

```
            & "como libro de Excel?", vbYesNo, _
            "Formato original: " _
            & "archivo delimitado por comas") = vbYes Then
            Wb.SaveAs FileFormat:=xlWorkbookNormal
        End If
    End If
End Sub
Private Sub App_WorkbookBeforeSave(ByVal _
Wb As Workbook, ByVal SaveAsUI As Boolean, _
Cancel As Boolean)
    If Wb.Path <> vbNullString Then
        ActiveWindow.Caption = Wb.FullName & _
        " [Último guardado: " & Time & "]"
    End If
End Sub
Private Sub App_WorkbookBeforePrint(ByVal _
    Wb As Workbook, Cancel As Boolean)
    Wb.PrintOut Copies:=2
End Sub
Private Sub App_WorkbookBeforeClose(ByVal _
Wb As Workbook, Cancel As Boolean)
    Dim r As Integer
    Dim p As Variant

    Sheets.Add
    r = 1
    For Each p In Wb.BuiltinDocumentProperties
        On Error GoTo GestionError
        Cells(r, 1).Value = p.Name & " = " & _
        ActiveWorkbook.BuiltinDocumentProperties _
        .Item(p.Name).Value
        r = r + 1
    Next
    Exit Sub
GestionError:
    Cells(r, 1).Value = p.Name
    Resume Next
End Sub
Private Sub App_SheetSelectionChange(ByVal Sh _
As Object, ByVal Target As Range)
```

```
        If Selection.Count > 1 Or _
        (Selection.Count < 2 And _
        IsEmpty(Target.Value)) Then
            Application.StatusBar = Target.Address
            Else
            Application.StatusBar = Target.Address & _
            "(" & Target.Value & ")"
        End If
    End Sub
    Private Sub App_WindowActivate(ByVal _
        Wb As Workbook, ByVal Wn As Window)
        Wn.DisplayFormulas = True
    End Sub
```

6. Después de haber introducido el código de los procedimientos anteriores en el módulo de clase, haz clic en el menú **Insertar – Módulo** para insertar un módulo estándar en el proyecto actual.
7. En el módulo estándar, crea una nueva instancia de la clase `clsAplicacion` y conecta el objeto ubicado en el módulo de clase con la variable de objeto App que representa el objeto `Application`, como se muestra a continuación:

```
Dim HazEsto As New clsAplicacion
Public Sub IniciaEventosAplicacion()
    Set HazEsto.App = Application
End Sub
```

   Recuerda que en el paso 4 declaraste la variable de objeto `App` para que apuntase al objeto `Application`.
8. Ahora coloca el puntero del ratón dentro del procedimiento `IniciaEventosAplicacion` y presiona **F5** para ejecutarlo.

Como resultado de la ejecución, el objeto `App` en el módulo de clase se referirá a la aplicación Excel. A partir de ahora, cuando se produzca un evento específico se ejecutará el código de los procedimientos de eventos que hayas introducido en el módulo de clase.

Si no deseas que responda a los eventos generados por el objeto `Application`, puedes romper la conexión entre el objeto y su variable introduciendo en un módulo estándar (y luego ejecutándolo) el siguiente procedimiento:

```
Public Sub CancelaEventosApp()
    Set DoThis.App = Nothing
End Sub
```

Al poner la variable de objeto en `Nothing`, se libera la memoria y se rompe la conexión entre la variable de objeto y el objeto al que se refiere la variable. Cuando se ejecuta el procedimiento `CanceleEventosApp`, el código de los procedimientos de eventos en el módulo de clase no se ejecutará automáticamente cuando se produzca un evento específico.

Ahora probaremos a activar los eventos que introdujiste en el módulo de clase:

9. Activa la ventana de Excel. Haz clic en la ficha **Archivo** y a continuación, en **Nuevo**. Selecciona **Libro en blanco** y haz clic en **Crear**.
10. En la ficha **Archivo** haz clic en **Guardar como**. Guarda el libro abierto en el paso anterior como ProbarAntesdeGuardar.xlsx.
11. Escribe cualquier cosa en la Hoja 1 de libro y guárdalo. Observa que Excel escribe el nombre completo de la hoja de trabajo y la hoja en que se guardó el libro por última vez en la barra de título, tal y como se codificó en el procedimiento `WorkbookBeforeSave`. Cada vez que guardes este libro, Excel actualizará la última vez que lo guardaste en la barra de título.
12. Mira el código de los otros eventos y realiza las acciones necesarias para activarlos.
13. Cierra el archivo Capítulo 15 – Eventos de aplicación.xlsm y los demás libros (si están abiertos).

9 Resumen

En este capítulo has adquirido experiencia práctica con los eventos y la programación de eventos con VBA. Estos son conocimientos cruciales, tanto si planeas crear aplicaciones de Excel para que las usen otras personas como si automatizas tus tareas diarias. Excel proporciona muchos eventos a los que se puede responder. Al escribir procedimientos de eventos, puedes cambiar la forma en que los objetos responden a los eventos. Como has comprobado, un procedimiento de eventos puede ser tan simple como una sola línea de código, o más complejos, con código que incluya estructuras de bucles o toma de decisiones.

Cuando ocurre un determinado evento, VBA simplemente ejecutará el procedimiento apropiado en lugar de responder de la manera predeterminada.

Has aprendido que algunos procedimientos de eventos se escriben en módulos estándar mientras que otros requieren que crees un nuevo objeto usando la palabra `WithEvents` en un módulo de clase. También has aprendido que puedes habilitar o deshabilitar eventos usando la propiedad `EnableEvents`.

En el siguiente capítulo se explicará el proceso a seguir para acceder a los cuadros de diálogo de Excel con VBA.

Capítulo 16
Los cuadros de diálogo

En los capítulos 4 y 5 aprendiste a utilizar la función predefinida `InputBox` para recopilar datos del usuario durante la ejecución de una macro. Pero ¿qué ocurre si el procedimiento necesita más datos en tiempo de ejecución?

El usuario puede querer introducir todos los datos de una vez o seleccionar determinados valores de una lista de elementos.

Si tus procedimientos deben recopilar datos, puedes hacerlo:

- Desde los cuadros de diálogo predefinidos.
- Mediante formularios personalizados.

En este capítulo se mostrará el uso de los cuadros de diálogo predefinidos. En el Capítulo 17 aprenderás a diseñar tus propios formularios desde cero.

1 Los cuadros de diálogo de Excel

Antes de comenzar a crear nuestros propios formularios, debemos dedicar algún tiempo a aprender a aprovechar los cuadros de diálogo existentes en Excel y que, por lo tanto, están listos para usarse. No hablamos de su configuración ni de nuestra capacidad para seleccionar las opciones apropiadas, sino a llamar a estos cuadros de diálogo desde procedimientos VBA. Excel cuenta con una gran colección de cuadros de diálogo predefinidos, representados por constantes que comienzan con el prefijo `xlDialog`.

Algunos de estos cuadros de diálogo se enumeran en la siguiente tabla. Son objetos de Excel que pertenecen a la colección de cuadros de diálogo:

Cuadro de diálogo	Constante
Nuevo	xlDialogNew
Abrir	xlDialogOpen

Cuadro de diálogo	Constante
Guardar como	xlDialogSaveAs
Configurar página	xlDialogPageSetup
Imprimir	xlDialogPrint
Fuente	xlDialogFont

Para mostrar un cuadro de diálogo debemos usar el método **Show** con el siguiente formato:

```
Application.Dialogs(constante).Show
```

Por ejemplo, la siguiente instrucción muestra el cuadro de diálogo **Fuente**:

```
Application.Dialogs(xlDialogFont).Show
```

La tabla anterior muestra una lista de constantes que identifican los cuadros de diálogo disponibles en la ventana **Explorador de objetos** después de seleccionar la biblioteca de Excel y buscar **xlDialog**.

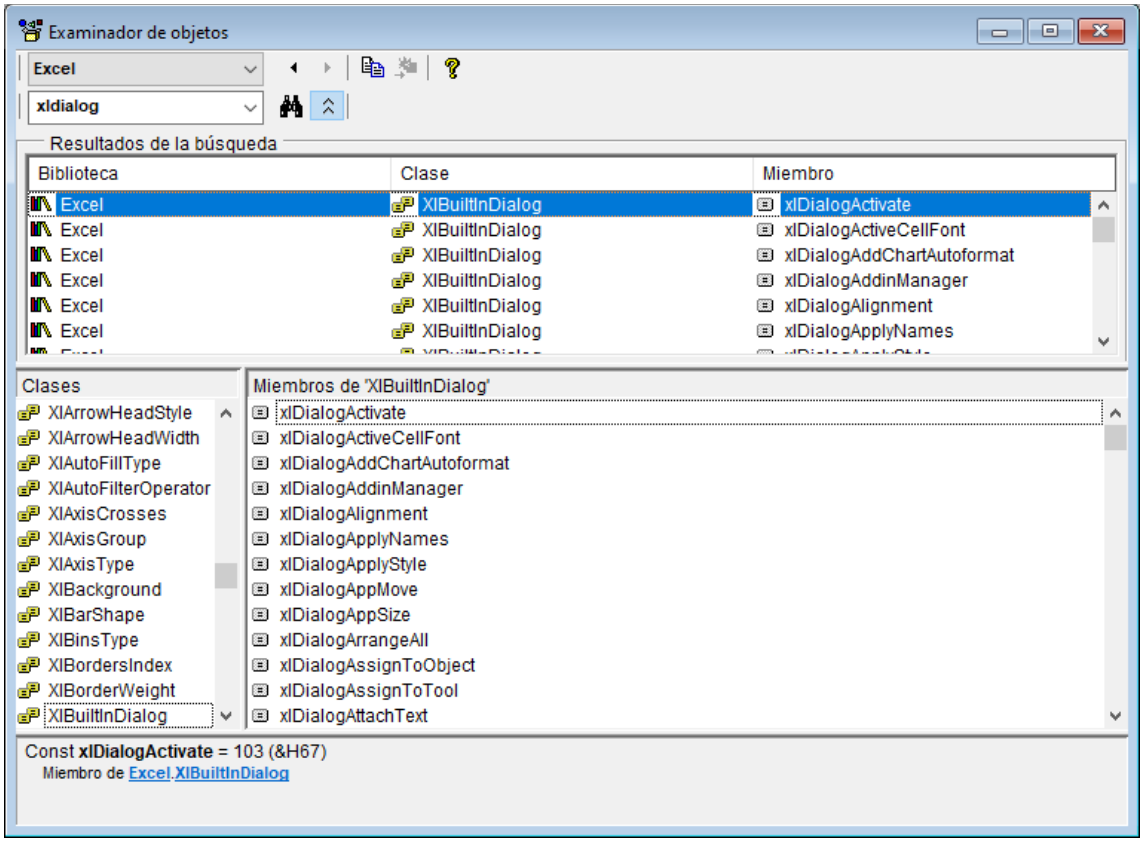


Imagen 1.1 Las constantes con el prefijo "xlDialog" identifican los cuadros de diálogo predefinidos.

Practiquemos mostrando algunos cuadros de diálogo directamente desde la ventana **Inmediato**.

1. Crea un libro nuevo y guárdalo en la carpeta C:\Archivos Manual VBA con el nombre Capítulo 16 – Cuadros de diálogo.xlsm.
2. Dirígete al editor de VBA y haz que se muestre la ventana **Inmediato**.
3. En la ventana **Inmediato** introduce la siguiente instrucción:

   ```
   Application.Dialogs(xlDialogFont).Show
   ```

   La instrucción anterior muestra el cuadro de diálogo **Fuentes**.

   Después de mostrar un cuadro de diálogo puedes seleccionar la opción apropiada, y Excel dará formato a la celda o rango seleccionado. Aunque no puedes modificar el aspecto o comportamiento de un cuadro de diálogo predefinido, puedes decidir qué configuración aparecerá al mostrarse en la pantalla. Si no modificas la configuración inicial, VBA mostrará el cuadro de diálogo con su configuración habitual.

4. Presiona **Cancelar** para salir el cuadro de diálogo.
5. En la ventana **Inmediato**, escribe la siguiente declaración y presiona **Intro**:

   ```
   Application.Dialogs(xlDialogFontProperties).Show
   ```

   Esta instrucción muestra el cuadro de diálogo **Formato de celdas** con la pestaña **Fuente** activa.

6. Presiona **Cancelar** para cerrar el cuadro.
7. En la ventana **Inmediato** escribe la siguiente instrucción y presiona **Intro**:

   ```
   Application.Dialogs(xlDialogDefineName).Show
   ```

   La instrucción anterior muestra el cuadro de diálogo **Definir nombre** donde se puede crear un nombre para una celda o rango de celdas.

8. Presiona Cerrar en el cuadro de diálogo **Definir nombre**.
9. En la ventana **Inmediato** escribe la siguiente declaración y presiona **Intro**:

   ```
   Application.Dialogs(xlDialogOptionsView).Show
   ```

Esta instrucción abre el cuadro de diálogo **Opciones de Excel** mostrando la pestaña **Avanzadas**, como se muestra en la Imagen 1.1.

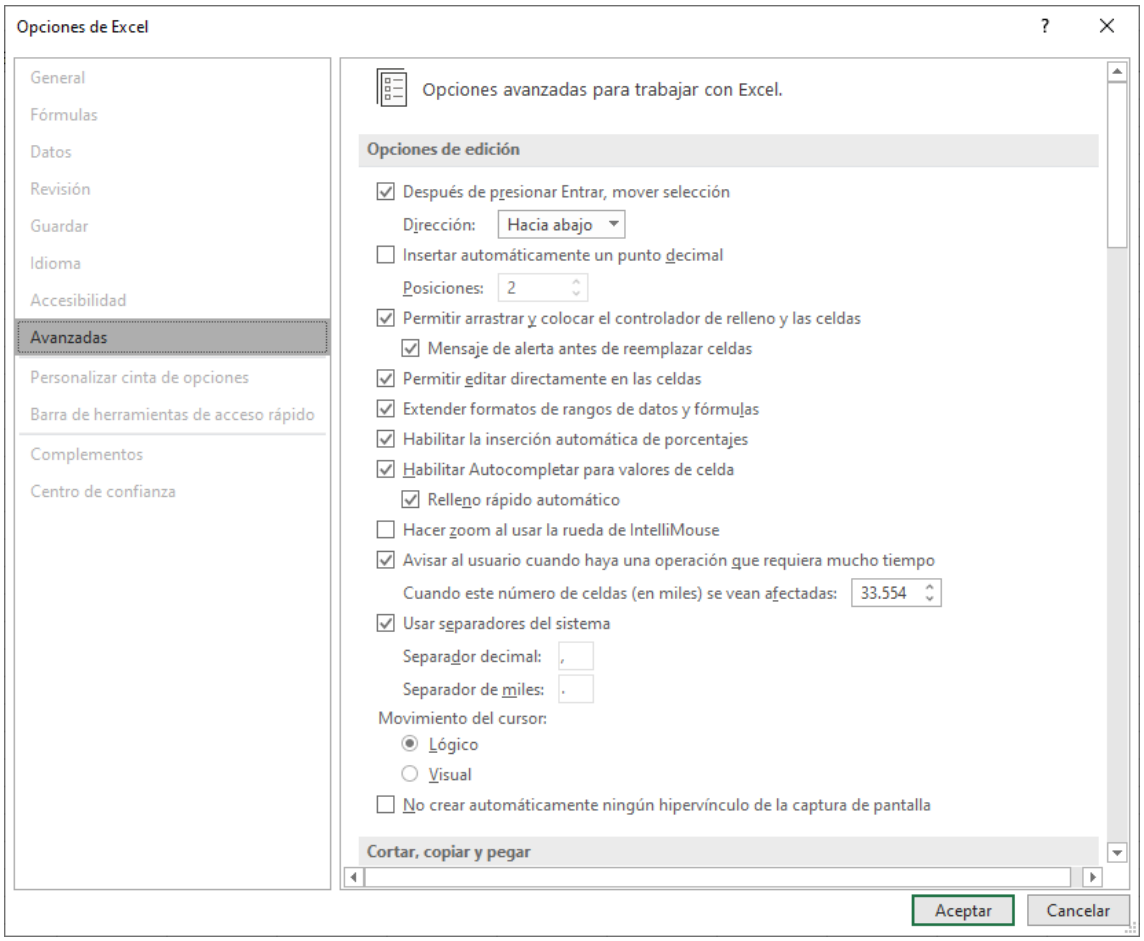


Imagen 1.2 La configuración avanzada del cuadro de diálogo Opciones de Excel se identifica con la constante xlDialogOptionsView.

10. Presiona **Cancelar** para salir del cuadro de diálogo.
11. A continuación, escribe la siguiente instrucción y presiona **Intro**:

```
Application.Dialogs(xlDialogClear).Show
```

Excel muestra el cuadro de diálogo **Borrar** con cinco botones de opción: Todo, Formatos, Contenido, Comentarios e Hipervínculos. De forma predeterminada, el botón **Contenido** está seleccionado cuando Excel muestra el cuadro de diálogo. Pero ¿y si quieres llamar al cuadro con una opción diferente como predeterminada? Para hacer esto, puedes incluir una lista de argumentos. Los argumentos se introducen después del método `Show`. Por ejemplo, para mostrar el cuadro de diálogo **Borrar** con el primer botón de opción seleccionado, debes introducir la siguiente instrucción:

```
Application.Dialogs(xlDialogClear).Show 1
```

Excel suele numerar las opciones disponibles. Por lo tanto Todo=1, Formato=2, Contenido=3, Comentarios=4 e Hipervínculos=5.

Las listas de argumentos del cuadro de diálogo están disponibles en el siguiente enlace:

https://docs.microsoft.com/es-es/office/vba/excel/Concepts/Controls-DialogBoxes-Forms/built-in-dialog-box-argument-lists

12. Presiona **Cancelar** para cerrar el cuadro de diálogo **Borrar**.
13. Para mostrar el cuadro de diálogo **Fuentes**, en el que ya esté seleccionada la fuente Arial de 14 puntos, escribe la siguiente instrucción en la ventana Inmediato y presiona **Intro**:

```
Application.Dialogs(xlDialogFont).Show "Arial", 14
```

14. Presiona **Cancelar** para cerrar el cuadro de diálogo.
15. Para especificar solo el tamaño de la fuente, introduce una coma en la posición del primer argumento.

```
Application.Dialogs(xlDialogFont).Show , 8
```

16. Presiona **Cancelar** para salir del cuadro de diálogo.
17. Escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro**:

```
Application.Dialogs(xlDialogDefineName).Show "Manuel", "=$A$1"
```

La declaración anterior muestra el cuadro de diálogo **Definir nombre**, con el nombre Manuel en el cuadro de texto **Nombres en el libro** y la celda A1 en el cuadro **Se refiere a**. El método `Show` devuelve `True` si haces clic en Aceptar y `False` si cancelas.

18. Presiona **Cerrar** para salir del cuadro de diálogo **Definir nombre**.

2 Los cuadros de diálogo Abrir y Guardar como

`FileDialog` es un objeto de cuadros de diálogo muy potente, pues nos permite mostrar los cuadros de diálogo **Abrir archivo** y **Guardar como** en nuestros procedimientos VBA. Dado que el objeto `FileDialog` forma parte de la biblioteca de objetos de Microsoft Office, está disponible en todas las aplicaciones de Office. También podemos utilizar dos métodos del objeto `Application` (`GetOpenFilename` y `GetSaveAsFilename`) para mostrar los cuadros de diálogo correspondientes sin abrir ni guardar ningún archivo (esto se tratará más adelante en el capítulo). Practiquemos el uso del objeto `FileDialog` en la ventana **Inmediato**.

1. Para mostrar el cuadro de diálogo **Abrir archivo**, introduce la siguiente instrucción en la ventana **Inmediato** y presiona **Intro**:

```
Application.FileDialog(msoFileDialogOpen).Show
```

2. Presiona **Cancelar** para cerrar el cuadro de diálogo.
3. Para mostrar el cuadro de diálogo **Guardar como**, escribe la siguiente instrucción y presiona **Intro**:

```
Application.FileDialog(msoFileDialogSaveAs).Show
```

4. Presiona **Cancelar** para salir del cuadro.

Además de los cuadros de diálogo **Abrir archivo** y **Guardar como**, el objeto `FileDialog` es capaz de mostrar un cuadro de diálogo con una lista de archivos y carpetas. Veámoslos

5. Escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro**:

```
Application.FileDialog(msoFileDialogFilePicker).Show
```

6. Presiona **Cancelar** para cerrar el cuadro de diálogo.
7. Escribe la siguiente instrucción en la ventana **Inmediato** y presiona **Intro**:

```
Application.FileDialog(msoFileDialogFolderPicker).Show
```

   Excel mostrará un cuadro de diálogo con una lista de carpetas.
8. Presiona **Cancelar** para cerrar el cuadro de diálogo.

Las constantes que utiliza el objeto `FileDialog` se enumeran en la siguiente tabla. El prefijo "`mso`" indica que la constante es parte del modelo de objetos de Microsoft Office.

Constante	Valor
msoFileDialogOpen	1
msoFileDialogSaveAs	2
msoFileDialogFilePicker	3
msoFileDialogFolderPicker	4

3 Filtrar archivos

Al seleccionar **Archivo > Abrir**, Excel muestra el cuadro de diálogo **Abrir** con una lista de todos los archivos de Excel. Podemos controlar los tipos de archivo que se muestran en esta ventana mediante el cuadro desplegable situado debajo del cuadro de texto **Nombre del archivo**, o podemos hacerlo mediante de programación, usando la propiedad `Filters`. Si el filtro que necesitamos no aparece en el cuadro de diálogo **Abrir**, podemos agregarlos a la lista de filtros. Los filtros se almacenan en la colección `FileDialogFilters` del objeto `FileDialog`.

En los siguientes ejemplos, crearemos un procedimiento sencillo que devuelve la lista de archivos predeterminados de una hoja de trabajo de Excel:

1. En el **Explorador de proyectos** del editor de VBA, haz clic en el nombre del proyecto y cámbiale el nombre a `CuadrosDialogo`.
2. Inserta un módulo nuevo y llámalo `Cuadros`.
3. Introduce el siguiente procedimiento en la ventana **Código**:

```
Sub ListaFiltros()
    Dim fdf As FileDialogFilters
    Dim fltr As FileDialogFilter
    Dim c As Integer

    Set fdf = Application.FileDialog(msoFileDialogOpen).Filters
    Workbooks.Add
    Cells(1, 1).Select
    Selection.Formula = "Extensiones predeterminadas"
```

```
    With fdf
        c = .Count
        For Each fltr In fdf
            Selection.Offset(1, 0).Formula = _
            fltr.Description & _
            ": " & fltr.Extensions
            Selection.Offset(1, 0).Select
        Next
        MsgBox c & " filtros."
    End With
End Sub
```

El procedimiento anterior declara dos variables de objeto. La variable `fdf` devuelve una referencia a la colección `FileDialogFilters` del objeto `FileDialog`, y la variable `fltr` almacena una referencia al objeto `FileDialogFilter`.

La propiedad `Count` de la colección `FileDialogFilters` devuelve el número total de filtros.

A continuación, el procedimiento se repite a través de la colección `FileDialogFilters` y recupera tanto la descripción como las extensiones de cada filtro definido.

4. Ejecuta el procedimiento `ListaFiltros`.
   Cuando el procedimiento finalice, deberías ver una lista de las extensiones predeterminadas en la hoja de trabajo de un libro nuevo.
   Usando el método `Add` de la colección `FileDialogFilters`, puedes agregar fácilmente tu propio filtro a los filtros predeterminados. El siguiente procedimiento modificado de `ListaFiltros` (ListaFiltros2) muestra cómo añadir un filtro para filtrar los archivos temporales (*.tmp). La última declaración de este procedimiento abrirá el cuadro de diálogo **Abrir archivo** para que puedas comprobar personalmente que el filtro personalizado Archivos temporales (*.tmp) se ha añadido a la lista desplegable.

```
Sub ListaFiltros2()
    Dim fdf As FileDialogFilters
    Dim fltr As FileDialogFilter
    Dim c As Integer

    Set fdf = Application.FileDialog(msoFileDialogOpen).Filters
    Workbooks.Add
    Cells(1, 1).Select
    Selection.Formula = "Lista de filtros predeterminados"
    With fdf
        c = .Count
        For Each fltr In fdf
```

```
                Selection.Offset(1, 0).Formula = _
                fltr.Description & _
                ": " & fltr.Extensions
                Selection.Offset(1, 0).Select
            Next
            MsgBox c & " filtros agregados a la hoja."
            .Add "Archivos temporales", "*.tmp", 1
            c = .Count
            MsgBox "Ahora hay " & c & " filtros." & vbCrLf _
            & "Compruébalo personalmente."
            Application.FileDialog(msoFileDialogOpen).Show
        End With
    End Sub
```

Puedes quitar todos los filtros preestablecidos usando el método `Clear` de la colección `FileDialogFilters`. Por ejemplo, podrías modificar el procedimiento `ListaFiltros2` para borrar los filtros predeterminados antes de añadir el filtro personalizado.

4 Seleccionar archivos

Cuando seleccionamos un archivo en el cuadro de diálogo Abrir archivo, el nombre y la ruta del archivo se colocan en la colección `FileDialogSelectedItems`. Utilizaremos la propiedad `SelectedItems` para devolver la colección `FileDialogSelectedItems`. Al establecer la propiedad `AllowMultiselect` del objeto `FileDialog` en True, un usuario puede seleccionar uno o más archivos manteniendo pulsadas las teclas Mayús o Ctrl mientras hace clic en los nombres de los archivos.

El siguiente procedimiento muestra cómo utilizar las propiedades mencionadas anteriormente. Este procedimiento abrirá un libro nuevo e insertará un control `ListBox`.

El usuario podrá seleccionar más de un archivo. Los archivos seleccionados se cargarán en el control `ListBox` y se resaltará el primer nombre del archivo:

1. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub ListaArchivosSeleccionados()
    Dim fd As FileDialog
    Dim miArchivo As Variant
    Dim lbox As Object

    Application.FileDialog(msoFileDialogOpen).Filters.Clear
    Set fd = Application.FileDialog(msoFileDialogOpen)
    With fd
        .AllowMultiSelect = True
        If .Show Then
```

```
                Workbooks.Add
                Set lbox = Worksheets(1).Shapes. _
                AddFormControl(xlListBox, _
                Left:=20, Top:=60, Height:=40, Width:=300)
                lbox.ControlFormat.MultiSelect = xlNone
                For Each miArchivo In .SelectedItems
                    lbox.ControlFormat.AddItem miArchivo
                Next
                Range("B4").Formula = _
                "Has seleccionados los siguientes " & _
                lbox.ControlFormat.ListCount & " archivos:"
                lbox.ControlFormat.ListIndex = 1
            End If
        End With
    End Sub
```

Este procedimiento utiliza la siguiente declaración para borrar la lista de filtros en el cuadro de diálogo **Abrir archivos** para asegurarse de que solo se enumeran los filtros preestablecidos.

```
Application.FileDialog(msoFileDialogOpen).Filters.Clear
```

A continuación, la referencia al objeto `FileDialog` se almacena en la variable de objeto `fd`:

```
Set fd = Application.FileDialog(msoFileDialogOpen)
```

Antes de mostrar el cuadro de diálogo **Abrir archivo**, debes establecer la propiedad `AllowMultiSelect` en True para que los usuarios puedan seleccionar más de un archivo.

A continuación, el método `Show` se utiliza para mostrar el cuadro de diálogo **Abrir archivo**. Este método no abre los archivos seleccionados por el usuario. Cuando el usuario hace clic en el botón **Abrir**, los nombres de los archivos se recuperan de la colección `SelectedItems` a través de la propiedad `SelectedItems` y se colocan en un cuadro de lista en una hoja de trabajo.

2. Ejecuta el procedimiento `ListaArchivosSeleccionados`. Cuando aparezca en la pantalla el cuadro de diálogo **Abrir archivos**, dirígete a la carpeta C:\Archivos Manual VBA y selecciona dos o tres archivos (manteniendo presionada la tecla Ctrl) y a continuación, haz clic en abrir. Los archivos seleccionados no se abren. El procedimiento simplemente carga los nombres de los archivos seleccionados en un control de lista agregado a la hoja de trabajo (ver Imagen 4.1).

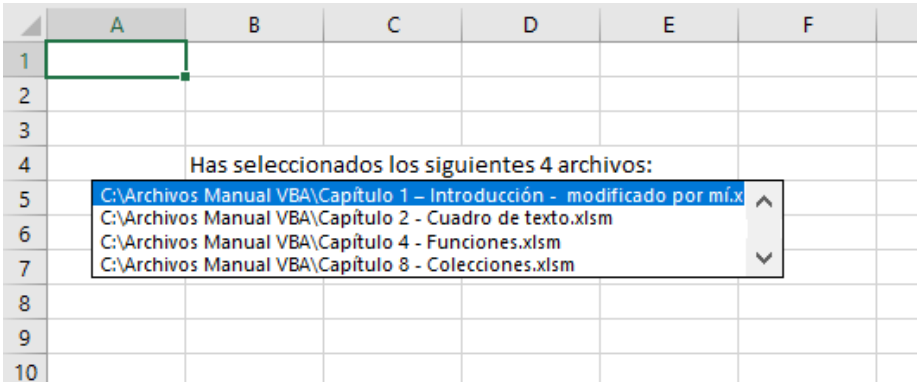


Imagen 4.1 Los archivos seleccionados se cargan en un control ListBox colocado en una hoja de trabajo.

Si deseas realizar inmediatamente la operación de apertura de archivos cuando el usuario hace clic en el botón **Abrir**, debes utilizar el método `Execute` del objeto `FileDialog`. El siguiente procedimiento muestra cómo abrir inmediatamente los archivos seleccionados por el usuario:

```
Sub AbrirArchivos()
    Dim fd As FileDialog
    Dim miArchivo As Variant

    Set fd = Application.FileDialog(msoFileDialogOpen)
    With fd
        .AllowMultiSelect = True
        If .Show Then
            For Each miArchivo In .SelectedItems
                .Execute
            Next
        End If
    End With
End Sub
```

5 Los métodos GetOpenFilename y GetSaveAsFilename

Desde hace muchos años, Excel ofrece a sus programadores dos prácticos métodos de VBA para mostrar los cuadros de diálogo **Abrir archivo** y **Guardar como**: `GetOpenFilename` y `GetSaveAsFilename`. Estos métodos solo están disponibles en Excel y pueden seguir utilizándose si se requiere compatibilidad con versiones anteriores. El método `GetOpenFilename` muestra el cuadro de diálogo **Abrir**, en el que se puede seleccionar el nombre de un archivo para abrirlo. El método `GetSaveAsFilename` muestra el cuadro de diálogo **Guardar como**. Probemos estos dos métodos en la ventana **Inmediato**.

5.1 El método GetOpenFilename

Vamos a abrir un archivo utilizando este método:

1. Escribe la siguiente instrucción en la ventana **Inmediato**:

```
Application.GetOpenFilename
```

La declaración anterior muestra el cuadro de diálogo **Abrir**, donde se puede seleccionar un archivo. El método `GetOpenFilename` obtiene un nombre de archivo de usuario sin abrir el archivo especificado. Este método tiene cuatro argumentos opcionales. Los más frecuentes son el primero y el tercero, que se muestran en la siguiente tabla:

Argumento	Descripción
FileFilter	Determina lo que aparece en la lista desplegable de tipos de archivo.
Title	El título del cuadro de diálogo. Si se omite, el título predeterminado es “Abrir”.

2. Haz clic en **Cancelar** para cerrar el cuadro de diálogo.
3. Para ver cómo se usan los argumentos con el método `GetOpenFilename`, introduce la siguiente declaración en la ventana **Inmediato** (asegúrate de introducirla en una sola línea) y presiona **Intro**:

```
Application.GetOpenFilename("Excel preparado para macros
(*.xlsm),*.xlsm"),,"Selecciona el archivo"
```

Observa que el cuadro de diálogo **Abrir** ahora tiene el título “Selecciona el archivo” en la barra de título. Además, el cuadro desplegable se filtra para mostrar únicamente el tipo de archivo especificado.

4. Haz clic en **Cancelar** para cerrar el cuadro de diálogo. El método `GetOpenFilename` devuelve el nombre del archivo seleccionado o especificado. Este nombre se puede utilizar más adelante en el procedimiento para abrir el archivo. Veamos cómo se hace esto.
5. En la ventana **Inmediato**, escribe la siguiente declaración y presiona **Intro**.

```
tuArchivo = Application.GetOpenFilename
```

Esta instrucción muestra el cuadro de diálogo **Abrir**. El archivo que selecciones mientras el cuadro de diálogo esté abierto se almacenará en la variable `tuArchivo`.

6. Selecciona un archivo de Excel y haz clic en **Abrir**.
Observa que Excel no abrió el archivo seleccionado. Lo único que hizo fue guardar el nombre en la variable `tuArchivo`.
7. Introduce la siguiente instrucción en la ventana **Inmediato**:

```
?tuArchivo
```

Excel muestra el nombre del archivo seleccionado en la ventana **Inmediato**. Ahora que tienes un nombre de archivo, puedes escribir una declaración para abrir realmente el archivo.

8. En la ventana **Inmediato** escribe la siguiente instrucción y presiona **Intro**:

   ```
   Workbooks.Open Filename:=tuArchivo
   ```

   Observa que se abre el archivo especificado.

9. Cierra el archivo que se abrió en el paso anterior.

> **Atención**
>
> El método `GetOpenFilename` devuelve `False` si cancelas el cuadro de diálogo presionando el botón Cancelar o con la tecla Esc.

5.2 El método GetSaveAsFilename

Ahora que sabemos cómo abrir un archivo usando el método `GetOpenFilename`, examinemos un método similar que nos permitirá guardar un archivo. Continuaremos trabajando en la ventana **Inmediato**.

1. Crea un libro nuevo y activa la ventana de VBA.
2. En la ventana **Inmediato** escribe la siguiente instrucción y presiona **Intro**:

   ```
   tuArchivo = Application.GetSaveAsFilename
   ```

   La declaración anterior muestra el cuadro de diálogo **Guardar como**. El nombre del archivo sugerido se introduce automáticamente en el cuadro **Nombre del archivo** en la parte inferior del cuadro de diálogo. El método `GetSaveAsFilename` es conveniente para obtener el nombre del archivo en el que se debe guardar el libro. El nombre de archivo que el usuario introduce en el cuadro **Nombre del archivo** se guardará en la variable `tuArchivo`.

3. Escribe Test1.xlsx en el cuadro **Nombre del archivo** y haz clic en **Guardar**.
   Cuando haces clic en este botón, el método `GetSabeAsFilename` almacenará el nombre de archivo y su ruta en la variable `tuArchivo`. Puedes comprobar el valor de la variable `tuArchivo` en la ventana **Inmediato** introduciendo la siguiente declaración y presionando **Intro**:

   ```
   ?tuArchivo
   ```

Para guardar el archivo, hay que introducir una declaración diferente.

4. En la ventana **Inmediato** escribe la siguiente declaración y presiona **Intro**:

   ```
   ActiveWorkbook.SaveAs tuArchivo
   ```

Ahora el archivo abierto en el paso 1 se ha guardado como Test1.xlsx.

5. Para cerrar el archivo Test1.xlsx escribe la siguiente declaración en la ventana **Inmediato** y presiona **Intro**:

   ```
   Workbooks("Test1.xlsx").Close
   ```

Atención

Como el archivo con el que hemos trabajado no contiene ningún código VBA, lo guardamos con la extensión xlsx en lugar de utilizar el formato de archivo habilitado para macros (xlsm).

Al usar el método **GetSaveAsFilename**, podemos especificar el nombre del archivo, el filtro de archivos y el título personalizado del cuadro de diálogo:

```
tuArchivo = Application.GetSaveAsFilename("Test1.xlsx",
"Archivos de Excel(*.xlsx), *.xlsx",,"Nombre del archivo")
```

6 Resumen

En este capítulo has aprendido a utilizar las declaraciones de VBA para mostrar varios cuadros de diálogo predeterminados. También has aprendido a seleccionar archivos usando el objeto **FileDialog**. El capítulo ha finalizado familiarizándote con dos métodos antiguos para abrir y guardar como, que encontrarás a menudo en procedimientos escritos hace algún tiempo.

En el capítulo siguiente aprenderás a crear y mostrar tus propios cuadros de diálogo gracias a los formularios personalizados.

Capítulo 17
Los formularios personalizados

Aunque ya están listos para usarse y es conveniente utilizarlos, en muchas ocasiones los cuadros de diálogo predefinidos no cumplirán nuestras necesidades. Aparte de mostrar un cuadro de diálogo en la pantalla y especificar su configuración inicial, no podemos controlar el aspecto de este. No podemos decidir qué botones añadir, cuáles quitar y cuáles mover. Tampoco podemos cambiar el tamaño de un cuadro de diálogo predefinido. Por lo tanto, si queremos crear una interfaz personalizada necesitamos crear un formulario personalizado.

1 Crear formularios

Un formulario (o `UserForm`) es similar a un cuadro de diálogo, con la diferencia de que lo podemos personalizar. Podemos agregar varios controles, establecer propiedades para estos controles y escribir procedimientos VBA que respondan a los eventos tanto del formulario como de los controles. Los formularios son objetos que se agregan a un proyecto VBA desde el menú **Insertar – Formulario** del editor de VBA. Pueden ser compartidos entre aplicaciones. Por ejemplo, puedes reutilizar el formulario que diseñaste en Excel para utilizarlo en Word o en cualquier otra aplicación que utilice el editor VBA.

Para crear un formulario personalizado sigue estos pasos:

1. Presiona **Alt + F11** o selecciona la ficha **Programador > Visual Basic** para mostrar el editor de VBA.
2. En el menú del editor, selecciona **Insertar – UserForm**.
   Aparece una carpeta nueva llamada **Formularios** en el **Explorador de proyectos**. Esta carpeta contiene un objeto `UserForm` en blanco. El área de trabajo muestra automáticamente el formulario y el Cuadro de herramientas con las herramientas necesarias para agregar controles (ver la Imagen 1.1).

La ventana **Propiedades** muestra algunas propiedades que se pueden configurar. Para mostrar las propiedades de los formularios por categorías, debemos hacer clic en la pestaña **Por categorías**.

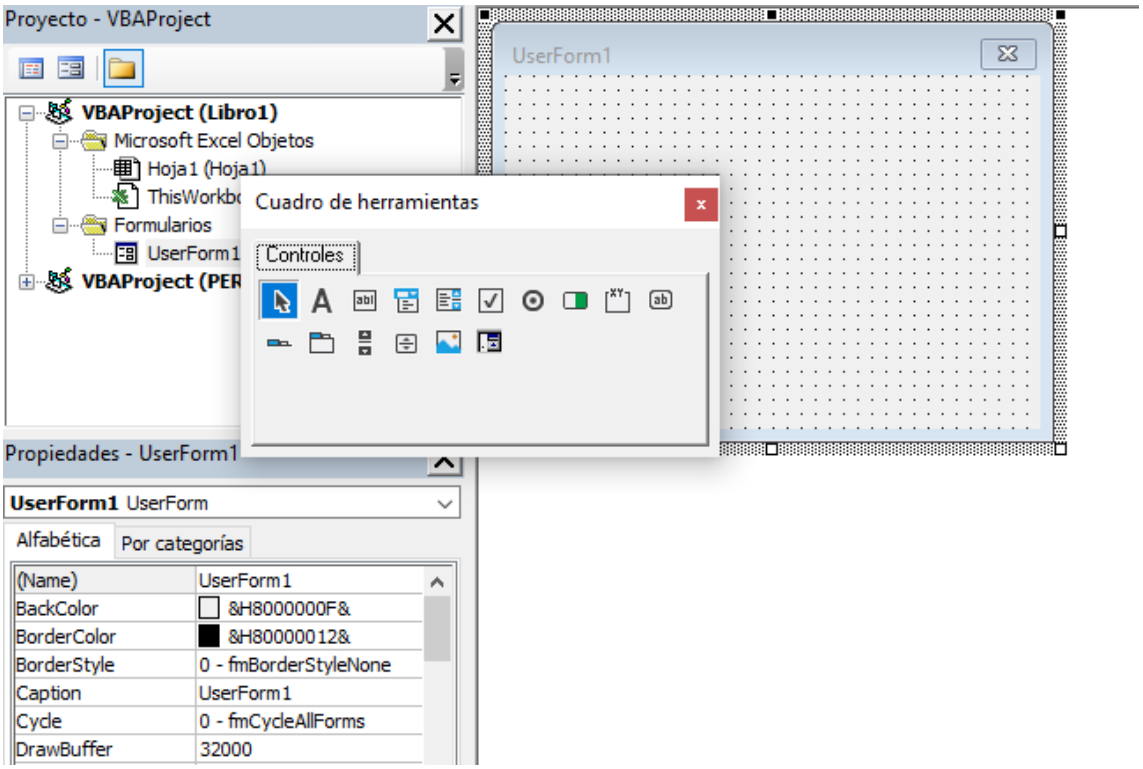


Imagen 1.1 Se puede añadir un formulario al proyecto abierto de VBA seleccionando UserForm en el menú Insertar.

Para obtener información sobre una propiedad específica, hacemos clic en el nombre de la propiedad y presionamos F1. Se abrirá la ayuda en línea con el tema de la descripción de la propiedad.

Después de agregar un formulario al proyecto VBA, debemos asignarle un nombre único estableciendo la propiedad `Name`. También podemos darle un título al formulario desde la propiedad `Caption`.

Todas las aplicaciones que usan el editor de VBA comparten características para crear formularios personalizados. Podemos compartir formularios, exportar e importar archivos de formularios, o arrastrar un objeto de formulario a otro proyecto. Para importar o exportar un formulario, hacemos clic en el menú **Archivo – Importar archivo o Exportar archivo**. Antes de exportar un formulario debemos asegurarnos de que lo tenemos seleccionado en el Explorador de proyectos. Antes de arrastrar un formulario a una aplicación VBA diferente, debemos organizar las ventanas del editor de forma que podamos ver el Explorador en ambas aplicaciones.

1.1 Herramientas para crear formularios

Cuando se diseña un formulario, tenemos que insertar los controles adecuados para que sirva para algo. El **Cuadro de herramientas** (Imagen 1.2) contiene botones estándar de VBA de todos los controles que se pueden añadir a un formulario. También puede contener otros controles instalados en el ordenador. Los controles que se encuentran en el **Cuadro de herramientas** se conocen como controles ActiveX. Estos controles pueden responder a acciones específicas del usuario, como hacer clic o cambiar su valor.

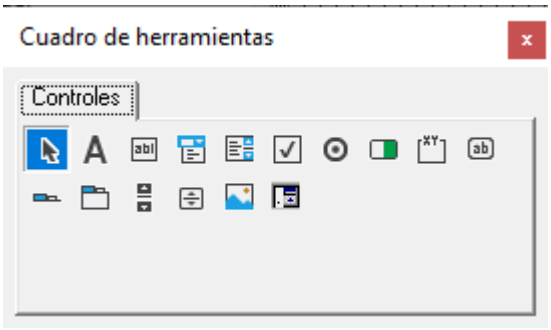


Imagen 1.2 El Cuadro de herramientas muestra los controles que podemos añadir a un formulario.

Aprenderemos a utilizar los controles del **Cuadro de herramientas** a lo largo del capítulo. Si contamos con otras aplicaciones instaladas en nuestro ordenador que también contienen controles ActiveX, también podemos colocarlas en el Cuadro de herramientas. Vamos a agregar un control ActiveX (un selector de fecha y hora) al Cuadro de herramientas.

1. Crea un nuevo libro y llámalo Capítulo 17 – Formularios.xlsm. Guárdalo en la carpeta habitual.
2. Activa el editor de VBA, selecciona el proyecto de VBA y cámbiale el nombre a `FormulariosVBA`.
3. Haz clic en **Insertar – UserForm**.
   Aparecerá un formulario llamado `UserForm1` acompañado del **Cuadro de herramientas**.
4. Haz clic con el botón derecho del ratón fuera de la pestaña **Controles** y selecciona **Nueva página** en el menú contextual.
   Aparece la pestaña Nueva página en el Cuadro de herramientas.
5. Haz clic con el botón derecho del ratón en la nueva pestaña y selecciona **Cambiar nombre**.
6. En el cuadro **Título** escribe “Otros controles”.
7. En el cuadro **Texto de** escribe “Controles ActiveX extra”.
8. Haz clic en **Aceptar** para volver al Cuadro de herramientas.
9. Haz clic con el botón derecho del ratón en cualquier lugar del área de la pestaña nueva y elige **Controles adicionales** en el menú contextual.
10. Cuando aparezca el cuadro de diálogo **Controles adicionales**, haz clic en la casilla de verificación de Microsoft TreeView Control 6.0 o en cualquier otro control de la lista (Imagen 1.3).
11. Haz clic en **Aceptar** para cerrar el cuadro de diálogo Controles adicionales.

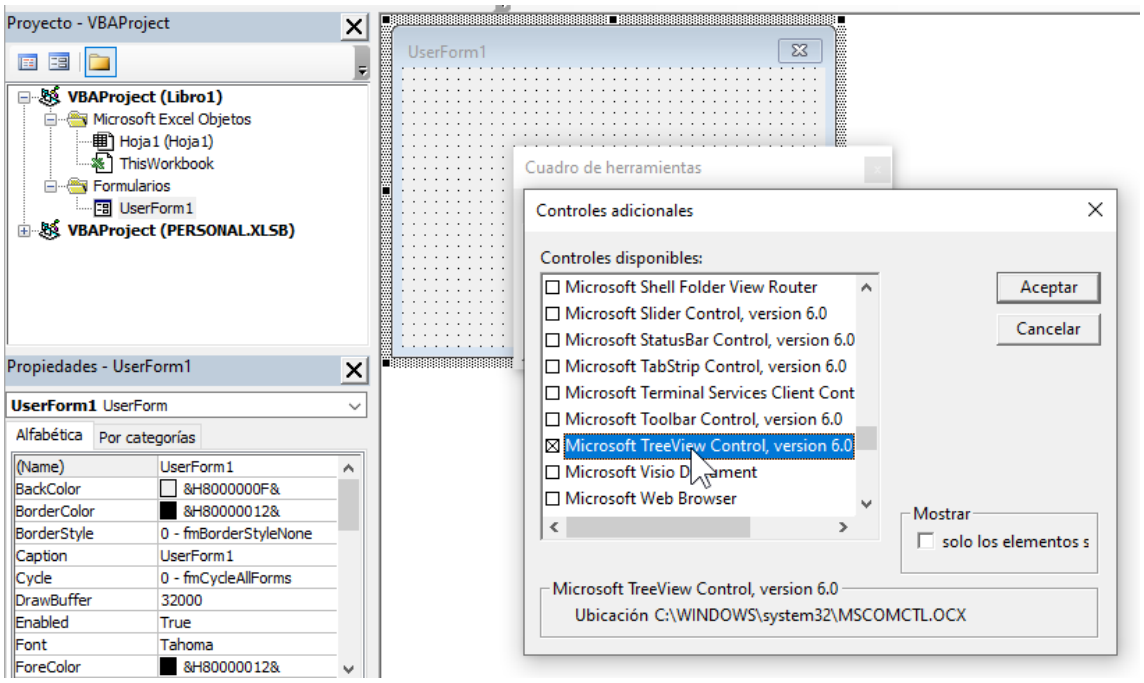


Imagen 1.3 Podemos agregar al Cuadro de herramientas los controles ActiveX que estén instalados en nuestro ordenador.

Los controles estándar suelen ser suficiente para satisfacer las necesidades de nuestros proyectos. Utilizaremos muchos de ellos en los ejercicios de este capítulo:

Seleccionar objetos

Es el único elemento del Cuadro de herramientas que no dibuja un control. Se usa para cambiar el tamaño o mover un control que ya ha sido dibujado en el formulario.

Etiqueta (Label)

Se utiliza a menudo para añadir pies de foto, títulos, encabezados y explicaciones a los formularios. Podemos usar la etiqueta para asignar un título a los controles que no tienen la propiedad `Caption` (por ejemplo, cuadros de texto, cuadros de lista, barras de desplazamiento, etc.). Podemos definir un atajo de teclado para la etiqueta. Por ejemplo, si presionamos Alt y una tecla especificada podemos activar el control.

Cuadro de texto (TextBox)

Pueden utilizarse para mostrar o solicitar datos al usuario. En ellos se puede introducir texto, números, referencias de celdas o fórmulas. Al cambiar la configuración de la propiedad `MultiLine`, podemos introducir más de una línea de texto.

Cuadro combinado (ComboBox)

Combina un cuadro de texto con un cuadro de lista. Se usa a menudo para ahorrar espacio en el formulario. Cuando el usuario hace clic en la flecha hacia abajo ubicada a la derecha del

cuadro, éste se abrirá para revelar una serie de elementos entre los que elegir. El usuario puede introducir un valor nuevo si establecemos la propiedad `MatchRequired` en `False`.

Cuadro de lista (ListBox)

En vez de pedir al usuario que introduzca un valor específico en un cuadro de texto, a veces es mejor presentar una lista de opciones disponibles entre las que seleccionar. El cuadro de lista reduce la posibilidad de errores en la introducción de datos. Los datos de un cuadro de lista pueden mostrarse desde una hoja de trabajo o cargarse directamente de un procedimiento VBA utilizando el método `AddItem`.

Casilla de verificación (CheckBox)

Se usan para activar y desactivar opciones específicas. A diferencia de los botones de opción, podemos seleccionar una o más casillas de verificación.

Botón de opción (OptionButton)

Permite seleccionar una de varias opciones. Suelen aparecer en grupos de dos o más botones rodeados por un control de marco. Solo se puede seleccionar un botón de opción. Cuando se selecciona otro botón diferente, el botón previamente seleccionado se deselecciona automáticamente.

Botón de alternar (ToggleButton)

Se parece a un botón de comando y funciona de manera similar a un botón de opción. Cuando hacemos clic, el botón permanece presionado. El siguiente clic en el botón lo devuelve a su estado normal (sin presionar).

Marco (Frame)

Permiten organizar visualmente y agrupar de forma lógica varios controles del formulario. Veremos un ejemplo más adelante.

Botón de comando (CommandButton)

Lleva a cabo una acción cuando se hace clic sobre él. En este capítulo aprenderemos a ejecutar procedimientos VBA desde los botones de comando.

Barra de tabulaciones (TabStrip)

Aunque tiene muchas similitudes con el control de página múltiple, cada uno tiene una función diferente. Este control permite utilizar los mismos controles para mostrar múltiples conjuntos de los mismos datos. Supongamos que un formulario muestra los exámenes de los estudiantes. Cada alumno tiene que pasar un examen de las mismas materias. Cada asignatura

se puede colocar en una página separada (tab) y cada pestaña contendrá los mismos controles para recopilar datos, como la calificación o la fecha del examen. Al activar cualquier pestaña se verán los mismos controles. Solo cambiarán los datos.

 **Página múltiple (MultiPage)**

Muestra una serie de pestañas en la parte superior del formulario. Cada pestaña actúa como una página diferente. Podemos diseñar formularios que contengan dos o más páginas. Podemos colocar un conjunto de controles diferentes en cada página para que los datos sean más legibles. Es mucho más fácil hacer clic en una pestaña de un formulario que moverse hacia abajo en un formulario largo.

 **Barra de desplacamiento (ScrollBar)**

Aunque habitualmente se utilizan para navegar por las ventanas, también pueden utilizarse en un formulario para introducir valores en un rango predefinido.

 **Botón de número (SpinButton)**

Funciona de forma similar a una barra de desplazamiento. Podemos hacer clic en una flecha para aumentar o disminuir un valor. A menudo va asociado a un cuadro de texto. El usuario puede escribir el valor exacto en el cuadro de texto o seleccionar un valor mediante las flechas.

 **Imagen (Image)**

Permite mostrar una imagen gráfica en el formulario. Este control soporta los siguientes formatos: bmp, cur, gif, ico, jpg y wmf.

 **RefEdit (RefEdit)**

Es específico de los formularios de Excel, ya que permite seleccionar una celda o rango de celdas de una hoja de cálculo y pasarla al procedimiento VBA. Podemos ver cómo funciona este control echando un vistazo a algunos de los cuadros de diálogo predeterminados en Excel, como por ejemplo el cuadro **Consolidar,** al que se accede desde la ficha **Datos**. Este formulario tiene un control llamado **Referencia** que nos permite especificar el rango de datos que queremos consolidar.

1.2 Dibujar controles en el formulario

Cuando se crea un formulario personalizado se dibujan uno o varios controles del Cuadro de herramientas en un formulario vacío. El tipo de control que seleccionemos dependerá del tipo de datos que el control tenga que almacenar y de la funcionalidad del formulario. El Cuadro de herramientas se puede mover por la pantalla. También se puede cambiar su tamaño o cerrarlo cuando ya tengamos todos los controles en el formulario y lo único que se quiere hacer es trabajar con sus propiedades. El Cuadro de herramientas se puede activar o desactivar seleccionando el menú **Ver - Cuadro de herramientas**. Para agregar un control nuevo a un

formulario, primero hacemos clic en la imagen del control y luego, en el formulario, dibujamos un marco. Si hacemos clic en el formulario (sin dibujar el marco), el control se dibujará en su tamaño predeterminado.

Los ajustes de cada control se pueden consultar en la ventana **Propiedades**. Por ejemplo, el tamaño estándar del cuadro de texto es de 72x18 puntos. Después de colocar un control en el formulario, el control Seleccionar objeto (el de la flecha) se convierte en el activo. Si hacemos doble clic en un control del Cuadro de herramientas, podemos dibujar tantas instancias de ese control como queramos. Por ejemplo, para colocar rápidamente tres cuadros de texto, hacemos doble clic en el control Cuadro de texto y a continuación, clic tres veces en el formulario.

1.3 Establecer las opciones de cuadrícula

Cuando dibujamos un control en el formulario, VBA lo ajusta para que se alinee con la cuadrícula del formulario. Podemos ajustar la cuadrícula a nuestro gusto usando el cuadro de diálogo **Opciones**.

Para acceder a las opciones de cuadrícula:

1. Haz clic en el menú **Herramientas – Opciones**.
2. Haz clic en la pestaña **General** del cuadro de diálogo.
   El área **Opciones de la cuadrícula** nos permite desactivar la cuadrícula, ajustar el tamaño de la misma y decidir si queremos que los controles se alineen a la cuadrícula.

2 Aplicación de ejemplo: encuesta para obtención de información

La mejor forma de entender una característica compleja es ponerla en práctica en un proyecto de la vida real. En esta sección crearemos un formulario para agilizar el aburrido proceso de introducir los datos de una encuesta en una hoja de Excel

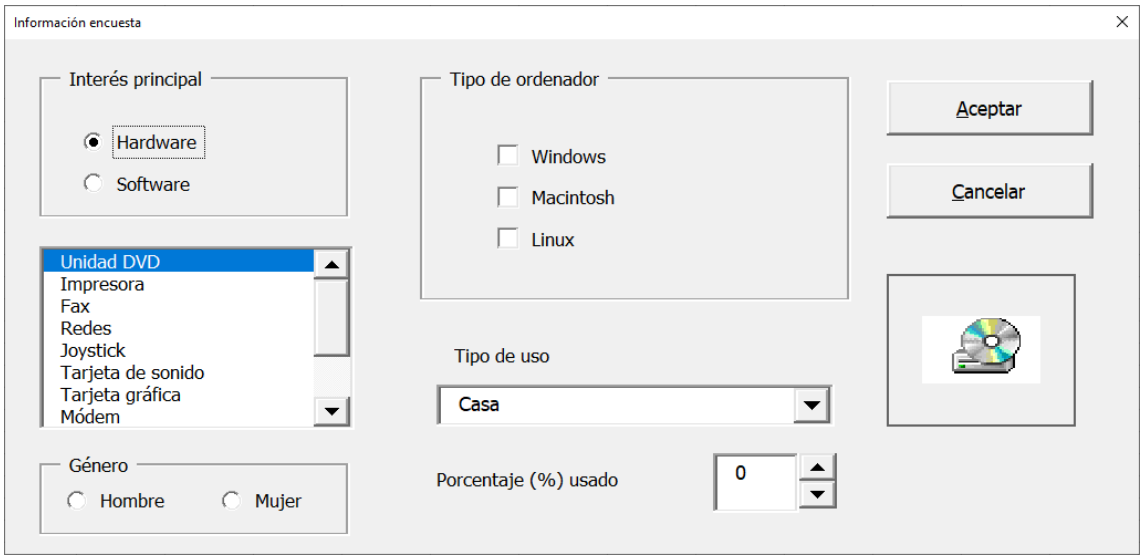
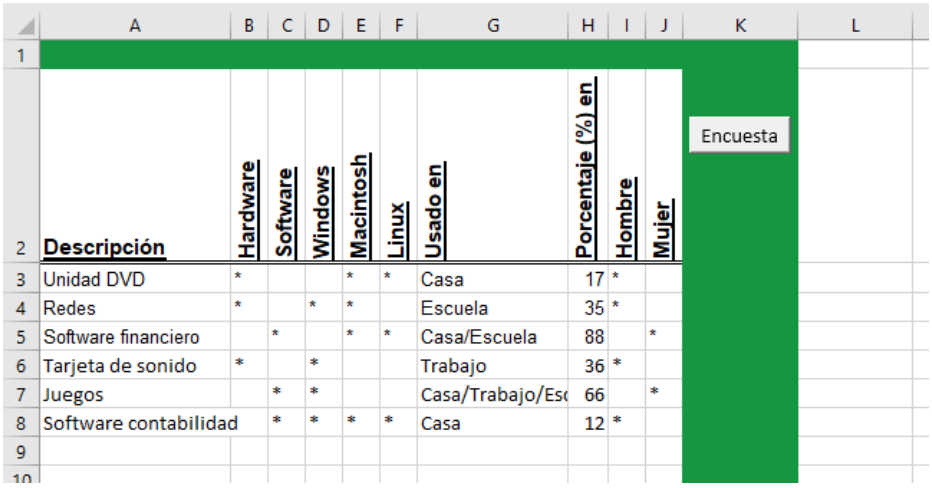


Imagen 2.1 El formulario personalizado permite al usuario introducir rápidamente los datos seleccionándolos de diversos controles colocados en él.



Descripción	Hardware	Software	Windows	Macintosh	Linux	Usado en	Porcentaje (%) en	Hombre	Mujer
Unidad DVD	*			*	*	Casa	17	*	
Redes	*		*	*		Escuela	35	*	
Software financiero		*		*	*	Casa/Escuela	88		*
Tarjeta de sonido	*		*			Trabajo	36	*	
Juegos		*	*			Casa/Trabajo/Es	66		*
Software contabilidad		*	*	*	*	Casa	12	*	

Imagen 2.2 Cada vez que utilizamos el formulario, los datos introducidos se transfieren a una hoja de cálculo.

Mientras vamos desarrollando el proyecto tendremos la oportunidad de experimentar con muchos controles y sus propiedades. También aprenderemos a transferir datos de un formulario a una hoja de cálculo.

2.1 Configuración del formulario

Antes de comenzar a programar, necesitamos desglosar el proyecto en varias tareas:

1. **Paso 1**. Ejercicio 1. Insertar un formulario al proyecto VBA y establecer las propiedades básicas como el nombre y el título, que servirán para identificarlo.
2. **Paso 2**. Ejercicio2. Ajustar el tamaño del formulario para poder colocar fácilmente todos los controles.
3. **Paso 3**. Ejercicio3. Dibujar los controles en el formulario.
4. **Paso 4**. Ejercicio 4. Ajustar otras propiedades del formulario y de los controles.
5. **Paso 5**. Ejercicio 5. Establecer el orden de tabulación de los controles.
6. **Paso 6**. Ejercicio 6. Preparar la hoja que recibirá los datos del formulario.
7. **Paso 7**. Ejercicio 7. Mostrar el formulario finalizado.
8. **Paso 8**. Ejercicio 8. Escribir el procedimiento que inicie el formulario.
9. **Paso 9**. Ejercicio 9. Escribir un procedimiento que rellene el control ListBox.
10. **Paso 10**. Ejercicio 10. Escribir procedimientos que controlen los botones de opción.
11. **Paso 11**. Ejercicio 11. Escribir un procedimiento que sincronice el cuadro de texto con el botón de número.
12. **Paso 12**. Ejercicio 12. Escribir un procedimiento que cierre el formulario.
13. **Paso 13**. Ejercicio 13. Escribir un procedimiento que transfiera los datos del formulario a la hoja.
14. **Paso 14**. Ejercicio 14. Utilizar la aplicación.

2.1.1 Ejercicio 1. Insertar un formulario y establecer las propiedades iniciales

1. Haz clic en **Insertar – UserForm** para añadir un formulario vacío al proyecto **`FormulariosVBA`** (Capítulo 17 – Formularios.xlsm).
2. En la ventana **Propiedades** haz doble clic en la propiedad **`Name`** y escribe **InfoEncuesta**. El nombre se usará más adelante para hacer referencia a este objeto **`UserForm`**.
3. Haz doble clic en la propiedad **`Caption`** y escribe **Información encuesta**.

El nombre Información encuesta aparecerá en la barra de título del formulario.

4. Haz doble clic en la propiedad **BackColor** y a continuación, en la pestaña **Paleta**, selecciona un color para el fondo del formulario.

2.1.2 Ejercicio 2. Ajustar el tamaño del formulario

Cuando se inserta un nuevo formulario, puede que sea demasiado grande o pequeño para albergar todos los controles que vayamos a crear. Es posible cambiar su tamaño utilizando el ratón o estableciendo el tamaño en la ventana **Propiedades**.

Para cambiar el tamaño del formulario con el ratón hacemos clic en una parte vacía del formulario. Observemos que aparecen varios controles de selección alrededor del formulario. Colocamos el puntero del ratón sobre cualquier punto de selección y lo arrastramos hasta la posición deseada.

Si deseamos medidas exactas podemos cambiar el tamaño mediante la ventana **Propiedades**. En este caso, debemos modificar los valores de las propiedades **Height** y **Width**.

1. Haz clic en la barra de título del formulario.
2. En la ventana **Propiedades**, haz doble clic en **Height** y escribe el valor 252.75.
3. Haz doble clic en la propiedad **Width** y escribe el valor 405.75.

2.1.3 Ejercicio 3. Dibujar los controles en el formulario

Ahora estamos listos para proceder a la colocación de los controles necesarios.

La barra de herramientas **UserForm** contiene muchas utilidades para trabajar con formularios, por ejemplo, hacer que los controles tengan el mismo tamaño, centrar un control horizontal o verticalmente, alinear los valores del control y agrupar y desagrupar los controles.

Para mostrar esta barra de herramientas, selecciona **Ver – Barras de herramientas – UserForm**.

1. Haz clic en el control **Marco** del cuadro de herramientas. El puntero del ratón cambia a una cruz.
2. Dibuja un pequeño rectángulo en la parte superior izquierda arrastrando el ratón desde la parte superior izquierda a la parte inferior derecha mientras mantienes el botón del ratón presionado.
   Cuando sueltes el ratón verás el rectángulo con el nombre Frame1. Al seleccionar el cuadro aparecerán varios puntos a su alrededor para modificar el tamaño.
3. En la ventana **Propiedades**, haz doble clic en la propiedad **Caption** y reemplaza el valor que contiene por el de **Interés principal**.
4. Haz clic en el botón de opción del Cuadro de herramientas. A continuación, haz clic dentro del cuadro que acabas de dibujar. Haz clic y arrastra el ratón hasta que veas un rectángulo con la etiqueta predeterminada **OptionButton1**.
5. En la ventana **Propiedades**, cambia la propiedad **Caption** a “Hardware”.
6. Crea otro botón de opción y sitúalo debajo del que acabas de crear. El valor de la propiedad **Caption** será “Software”.

Los botones de opción se utilizan siempre que el usuario debe seleccionar una opción de un grupo de opciones. Si el usuario pudiera seleccionar más de una opción, se utilizarían casillas de verificación.

7. Haz clic en el botón **Cuadro de lista** del Cuadro de herramientas. El cursor del ratón vuelve a ser una cruz.
8. Haz clic debajo del marco que acabas de dibujar y arrastra el ratón hacia abajo y hacia la derecha para crear el cuadro de lista. Al soltar el botón verás un rectángulo en blanco.
9. Inserta un marco nuevo debajo del cuadro de lista. Cambia la propiedad `Caption` a Género. Añade dos botones de opción dentro del marco y cambia sus propiedades `Caption` a Hombre y Mujer.
10. Dibuja un nuevo marco a la derecha del marco **Interés principal**.
11. Cambia la propiedad `Caption` a Tipo de ordenador.
12. Haz clic en el control **Casilla de verificación** del **Cuadro de herramientas** y haz clic dentro del marco vacío que acabas de agregar. El control `CheckBox1` debería aparecer dentro del marco.
13. Cambia la propiedad `Caption` a Windows.
14. Crea dos casillas de verificación más. Cambia sus propiedades `Caption` a Macintosh y Linux respectivamente.
    A diferencia de los botones de opción, que se excluyen mutuamente, las casillas de verificación permiten al usuario activar una o más opciones simultáneamente. La casilla de verificación puede estar marcada, desmarcada o no disponible. Cuando se encuentra no disponible tiene su etiqueta en color gris y, por lo tanto, está inactiva (no puede seleccionarse).
15. Haz clic en el control **Etiqueta** del **Cuadro de herramientas**.
16. Haz clic en el espacio vacío debajo del marco Tipo de ordenador. Debería aparecer el control `Label1`.
17. Cambia la propiedad `Caption` de `Label1` a “Tipo de uso”.
18. Haz clic en el control **Cuadro combinado** en el **Cuadro de herramientas**.
19. Dibuja un rectángulo debajo de la etiqueta Tipo de uso.
    Suelta el botón del ratón. El cuadro combinado muestra una serie de opciones disponibles solo después de hacer clic en la flecha hacia abajo situada a la derecha del control. El cuadro combinado a veces se denomina lista desplegable y se utiliza para ahorrar un espacio valioso en la pantalla. Aunque el usuario solo puede ver un elemento de la lista en un momento dado, la selección actual se puede cambiar rápidamente haciendo clic en el botón de la flecha.
20. Haz clic en el control **Etiquetas** del **Cuadro de herramientas**.
21. Haz clic debajo del cuadro combinado que acabas de dibujar. Cambia la propiedad `Caption` por “Porcentaje (%) utilizado”.
22. Haz clic en el control **Cuadro de texto** del **Cuadro de herramientas**.
23. Haz clic a la derecha de la etiqueta que acabas de crear para colocar ahí el cuadro de texto.
24. Haz clic en el botón **Control de número** en el **Cuadro de herramientas** y a continuación, clic a la derecha del cuadro de texto. Aparecerá un botón de tamaño predeterminado.

El control de número tiene dos flechas que se usan para aumentar o disminuir un valor en un rango determinado. El valor máximo está determinado por la propiedad **`Max`** y el valor mínimo se establece con la propiedad **`Min`**. Este control tiene las mismas propiedades que la barra de desplazamiento, con dos diferencias: no cuenta con la barra de desplazamiento y carece de la propiedad **`LargeChange`**. Habitualmente se coloca junto a un cuadro de texto. Esto permite al usuario introducir un valor directamente en el cuadro de texto o utilizar los botones para determinar el valor. Si el botón de número debe funcionar en conjunto con el cuadro de texto, debes crear un procedimiento para que los dos controles estén sincronizados. En este ejemplo se usará el botón del número para indicar el porcentaje de interés que el usuario tiene en el producto de hardware o software seleccionado.

25. Haz doble clic en el control **Botón de comando** en el **Cuadro de herramientas**. Recuerda que, al hacer doble clic en el control, indicas que quieres crear más de un control con la herramienta seleccionada.
26. Haz clic en la esquina superior derecha del formulario. Esto hará que aparezca el control **`CommandButton1`**.
27. Haz clic justo debajo del botón que acabas de crear. Aparecerá el **`CommandButton2`**.
28. Cambia la propiedad **`Caption`** de **`CommandButton1`** a “Aceptar” y la de **`CommandButton2`** a “Cancelar”.
    La mayoría de los formularios tienen dos botones de comando (Aceptar y Cancelar), que permiten al usuario que acepte los datos introducidos en el formulario o los desestime. En este ejemplo, el botón Aceptar transferirá los datos introducidos en el formulario a una hoja de trabajo. El usuario podrá hacer clic en el botón Cancelar cuando haya terminado de introducir los datos. Para que los botones respondan a las acciones apropiadas, se escribirán dos procedimientos más adelante.
29. Haz clic en el control **Imagen** en el **Cuadro de herramientas**.
30. Haz clic con el ratón debajo del botón Cancelar y arrástralo para dibujar un rectángulo. La imagen se cargará más tarde desde un procedimiento.
31. Haz clic en la barra de título o en cualquier área vacía del formulario para seleccionarla.
32. Presiona **F5** o haz clic en el menú **Ejecutar – Ejecutar Sub/UserForm** para mostrar el formulario tal como lo verá el usuario. VBA cambia a la hoja activa en la ventana de Excel y muestra el formulario que acabas de diseñar.
    Si olvidas seleccionar el formulario, aparecerá el cuadro de diálogo **Macro**. En este caso, cierra el cuadro y repite los pasos 31 y 32.
33. Haz clic en el botón Cerrar (x) en la esquina superior derecha del formulario para volver a VBA. Ten en cuenta que los botones Aceptar y Cancelar del formulario aun no funcionan. Requieren procedimientos VBA que los hagan funcionar.

Una vez que se han agregado los controles al formulario, utilizaremos el ratón o los comandos del menú **Formato** para ajustar la alineación y el espaciado de los controles.

El diseño del formulario **Información encuesta** ya está completo. A partir de ahora ya estás preparado para diseñar cualquier formulario que desees.

Al trabajar con los controles, merece la pena aprender algunos atajos. Por ejemplo, para copiar y mover controles rápidamente:

- Para copiar un control, haz clic en la herramienta **Seleccionar objetos** del **Cuadro de herramientas** y selecciona el control. Mantén pulsada la tecla **Ctrl**, posiciona el puntero del ratón dentro del control y presiona el botón izquierdo. Arrastra el puntero a la posición que desees y suéltalo. Esto cambiará la propiedad `Caption` del control.
- Para seleccionar un grupo completo de controles, haz clic en la herramienta **Seleccionar objetos** y comienza a dibujar un rectángulo alrededor del grupo de controles que deseas mover. Cuando sueltes el botón del ratón, todo los controles estarán seleccionados (también puedes seleccionar más de un control manteniendo pulsada la tecla **Ctrl** mientras haces clic en cada uno de los controles que quieres seleccionar).
- Para mover un grupo de controles a otra posición del formulario, haz clic dentro del área seleccionada y arrastra el ratón a la posición deseada.

2.1.4 Ejercicio 4. Ajustar otras propiedades del formulario y las de los controles

Después de haber colocado los controles en el formulario, y antes de comenzar a crear los procedimientos para controlarlos, debemos asignar nombres a los controles.

Aunque VBA asigna automáticamente un nombre por defecto a cada control (`OptionButton1`, `OptionButton2`, etc.), estos nombres son difíciles de localizar en un procedimiento que puede hacer referencia a objetos de la misma clase que tienen casi nombres idénticos. Asignar nombres significativos a los controles del formulario hace que los procedimientos VBA que hacen referencia a estos controles sean mucho más legibles.

Antes de cambiar la propiedad `Name`, debemos comprobar si la barra de título de la ventana **Propiedades** muestra el control correcto. Por ejemplo, para asignar un nombre nuevo al control de marco, haz clic en el control de marco del formulario. Cuando la ventana **Propiedades** muestre "Propiedades – Frame1", hacemos doble clic en la propiedad `Name` y escribimos el nuevo nombre. No debemos confundir el nombre del control con su título. Por ejemplo, el nombre predeterminado del control de marco es `Frame1`, pero el título de ese control es "Interés principal". El título del control puede cambiarse estableciendo la propiedad `Caption`. El nombre del control permite al usuario identificar su propósito y puede sugerir el tipo de datos que podría devolver. La propiedad `Name` es el nombre que se utilizará en el código de los procedimientos VBA para hacer que sucedan las cosas.

Volvamos a nuestro formulario para hacer ajustes a las propiedades de los controles:

1. Asigna los nombres a los controles del formulario como se muestra a continuación. Para asignar un nombre, sigue estos pasos:
   - Haz clic en el control adecuado en el formulario.
   - Haz doble clic en la propiedad `Name` de la ventana **Propiedades**.
   - Escribe el nombre correspondiente de acuerdo con la siguiente tabla:

Objeto	Nombre de la propiedad
Primer botón de opción	`optHard`
Segundo botón de opción	`optSoft`
Cuadro de lista	`lstSistemas`
Tercer botón de opción	`optHombre`
Cuarto botón de opción	`optMujer`
Primera casilla de verificación	`chkWindows`
Segunda casilla de verificación	`chkMac`
Tercera casilla de verificación	`chkLinux`
Cuadro combinado	`cboUso`
Cuadro de texto	`txtPorcentaje`
Control de número	`spnPorcentaje`
Primer botón de comando	`cmdAceptar`
Segundo botón de comando	`cmdCancelar`
Imagen	`imgImagen`

Los controles que colocaste en el formulario son objetos. Cada uno de estos objetos tiene sus propiedades y sus métodos. De hecho, acabas de cambiar la propiedad `Name` que será referenciada más tarde dentro de los procedimientos. Las propiedades de los controles se pueden establecer durante la fase de diseño (lo que acabas de hacer) o en tiempo de ejecución, es decir, cuando se ejecuta el procedimiento. Ahora vamos a establecer algunas propiedades para los controles seleccionados.

2. Cambia las propiedades del objeto como se muestra en la tabla siguiente.
   Para establecer una propiedad, localízala en la ventana **Propiedades** y escribe el nuevo valor a la derecha del nombre de la propiedad. Por ejemplo, para establecer la propiedad `ControlTipText` del control `lstSistemas`, haz clic en el control **ListBox** del formulario y localiza la propiedad `ControlTipText` en la ventana **Propiedades**. En la columna derecha de la ventana **Propiedades**, escribe el texto que deseas mostrar cuando el usuario posicione el puntero del ratón sobre el control del cuadro de lista: en este caso, selecciona solo un elemento.

Nombre del control	Propiedad	Cambiar a…
lstSistemas	ControlTipText	Selecciona solo un elemento

Nombre del control	Propiedad	Cambiar a…
spnPorcentaje	Max	100
spnPorcentaje	Min	0
cmdAceptar	Accelerator	A
cmdCancelar	Accelerator	C
imgImagen	PictureSizeMode	0-fmPictureSizeModeClip

La propiedad **`Accelerator`** indica qué letra del nombre (Caption) del objeto puede utilizarse para activar el control con la combinación de teclas de acceso directo. La letra especificada aparecerá subrayada en el título del objeto. Después de mostrar el formulario, podrás Aceptar o Cancelar la introducción de datos, utilizando el teclado.

El resto de las propiedades de los objetos del formulario se configurarán directamente desde los procedimientos VBA.

2.1.5 Ejercicio 5. Establecer el orden de tabulación de los controles

El usuario puede moverse por un formulario utilizando el ratón o la tecla **Tab**. Dado que muchos usuarios prefieren navegar utilizando el teclado, es importante determinar el orden en que se activa cada control del formulario. Sigue estos pasos para establecer el orden de los controles en el formulario:

1. En la carpeta **Formularios** (ventana del Explorador de proyectos), haz doble clic en el formulario **`InfoEncuesta`**.
2. En el menú, haz clic en **Ver – Orden de tabulación**. Este cuadro muestra los nombres de todos los controles del formulario en el orden en que fueron agregados. El lado derecho del cuadro tiene botones que permiten mover el control seleccionado hacia arriba o hacia abajo. Para mover un control, haz clic en su nombre y a continuación, en los botones Subir o Bajar hasta que el control aparezca en la posición deseada.
3. Reorganiza el orden de los controles del formulario como se muestra en la Imagen 2.3.

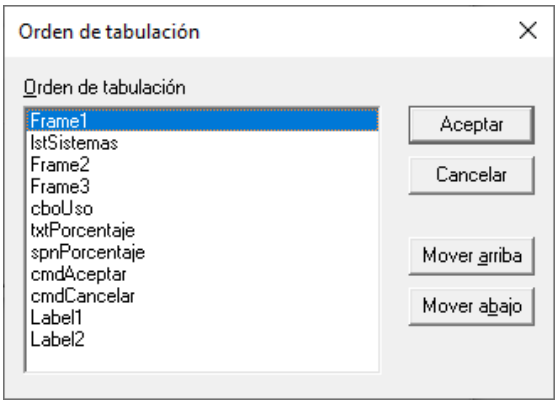


Imagen 2.3 El cuadro de diálogo Orden de tabulación te permite organizar los controles en el orden en el que te gustaría acceder a ellos.

4. Cierra el cuadro de diálogo **Orden de tabulación** haciendo clic en **Aceptar**.
5. Activa el formulario presionando **F5** y comprueba que puedes navegar entre los controles con las teclas Tab (hacia delante) y Mayús + Tab (hacia atrás).
6. Cierra el formulario.
   Si deseas cambiar el orden en el que se activan los controles, vuelve a abrir el cuadro de diálogo Orden de tabulación y realiza los cambios correspondientes.

2.1.6 Ejercicio 6. Preparar la hoja que recibirá los datos del formulario

Después de que el usuario seleccione las opciones apropiadas en el formulario personalizado y haga clic en Aceptar, los datos seleccionados serán transferidos a una hoja de trabajo. Sin embargo, antes de que esto suceda, necesitamos preparar esta hoja para aceptar los datos y dar al usuario una interfaz fácil para lanzar el formulario. Sigamos los pasos que se indican a continuación para preparar la hoja de trabajo:

1. Activa la ventana de Excel.
2. Haz doble clic en el nombre de la hoja Hoja1 del libro Capítulo 17 – Formularios.xlsm y escribe el nombre **Info encuesta**.
3. Introduce los encabezados como se muestra en la Imagen 2.2 vista anteriormente en el capítulo.
4. Selecciona la fila 1 hasta la columna K y cambia el fondo de todas las celdas (cualquier color) usando el botón **Color de relleno** (Ficha Inicio, grupo Fuente). También puedes modificar el color de la columna K completa, como se muestra en la Imagen 2.2.

La forma más fácil de lanzar un formulario personalizado desde una hoja de cálculo es hacer clic en un botón. Los siguientes pasos muestran el proceso de agregar el botón de la encuesta a la hoja de trabajo.

5. Haz clic en la ficha **Programador > Controles > Insertar**.
6. Haz clic en **Botón de formulario**. A continuación, haz clic en la celda K2 para insertar el botón. Cuando aparezca el cuadro de diálogo **Asignar macro**, escribe "`Encuesta`" en el cuadro de nombre de la macro y haz clic en **Aceptar**. Escribirás el procedimiento más tarde.
7. De vuelta en la hoja, verás que se ha creado un botón (Botón). Si se encuentra seleccionado, escribe el nombre que se verá en el botón (Encuesta). En caso de que no esté seleccionado, utiliza el botón derecho del ratón para seleccionarlo. En el menú contextual selecciona **Editar texto** y escribe el nombre.
8. Guarda los cambios que has hecho en el libro Capítulo 17 – Formularios.xlsm.

2.1.7 Ejercicio 7. Mostrar el formulario en pantalla

Los UserForms tienen un método `Show` que permite mostrar el formulario al usuario. En el siguiente ejemplo crearemos el evento `Encuesta`. Recordemos que, en el ejercicio anterior, asignamos el procedimiento `Encuesta` al botón en la hoja de trabajo.

1. En el editor de VBA del libro Capítulo 17 – Formularios.xlsm, haz clic en el menú **Insertar – Módulo**.
2. En la ventana **Propiedades**, cambia el nombre del módulo a `MostrarEncuesta`.
3. Introduce el siguiente procedimiento:

```
Sub Encuesta()
```

```
    InfoEncuesta.Show
End Sub
```

Observa que el método **Show** está precedido por el nombre del objeto **formulario**, tal y como aparece en la carpeta **Formularios del proyecto** de VBA.

4. Guarda los cambios realizados en el libro.
5. Activa la ventana de Excel y haz clic en el botón **Encuesta**. Aparecerá el formulario **InfoEncuesta**.

> ## Atención
>
> Si aparece un mensaje de error al hacer clic en el botón, significa que el nombre de la macro introducida en su creación, no coincide con el nombre de la macro que llama al formulario.
>
> Para corregir el problema, acepta el mensaje, haz clic con el botón derecho del ratón en el botón **Encuesta** y selecciona **Asignar macro** en el menú contextual. En el cuadro de diálogo que aparece, haz clic sobre la macro **Encuesta** y ciérralo.
>
> Vuelve a hacer clic en el botón para mostrar el formulario.

6. Cierra el formulario haciendo clic en la botón Cerrar (x).

Antes de poder utilizar el formulario, necesitamos programar algunos eventos.

2.1.8 Formularios y control de eventos

Además de propiedades y eventos, cada formulario y los controles tienen su propio conjunto predefinido de eventos. Como ya se indicó en un capítulo anterior, un evento es un tipo de acción, por ejemplo, hacer clic en una celda con el ratón, pulsar una tecla, seleccionar un elemento de una lista, etc. Los eventos pueden ser activados por el usuario o por el sistema.

Para especificar cómo debe responder un formulario o control ante un evento, se escriben procedimientos de eventos. Cuando se diseña un formulario, se deben tener en cuenta previamente los eventos que podrían producirse en tiempo de ejecución. El evento más común es el evento **Click**. Cada vez que se hace clic en un botón de comando se desencadena el procedimiento apropiado para responder al evento **Click**.

Un formulario puede responder a más de 20 eventos diferentes. La siguiente tabla enumera los eventos reconocidos por varios controles del formulario. Si un control reconoce un evento específico, la celda de la tabla mostrará un asterisco (*). Familiarízate con los nombres de los eventos. Por ejemplo, echemos un vistazo al evento **AddControl**. Como se puede ver, el evento está solo disponible para tres objetos: **UserForm**, **Frame** y **MultiPage**.

Nombre del evento	UserForm	Label	TextBox	ComboBox	Checkbox	OptionButton	ToggleButton	Frame	CommandButton	TabStripControl	MultiPageControl	ScrollBar	SpinButton	Image	RefEdit
Activate	*														
AddControl	*							*			*				
AfterUpdate			*	*	*	*						*	*		*
BeforeDragOver	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
BeforeDropOrPaste	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
BeforeUpdate			*	*	*	*						*	*		*
Change			*	*	*	*				*	*	*	*		*
Click	*	*		*	*	*	*	*	*	*	*			*	
DblClick	*	*	*	*	*	*	*	*	*	*	*			*	*
Deactivate	*														
DropButtonClick			*	*											*
Enter			*	*	*	*	*	*	*	*	*	*	*		*
Error	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*
Exit			*	*	*	*	*	*	*	*	*	*	*		*
Initialize	*														
KeyDown	*		*	*	*	*	*	*	*	*	*	*	*		*
KeyPress	*		*	*	*	*	*	*	*	*	*	*	*		*
KeyUp	*		*	*	*	*	*	*	*	*	*	*	*		*
Layout	*							*			*				

Nombre del evento	UserForm	Label	TextBox	ComboBox	Checkbox	OptionButton	ToggleButton	Frame	CommandButton	TabStripControl	MultiPageControl	ScrollBar	SpinButton	Image	RefEdit
`MouseDown`	*	*	*	*	*	*	*	*	*	*	*			*	*
`MouseMove`	*	*	*	*	*	*	*	*	*	*	*			*	*
`MouseUp`	*	*	*	*	*	*	*	*	*	*	*			*	*
`QueryClose`	*														
`RemoveControl`	*							*							
`Resize`	*										*				
`Scroll`	*							*			*	*			
`SpinDown`													*		
`SpinUp`													*		
`Terminate`	*														
`Zoom`	*							*			*				

Cada formulario que creamos contiene un módulo de formulario para almacenar los procedimientos de eventos.

Para acceder al módulo e introducir un procedimiento (o conocer los eventos de un determinado control) podemos:

- Hacer doble clic en un control.
- Hacer clic con el botón derecho del ratón sobre el control y seleccionar **Ver código** en el menú contextual.
- Hacer clic en el botón **Ver código** en la ventana del **Explorador de proyectos**.
- Hacer doble clic en cualquier área libre del `UserForm`.

Cuando realizas alguna de las acciones anteriores se abre una ventana **Código** como la que se muestra en la Imagen 2.4. Un módulo de formulario contiene una sección general, así como secciones individuales para cada control colocado en el formulario. La sección general (la de la izquierda) se utiliza para declarar las variables o constantes del formulario.

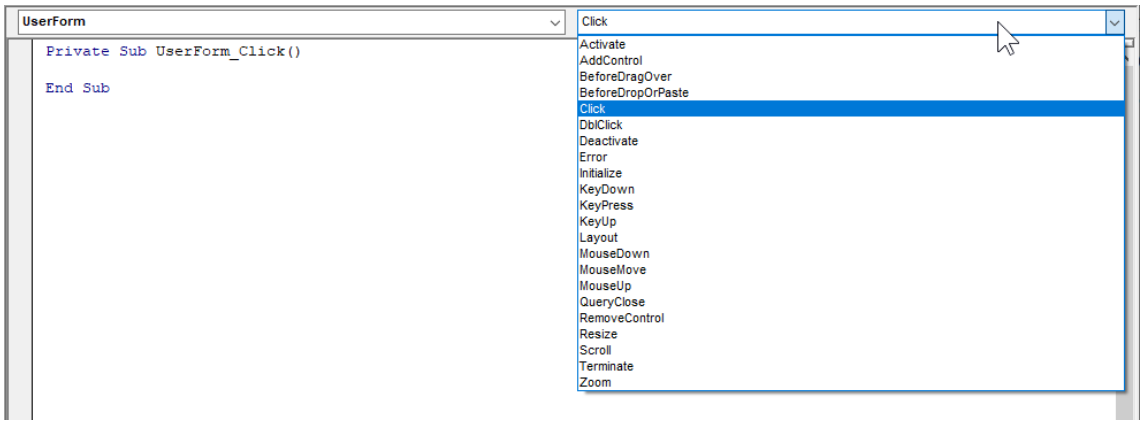


Imagen 2.4 El cuadro combinado de la derecha enumera los procedimientos de eventos disponibles para el objeto seleccionado en la parte izquierda.

Podemos acceder a la sección deseada haciendo clic en la flecha hacia abajo del cuadro combinado de la derecha. El ejemplo de la imagen muestra los eventos reconocidos por el formulario (seleccionado a la izquierda). Los eventos que ya tienen procedimientos escritos aparecen en negrita.

2.2 Escribir los procedimientos de eventos del UserForm y de los controles

Antes de que el usuario pueda hacer tareas específicas con un formulario personalizado, normalmente debemos escribir varios procedimientos. Como se mencionó anteriormente, cada formulario creado cuenta con su propio módulo para almacenar los procedimientos. Antes de mostrar un formulario es posible que deseemos establecer valores iniciales para los controles. Para establecer los valores que tendrán los controles cada vez que se muestre el formulario, escribimos un procedimiento de eventos **`Initialize`** para el **`UserForm`**. El evento **`Initialize`** se produce cuando se carga el formulario, pero antes de que se muestre en la pantalla.

2.2.1 Ejercicio 8. Escribir el procedimiento de inicio del formulario

Supongamos que queremos que el formulario **`InfoEncuesta`** aparezca con los siguientes ajustes iniciales:

- El botón Hardware se encuentre seleccionado.
- El cuadro de lista ubicado debajo contenga los elementos que corresponden a la opción Hardware seleccionada arriba.
- Ninguna de las casillas del cuadro Tipo de ordenador se encuentre seleccionada.
- El cuadro combinado muestre el primer elemento disponible y el usuario no pueda añadir nuevos elementos al cuadro.
- El cuadro de texto muestre el valor inicial de cero (0).
- El control de imagen muestre una imagen relacionada con la opción Hardware seleccionada.

1. En el **Explorador de proyectos** haz doble clic sobre el nombre del formulario **`InfoEncuesta`**.
2. Haz doble clic en algún espacio en blanco del formulario para mostrar la ventana **Código**.

En la definición del procedimiento, VBA añade automáticamente la palabra clave **Private** antes de la palabra **Sub**. Los procedimientos privados solo pueden ser llamados desde el módulo de formulario actual. En otras palabras, un procedimiento que se encuentra en otro módulo del proyecto no puede llamar a este procedimiento en particular (**Private**).

Hay dos cuadros combinados sobre la ventana **Código**. El cuadro de la izquierda muestra los nombres de todos los objetos del formulario. El de la derecha muestra los procedimientos de eventos por el objeto de formulario seleccionado.

3. Haz clic en la flecha hacia abajo en el cuadro combinado de la derecha y selecciona el evento **Initialize**. VBA muestra el procedimiento en la ventana **Código**:

```
Private Sub UserForm_Initialize()


End Sub
```

4. Escribe la configuración inicial del formulario entre las dos líneas anteriores. El procedimiento completo se muestra a continuación:

```
Private Sub UserForm_Initialize()
    ' Se selecciona la opción Hardware
    optHard.Value = True
    'Desactiva la opción Software y las casillas de verificación
    optSoft.Value = False
    chkWindows.Value = False
    chkMac.Value = False
    chkLinux.Value = False
    ' Muestra un 0 en el cuadro de texto
    txtPorcentaje.Value = 0
    ' Llama al procedimiento ListaHardware
    Call ListaHardware
    ' Rellena el cuadro combinado
    With Me.cboUso
        .AddItem "Casa"
        .AddItem "Trabajo"
        .AddItem "Escuela"
        .AddItem "Trabajo/Casa"
        .AddItem "Casa/Escuela"
        .AddItem "Casa/Trabajo/Escuela"
    End With
    ' Selecciona el primer elemento del cuadro combinado
    Me.cboUso.ListIndex = 0
    ' Carga una imagen para la opción Hardware
    Me.imgImagen.Picture = LoadPicture _
```

```
    ("C:\Archivos Manual VBA\cd.bmp")
End Sub
```

Para simplificar el código del procedimiento, puedes utilizar la palabra `Me` en lugar del nombre del formulario real. Por ejemplo, en lugar de usar la instrucción:

```
InfoEncuesta.cboUso.ListIndex = 0
```

Puedes ahorrar tiempo escribiendo:

```
Me.cboUso.ListIndex = 0
```

Esta técnica es especialmente útil cuando el nombre del formulario es largo. Observa también que el primer elemento del cuadro combinado tiene el número de índice cero (0). Por lo tanto, si deseas seleccionar el segundo elemento de la lista, debes establecer la propiedad `ListIndex` en 1.

El procedimiento `UserForm_Initialize` llama al procedimiento externo `ListaHardware` para rellenar el control de lista. El código de este procedimiento se muestra en el siguiente paso.

Observa que el procedimiento finaliza con la carga de una imagen en el control de imagen. Asegúrate de que el archivo con la imagen necesaria puede ser localizado en la carpeta especificada. Si no tienes este archivo, introduce la ruta completa de un archivo de imagen válido que quieras mostrar.

5. Haz doble clic en el módulo `MostrarEncuesta` en el Explorador de proyectos e introduce en la ventana **Código** el procedimiento `ListaHardware` como se muestra a continuación:

```
Sub ListaHardware()
    With InfoEncuesta.lstSistemas
        .AddItem "Unidad DVD"
        .AddItem "Impresora"
        .AddItem "Fax"
        .AddItem "Redes"
        .AddItem "Joystick"
        .AddItem "Tarjeta de sonido"
        .AddItem "Tarjeta gráfica"
        .AddItem "Módem"
        .AddItem "Monitor"
        .AddItem "Ratón"
        .AddItem "Unidad externa"
        .AddItem "Escáner"
    End With
End Sub
```

Ahora que están listos los dos procedimientos, ejecuta el formulario para ver cómo se muestra en pantalla.

6. Ejecuta el formulario haciendo clic en el botón que creaste en la hoja.
   Una vez que se muestra el formulario, el usuario puede seleccionar las opciones adecuadas o hacer clic en el botón Cancelar. Cuando el usuario hace clic en el botón de opción Software, el cuadro de lista debería mostrar elementos diferentes. De la misma forma, el control de imagen debería cargar una imagen diferente. El siguiente ejercicio explica cómo puedes programar estos eventos.

2.2.2 Ejercicio 9. Escribir un procedimiento que rellene el control ListBox

En la sección anterior, creamos el procedimiento **`ListaHardware`** que rellenaba la lista **`lstSistemas`** con los elementos de Hardware. Podemos utilizar el mismo método para cargar los elementos de Software en la lista.

1. Activa el módulo **`MostrarEncuesta`** e introduce el siguiente procedimiento:

```
Sub ListaSoftware()
    With InfoEncuesta.lstSistemas
        .AddItem "Hojas de cálculo"
        .AddItem "Bases de datos"
        .AddItem "Gestión de archivos"
        .AddItem "Procesador de textos"
        .AddItem "Software financiero"
        .AddItem "Juegos"
        .AddItem "Software contabilidad"
        .AddItem "Software diseño"
        .AddItem "Retoque fotográfico"
        .AddItem "CRM"
    End With
End Sub
```

2.2.3 Ejercicio 10. Escribir procedimientos que controlen los botones de opción

Cuando el usuario hace clic en el botón de opción Software, los elementos de Hardware deben ser reemplazados por los de Software y viceversa.

Escribamos los procedimientos que controlarán los botones de Hardware y Software del marco Interés principal.

1. Activa el formulario **`InfoEncuesta`** y haz doble clic en el botón de la opción Software.
2. Cuando aparezca la ventana **Código** con el esqueleto del procedimiento **`optSoft_Click`**, resalta el código y elimínalo.
3. Haz clic en la flecha hacia abajo en el cuadro combinado superior derecho y selecciona el procedimiento **`Change`**. VBA introducirá automáticamente el encabezado y el pie del procedimiento.
4. Introduce el código del procedimiento **`optSoft_Change`** que se muestra a continuación:

```
Private Sub optSoft_Change()
```

```
    Me.lstSistemas.Clear
    Call ListaSoftware
    Me.lstSistemas.ListIndex = 0
    Me.imgImagen.Picture = _
    LoadPicture("C:\Archivos Manual VBA\software.bmp")
End Sub
```

El procedimiento `optSoft_Change` comienza con una declaración que utiliza el método `Clear` para eliminar la lista actual de elementos del control `lstSistemas`. La siguiente declaración llama al procedimiento `ListaSoftware` para rellenar el cuadro de lista con los elementos de Software. En otras palabras, cuando el usuario hace clic en el botón Software, el procedimiento elimina los elementos de Hardware del cuadro de lista y agrega los elementos de Software. Si no borras el cuadro de lista antes de agregar nuevos elementos, los nuevos elementos se agregarán a la lista actual. La instrucción `Me.lstSistemas.ListIndex = 0` selecciona el primer elemento de la lista. La declaración final del procedimiento carga un archivo de imagen al control de imagen. Asegúrate de que el archivo existe en la ruta especificada.

Debido a que el usuario puede querer volver a seleccionar el botón Hardware después de seleccionar el botón Software, debes crear un procedimiento para el evento `Change` similar aldel botón de opción `optHard`.

5. Introduce el siguiente procedimiento justo debajo del procedimiento `optSoft_Change`:

```
Private Sub optHard_Change()
    Me.lstSistemas.Clear
    Call ListaHardware
    Me.lstSistemas.ListIndex = 0
    Me.imgImagen.Picture = _
    LoadPicture("C:\Archivos Manual VBA\cd.bmp")
End Sub
```

6. Lanza el formulario haciendo clic en el botón **Encuesta** en la hoja de cálculo y comprueba el funcionamiento. Cuando hagas clic en el botón de opción Software, deberías ver los elementos de Software en el cuadro de lista de abajo. Al mismo tiempo, el control de imagen debería mostrar la imagen asignada. Después de hacer clic en el botón de opción Hardware, la lista debería mostrar los elementos correspondientes al Hardware. Al mismo tiempo, el control de imagen debería mostrar una imagen diferente.
7. Cierra el formulario haciendo clic en el botón cerrar en la esquina superior derecha del formulario.

2.2.4 Ejercicio 11. Escribir un procedimiento que sincronice el cuadro de texto con el botón de número

El formulario `InfoEncuesta` contiene un cuadro de texto delante de un botón de número. Para indicar el porcentaje de tiempo que se utiliza el elemento de Hardware o Software

seleccionado, el usuario puede escribir un valor en el cuadro de texto o utilizar el botón de número. El valor inicial del cuadro de texto se establece en cero (0). Supongamos que el usuario introdujo 10 en el cuadro de texto y ahora quiere aumentar este valor a 15 utilizando el botón de número. Para habilitar esta acción, el cuadro de texto y el botón de número deben estar sincronizados. Cada objeto requiere un procedimiento de evento `Change` diferente:

1. Haz clic con el botón derecho del ratón en el botón de número y selecciona **Ver código** en el menú contextual.
2. Introduce el siguiente procedimiento como se muestra:

```
Private Sub spnPorcentaje_Change()
    txtPorcentaje.Value = spnPorcentaje.Value
End Sub
```

   Utilizar el botón de número hará que el valor del cuadro de texto suba o baje.
3. En la misma ventana **Código**, introduce el siguiente procedimiento:

```
Private Sub txtPorcentaje_Change()
    Dim Valor As String

    On Error Resume Next
    Valor = Me.txtPorcentaje.Value
    If Valor > 100 Then
        Valor = 0
        Me.txtPorcentaje.Value = Valor
    End If
    spnPorcentaje.Value = txtPorcentaje.Value
End Sub
```

El procedimiento te asegura que solo se pueden introducir valores de 0 a 100 en el cuadro de texto. El procedimiento utiliza la declaración `On Error GoTo Next` para ignorar los errores de introducción de datos. Si el usuario introduce un valor no numérico en el cuadro de texto (o un número mayor de 100), VBA restablecerá el cuadro de texto a cero. Cada vez que se presiona el botón de número, el valor del cuadro de texto se incrementa o disminuye en uno.

2.2.5 Ejercicio 12. Escribir un procedimiento que cierre el formulario

Después de mostrar el formulario, el usuario puede querer cancelarlo pulsando la tecla **Esc** o haciendo clic en el botón **Cancelar**. Para eliminar el formulario de la pantalla crearemos un simple procedimiento que usa el método `Hide`.

```
Private Sub cmdCancelar_Click()
    Me.Hide
End Sub
```

El método `Hide` oculta el objeto, pero no lo elimina de la memoria. De esta manera, el procedimiento puede utilizar los objetos y propiedades del formulario de forma oculta. Utilizamos el método `Unload` para eliminar el formulario tanto de la pantalla como de los recursos de memoria:

```
Unload Me
```

Cuando se descarga el formulario, se recupera toda la memoria asociada a él. El usuario no puede interactuar con el formulario y tampoco puede acceder a sus objetos a través de procedimientos hasta que el formulario sea cargado en la memoria nuevamente usando el método **Load**.

2.2.6 Ejercicio 13. Escribir un procedimiento que transfiera los datos del formulario a la hoja.

Cuando el usuario hace clic en **Aceptar**, los datos introducidos en el formulario se deben escribir en una hoja de trabajo. El usuario puede dejar de utilizar el formulario en cualquier momento haciendo clic en el botón **Cancelar**. Escribamos un procedimiento que copie los datos del formulario en la hoja de trabajo cuando se haga clic en el botón **Aceptar**.

1. En el **Explorador de proyectos** haz clic en el formulario **InfoEncuesta**.
2. Haz doble clic en el botón **Aceptar** e introduce el siguiente procedimiento:

```
Private Sub cmdAceptar_Click()
    Dim r As Integer

    Me.Hide
    r = Application.CountA(Range("A:A"))
    Range("A1").Offset(r + 1, 0) = Me.lstSistemas.Value
        If Me.optHard.Value = True Then
    Range("A1").Offset(r + 1, 1) = "*"
    End If
    If Me.optSoft.Value = True Then
        Range("A1").Offset(r + 1, 2) = "*"
    End If
    If Me.chkWindows.Value = True Then
        Range("A1").Offset(r + 1, 3) = "*"
    End If
    If Me.chkMac.Value = True Then
        Range("A1").Offset(r + 1, 4) = "*"
    End If
    If Me.chkLinux.Value = True Then
        Range("A1").Offset(r + 1, 5) = "*"
    End If
    Range("A1").Offset(r + 1, 6) = Me.cboUso.Value
    Range("A1").Offset(r + 1, 7) = Me.txtPorcentaje.Value
    If Me.optHombre.Value = True Then
        Range("A1").Offset(r + 1, 8) = "*"
    End If
    If Me.optMujer.Value = True Then
        Range("A1").Offset(r + 1, 9) = "*"
```

```
    End If
    Unload Me
End Sub
```

El procedimiento comienza ocultando el formulario. La declaración:

```
r = Application.CountA(Range("A:A"))
```

utiliza la función **CountA** de VBA (CONTARA) para contar el número de celdas que contienen datos en la columna A. El resultado de la función se asigna a la variable **r**.

La siguiente declaración:

```
Range("A1").Offset(r + 1, 0) = Me.lstSistemas.Value
```

Introduce el elemento seleccionado de la lista en una celda situada una fila debajo de la última celda utilizada en la columna A (**r + 1**).

A continuación, hay varias declaraciones condicionales. La primera le indica a VBA que coloque un asterisco en la celda apropiada de la columna B si el botón de opción Hardware está seleccionado. La columna B se encuentra una columna a la derecha de la columna A; de ahí que haya un 1 en la posición del segundo argumento del método **Offset**. El segundo argumento **If** introduce el asterisco en la columna C si el usuario seleccionó el botón de opción Software.

De forma similar, se registran los valores de las casillas de verificación. En la columna G, el procedimiento introducirá el elemento seleccionado en el cuadro combinado Uso. La columna H mostrará el valor introducido en el cuadro de texto Porcentaje (%) usado, y las columnas I y J identificarán el género de la persona que ha rellenado la encuesta.

2.2.7 Ejercicio 14. Utilizar la aplicación

La aplicación está lista para la prueba final. Ahora toca disfrutar del resultado del trabajo desde el punto de vista de un usuario. A medida que vayamos trabajando con el formulario, podemos pensar qué cambios podemos hacer para mejorar la experiencia de usuario.

2.2.8 UserForm: Formularios modales o no modales

De forma predeterminada, el **UserForm** es modal, lo que significa que el usuario no puede interactuar con la aplicación principal mientras el formulario esté visible. La aplicación **InfoEncuesta** que hemos creado en este capítulo se comporta así. Cada vez que hacemos clic en el botón **Encuesta** (en la hoja), el formulario aparece y no podemos interactuar con ninguna otra ventana de Excel hasta que el formulario se cierre. Sin embargo, a veces puede que queramos acceder a otras partes de la aplicación mientras el formulario está activo. Por ejemplo, si estamos creando un formulario de búsqueda personalizado, es posible que los usuarios deban realizar operaciones específicas en Excel fuera de la interfaz del formulario. El formulario no modal nos permitirá hacer precisamente eso. Hacer que un formulario sea no modal es muy sencillo. Simplemente debemos pasar la constante **vbModeless** al método **Show**:

```
Sub Encuesta()
```

```
        InfoEncuesta.Show vbModeless
    End Sub
```

1. Ejecuta el procedimiento modificado `Encuesta` para observar el comportamiento del formulario no modal. Observa que cada vez que haces clic en el botón **Aceptar** los datos del formulario se escriben en la hoja de trabajo y el formulario no se cierra. Tienes control total sobre la hoja de trabajo; incluso puedes borrar filas de datos y volver al formulario para introducir nuevas encuestas.
2. La constante `vbModal` pasada al método `Show` hará que el formulario sea modal, sin embargo, puedes omitirla, pues es la predeterminada.

3 Resumen

En este capítulo hemos programado formularios personalizados desde cero desde cero. A modo de resumen has aprendido a:

- Crear el formulario.
- Insertar los controles necesarios para su funcionamiento.
- Ajustar algunas propiedades del formulario y de los controles.
- Crear los eventos que dotan de funcionalidad al formulario.

En el próximo capítulo, aprenderás a dar formato a las hojas de trabajo.

Capítulo 18
Los formatos con VBA

Microsoft Excel siempre ha contado con un gran número de herramientas para formatos. Aplicando diversas fuentes, colores, bordes y patrones, o usando el formato condicional, podemos transformar fácilmente cualquier dato en bruto en un documento visualmente atractivo y fácil de entender. Incluso si la apariencia de las celdas es lo de menos y nuestro único deseo es hacer un volcado de datos sin más, es probable que antes de compartir el trabajo con otras personas necesitemos modificar los formatos de las celdas.

Para resaltar la información importante aplicamos formatos condicionales con diferentes formatos visuales como barras de datos, escalas de color o conjuntos de iconos. Los minigráficos permiten incrustar pequeños gráficos en una celda para mostrar tendencias de datos. Podemos crear hojas de cálculo con una apariencia profesional aplicando temas y estilos. Cuando seleccionamos un tema, Excel automáticamente modificará el texto, los gráficos, las formas y las tablas para reflejar el tema seleccionado. Con las formas y los gráficos SmartArt podemos dar diferentes efectos artísticos a nuestras hojas de cálculo.

En este capítulo se da por hecho que ya cuentas con algo de experiencia manejando las herramientas anteriores y que sientes curiosidad sobre cómo se pueden aplicar este tipo de formatos de forma automática mediante programación. Así que, pongámonos a trabajar.

1 Tareas básicas de formato con VBA

Esta sección se centra en el formato de número de la celda, el cual debe modificarse siempre antes de dar formato a la apariencia de la propia celda. Siempre debemos comenzar nuestras tareas de formateo comprobando que Excel interpreta correctamente los valores que introdujimos o copiamos de otra fuente. Por ejemplo, cuando copiamos datos de un servidor SQL y estos datos contienen valores de fecha y hora separados por un espacio (por ejemplo 14/05/2020 12:03:05), Excel muestra el valor correcto en la barra de fórmulas, pero en la celda podría mostrarse 14/05/2020 o 12:03. Si en ese momento abrimos el cuadro de diálogo **Formato de celdas**, veremos que Excel ha aplicado un formato personalizado que quizá no es que deseemos.

Cuando introducimos datos manualmente, es más probable que nos demos cuenta inmediatamente de que Excel ha interpretado incorrectamente los datos. Cuando los datos llegan de una fuente externa, es mucho más difícil localizar los problemas de formato a menos que se ejecuten algunos procedimientos VBA personalizados que hagan estas comprobaciones y los arreglen automáticamente para evitar confusiones.

1.1 Formato de número

Dependiendo de cómo hayamos formateado la celda, el número que realmente vemos en la hoja de trabajo puede diferir del que realmente se ha almacenado. Como sabemos, al crear una hoja nueva, Excel da a todas las celdas su formato predeterminado (General). En este formato, Excel elimina los ceros iniciales y finales. Por ejemplo, cuando introducimos 08,40, Excel muestra 8,4. En VBA podemos escribir la siguiente instrucción para que Excel conserve ambos ceros:

```
ActiveCell.NumberFormat = "00.00"
```

La propiedad `NumberFormat` del objeto `CellFormat` se utiliza para establecer o devolver el formato de una celda o rango de celdas específico. El código del formato es la cadena que se muestra en el cuadro de diálogo **Formato de Celdas** (ver Imagen 1.1) o una cadena personalizada.

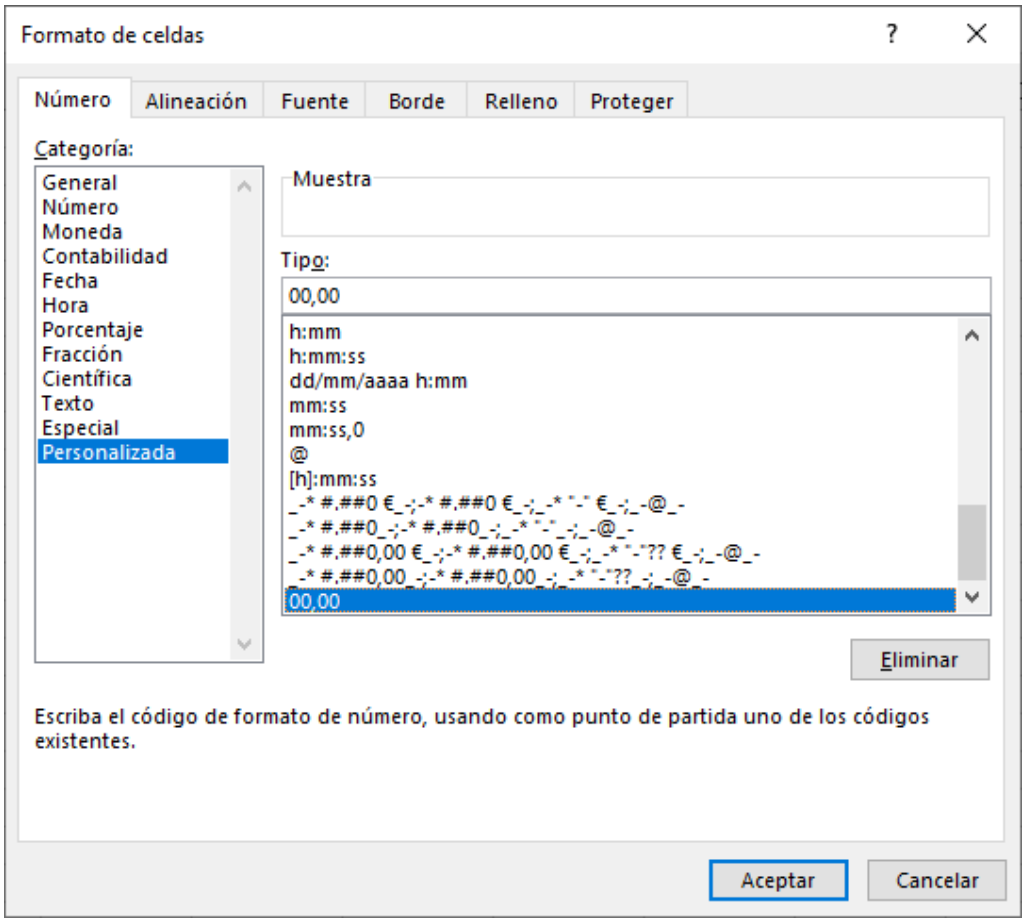


Imagen 1.1 Los códigos de formato de número se pueden encontrar en el cuadro de diálogo Formato de celdas (Ctrl + 1). La categoría Personalizada muestra una lista de formatos de número listos para usar y nos permite adaptarlo a nuestras necesidades particulares.

Si el número introducido tiene decimales, Excel solo mostrará tantos como puedan caber en el ancho de la columna. Por ejemplo, si introducimos 9,5497543154951 en una celda con formato General, Excel muestra 9,5497543 pero almacena el valor completo. Al ensanchar la columna, se ajustará el número de dígitos mostrados. Mientras pensamos en la visualización del número, Excel también determinará si el último dígito visualizado debe redondearse.

Podemos usar la propiedad `NumberFormat` para determinar el formato que Excel aplicó a las celdas. Por ejemplo, para averiguar el formato de la tercera columna de una hoja de trabajo, podemos utilizar la siguiente declaración en la ventana **Inmediato**:

```
?Columns(3).NumberFormat
```

Si todas las celdas de la columna especificada tienen el mismo formato, Excel muestra el nombre del formato (por ejemplo General) o el código de formato que se haya aplicado (por ejemplo #.##0,00 €). Si Excel encuentra un formato diferente en alguna celda del rango, se mostrará `Null` en la ventana **Inmediato**.

Se recomienda aplicar el mismo formato de número para toda la columna. Lo hagamos mediante una macro o de forma manual a través de la interfaz de usuario, el formato se puede aplicar antes o después de introducir los números. Excel solo aplicará el formato de número a las celdas que contengan valores numéricos, por lo tanto, no es necesario preocuparnos si la primera celda de la columna seleccionada contiene un texto de encabezamiento.

Debido a que la propiedad `NumberFormat` establece el formato de número de la celda asignando una cadena de texto con un formato válido desde el cuadro de diálogo **Formato de celdas**, podemos utilizar la grabadora de macros para obtener el código exacto para el formato que queremos aplicar. Por ejemplo, la grabadora de macros guardó las siguientes declaraciones para dar formato a los valores introducidos en las celdas D1 y E1

```
Range("D1").Select
Selection.NumberFormat = "#,##0"
```

y muestra números grandes con el separador de miles y sin decimales.

La siguiente declaración muestra números grandes formateados como moneda, con separador de miles y dos decimales:

```
Range("E1").Select
Selection.NumberFormat = "#,##0.00 €"
```

La categoría **Personalizada** en el cuadro de diálogo **Formato de celdas** (ver Imagen 1.1) enumera muchos formatos personalizados que podemos utilizar con la propiedad `NumberFormat` para controlar cómo se ven los valores en las celdas. Además de estos formatos predeterminados, podemos crear nuestros propios formatos de número personalizados utilizando los códigos de formato que se muestran en la siguiente tabla:

Código	Descripción
0	Marcador de posición de dígito. Lo usamos para forzar un cero. Por ejemplo, para mostrar .5 como 0,50 debemos usar la siguiente declaración:

Código	Descripción
	`Selection.NumberFormat = "0.00"` O introducir 0.00 en el cuadro de texto **Tipo** del cuadro **Formato de celdas**.
#	Marcador de posición de dígito. Lo usamos para indicar la posición en la que se puede colocar el número. Por ejemplo, el código #,### mostrará 12345 como 1,234.
.	Marcador de posición decimal. Debemos tener en cuenta la configuración regional del equipo para determinar si tenemos que utilizar un punto (.) o una coma (,).
,	Separador de miles. Al igual que el punto (.) debemos vigilar cuál es el separador de millares que se aplica en la configuración regional del equipo.
/	Carácter de barra inclinada. Se usa para dar formato a un número como fracción. Por ejemplo para dar formato al número 1,25 como 1¼, utilizaremos la siguiente declaración: `Selection.NumberFormat = "# ?/?"`
_	El carácter de guion bajo se utiliza para alinear los códigos de formato. Por ejemplo, para asegurarnos de que los números positivos y negativos se alineen como se muestra a continuación, se aplica el formato de la siguiente manera: `Selection.NumberFormat = "#,##0.00_);(#,##0.00)"`

*	El asterisco permite rellenar la celda con el carácter que le sigue. Por ejemplo, el siguiente formato produce el siguiente resultado:
	`Selection.NumberFormat = "*_0000"`
	`_____1045`
	`_____23455`

Cuando trabajamos con formatos de celda, debemos tener en cuenta que los formatos de número tienen cuatro partes diferentes separadas por punto y coma. La primera parte se aplica a los números positivos, la segunda a los negativos, la tercera al valor cero y la cuarta al texto. Por ejemplo, echemos un vistazo a la siguiente declaración:

```
Range("A1:A4").NumberFormat = "#,##0;[red](#,#0);""cero"";@"
```

Esta declaración le dice a Excel que dé formato a los números positivos con el separador de miles, que muestre los números negativos en rojo y entre paréntesis, que muestre el texto "cero" siempre que se introduzca un 0, y que dé formato a cualquier texto introducido en la celda como texto. Introduce los siguientes datos en el rango A1:A4:

2870,00
-3456
0
Prueba de texto

Cuando apliquemos el formato anterior a las celdas A1:A4 veremos que se muestra a continuación:

2.870
(3.456)
cero
Prueba de texto

Siguiendo esta pauta, es posible ocultar el contenido de cualquier celda usando la siguiente declaración:

```
Selection.NumberFormat = ";;;"
```

Cuando introducimos tres puntos y coma, Excel oculta la visualización del contenido de la celda tanto en la pantalla como en la impresión en papel. Solo se puede ver su valor real mirando la barra de fórmulas.

Utilizando los códigos de formato de número también es posible aplicar formatos condicionales con una o dos condiciones. Imaginemos el siguiente procedimiento:

```
Sub FormatoRango()
    ActiveSheet.UsedRange.Select
    Selection.SpecialCells(xlCellTypeConstants, 1).Select
    Selection.NumberFormat = _
    "[<150][Red];[>250][Green];[Yellow]"
End Sub
```

Este procedimiento le dice a Excel que seleccione todos los valores en la hoja activa y que los muestre de la siguiente forma:

- Los valores menores de 150 en rojo.
- Los valores mayores de 250 en verde.
- Todos los demás valores entre 150 y 250 en amarillo.

Excel soporta ocho colores comunes: [white], [black], [blue], [cyan], [green], [magenta], [red], y [yellow], así como 56 colores de la paleta de colores predefinidos a los que se puede acceder indicando un número entre 1 y 56. Debemos asegurarnos de utilizar corchetes para encerrar las condiciones y los colores. Más adelante en este capítulo aprenderemos a usar VBA para realizar un formato condicional más avanzado.

Además de la propiedad `NumberFormat`, VBA cuenta con la función `Format` que podemos usar para aplicar un formato específico a una variable. Por ejemplo, echa un vistazo al

siguiente procedimiento que aplica un formato de número antes de introducirlo en una celda de la hoja de cálculo.

```
Sub FormatoVariable()
    Dim miResultado, formatoResultado

    miResultado = "1435.60"
    formatoResultado = Format(miResultado, "Currency")
    Debug.Print formatoResultado
    ActiveSheet.Range("G1").FormulaR1C1 = formatoResultado
End Sub
```

Al ejecutar el procedimiento, la celda G1 de la hoja activa se rellenará con el valor 1.435,60 € (o la moneda del país). La función **Format** especifica la expresión a formatear (en este caso la expresión es el nombre de la variable que almacena un valor específico) y el formato de número que se debe aplicar a la expresión. Se puede utilizar uno de los formatos de número predefinidos como "**General**", "**Currency**", "**Standard**", "**Fixed**", etc. o podemos especificar un formato personalizado utilizando los códigos de la tabla mostrada anteriormente. Por ejemplo, la siguiente declaración aplicará el formato de número al valor almacenado en la variable **miResultado** y asignará el resultado a la variable **formatoResultado**

```
frmResult = Format(myResult, "#.##0.00")
```

Para obtener más información sobre la función **Format** y la lista completa de códigos de formato, podemos acceder a la ayuda online.

Para comprobar si el valor de la celda es un número, utilizamos la función **IsNumber** como se muestra a continuación:

```
MsgBox Application.WorksheetFunction.IsNumber(ActiveCell.Value)
```

Si la celda active contiene un número, Excel devuelve **True**. En caso contrario, devuelve **False**.

Atención

Para utilizar una función de hoja de Excel en VBA, debemos anteponerle el prefijo **Application.WorksheetFunction**. La propiedad **WorksheetFunction** devuelve el objeto **WorksheetFunction**, que contiene las funciones a las que se puede acceder desde VBA.

1.2 Formato de texto

Para dar formato de texto a una celda utilizamos la siguiente declaración VBA:

```
Selection.NumberFormat = "@"
```

Para saber si el valor de una celda es una cadena de texto, usaremos la siguiente declaración:

```
MsgBox Application.WorksheetFunction.IsText(ActiveCell.Value)
```

La función `UCase` se utiliza para convertir el texto de una celda a letras mayúsculas.

```
Range("K3").value = UCase(ActiveCell.Value)
```

Usaremos la función `LCase` para convertir el texto de una celda a minúsculas:

```
If Not Range(ActiveWindow.Selection.Address).HasFormula Then _
ActiveCell.Value = LCase(ActiveCell.Value)
End If
```

De igual forma, podemos utilizar la función `Proper` poner en mayúsculas la primera letra de cada palabra:

```
ActiveCell.Value = _
Application.WorksheetFunction.Proper(ActiveCell.Value)
```

Otra función de texto es `Replace`, que sirve para reemplazar un carácter específico dentro del texto. Por ejemplo, la siguiente declaración reemplaza un espacio por un guion bajo (_) en la celda activa:

```
ActiveCell.Value = Replace(ActiveCell.Value, " ", "_")
```

Para asegurarnos de que el contenido de la celda no contiene espacios por delante o por detrás, utilizaremos las siguientes funciones:

- `LTrim`: Elimina los espacios del principio.
- `RTrim`: Elimina los espacios del final.
- `Trim`: Elimina los espacios del principio y del final.

La siguiente declaración escrita en la ventana **Inmediato** eliminará los espacios que se encuentren al final del contenido de la celda:

```
ActiveCell.value = RTrim(ActiveCell.Value)
```

La propiedad `Font` se usa para dar formato al texto mostrado en la celda. Por ejemplo, la siguiente declaración cambia el tipo de fuente del rango seleccionado a Verdana:

```
Selection.Font.Name = "Verdana"
```

También podemos dar formato a partes del texto en una celda utilizando la colección `Characters`. Por ejemplo, para mostrar el primer carácter del contenido de la celda en color rojo, utilizamos la siguiente declaración:

```
ActiveCell.Characters(1,1).Font.ColorIndex = 3
```

1.3 Formato de fecha

Excel almacena las fechas como números de serie. En el sistema operativo Windows, el número de serie 1 representa al 1 de enero de 1900. Si introducimos el número 1 en una celda de la hoja y le damos formato de fecha, Excel mostrará la fecha 1/1/1900 y almacenará el valor 1 (podemos comprobarlo mirando la categoría General en el cuadro de diálogo Formato de número). Al almacenar las fechas como números de serie, Excel puede realizar fácilmente

cálculos con fechas. Para aplicar un formato de fecha a una celda o rango de celdas se usa la propiedad **NumberFormat** del objeto **Range** así:

```
Range("A1").NumberFormat = "dd/mm/yyyy"
```

Los códigos de formato para fechas se enumeran a continuación:

Código	Descripción
d	Día del mes. Un solo dígito para días del 1 al 9.
dd	Día del mes. Dos dígitos para días del 1 al 9.
ddd	Abreviatura de tres letras del día de la semana.
m	Número del mes del 1 al 12 con un dígito para meses del 1 al 9.
mm	Número del mes del 1 al 12 con dos dígitos para meses del 1 al 9.
mmm	Abreviatura de tres letras del nombre del mes.
yy	Año en dos dígitos.
yyyy	Año en cuatro dígitos
h	Hora. Un solo dígito para horas de 0 a 9.
hh	Hora. Dos dígitos para horas de 0 a 9.
:m	Minutos. Un solo dígito para minutos de 0 a 9.
:mm	Minutos. Dos dígitos para minutos de 0 a 9.
:s :s.0 :s.00	Segundos. Un dígito para segundos de 0 a 9. Para expresarlo en décimas introduce un cero más a la derecha
:ss :ss.0 :ss.00	Segundos. Dos dígitos para segundos de 0 a 9. Para expresarlo en décimas introduce un cero más a la derecha
AM/PM	Formato de hora de 12 horas.

Código	Descripción
am/pm A/P a/p	
[ ]	Se expresan entre corchetes los componentes de tiempo para evitar que Excel cambie de unidad cuando se llegue al máximo. Por ejemplo, para mostrar la hora como 25 horas, 59 minutos y 12 segundos utilizaremos el siguiente código: [hh]:[mm]:ss

El siguiente procedimiento aplica un formato de fecha a los datos cuyo formato sea mm:ss.0 transformándolos en m/dd/yyyy h:mm:ss AM/PM:

```
Sub FormateaFechas()
    Dim wks As Worksheet
    Dim cell As Range

    Set wks = ActiveWorkbook.ActiveSheet
    For Each cell In wks.UsedRange
        If cell.NumberFormat = "mm:ss.0" Then
            cell.NumberFormat = "dd/m/yyyy h:mm:ss AM/PM"
        End If
    Next
End Sub
```

1.4 Formato de columnas y filas

Para acelerar las tareas de formato en las hojas de trabajo, podemos aplicar el formato a filas y a columnas completas en vez de a celdas individuales. La mejor forma de encontrar la declaración que necesitamos es usar la grabadora de macros. Debemos tener en cuenta que Excel registrará más código del necesario y tendremos que limpiarlo antes de copiarlo en el procedimiento VBA. Por ejemplo, a continuación, se muestra el código generado para establecer la alineación a la derecha en los datos de la fila 7.

```
Rows("7:7").Select
Range("D7").Activate
With Selection
    .HorizontalAlignment = xlRight
    .VerticalAlignment = xlBottom
```

```
        .WrapText = False
        .Orientation = 0
        .AddIndent = False
        .IndentLevel = 0
        .ShrinkToFit = False
        .ReadingOrder = xlContext
        .MergeCells = False
    End With
```

Para alinear a la derecha la fila 7 podemos escribir una sola declaración como esta:

```
    Rows(7).HorizontalAlignment = xlRight
```

Podemos utilizar la grabadora de macros para ayudarnos a encontrar los nombres de las propiedades que deben utilizarse para activar o desactivar una función de formato específica. Una vez que conozcamos la propiedad y el ajuste requerido, podemos escribir nuestras propias declaraciones para dar formatos a columnas y filas:

Formato requerido	Declaración
Dar formato de fecha la columna D con la propiedad NumberFormat	`Columns("D").NumberFormat = "mm/dd/yyyy"`
Dar formato de moneda a la columna G	`Columns("G").NumberFormat = "###,##0.00 €"`
Dar formato de moneda a la columna G usando la propiedad Style	`Columns("G").Style = "Currency"`
Establecer el ancho de columna y el alto de fila	`Columns(2).ColumnWidth = 21.5` `Rows(2).RowHeight = 55.55`
Autoajustar tamaño de una columna o fila	`Columns(2).AutoFit` `Rows(2).Autofit`
Aplicar formato Negrita a la primera fila	`Rows(1).Font.Bold = True`
Alinear a la derecha la fila 1	`Rows(1).HorizontalAlignment = xlRight`
Centrar los datos en la columna B	`To center data in column B:` `Columns("B").HorizontalAlignment = xlCenter`

Formato requerido	Declaración
Cambiar el color de fondo de la columna donde se encuentra la celda activa	`Columns(ActiveCell.Column).interior.color = vbYellow`
Comprobar el ancho de la columna de la celda activa.	`MsgBox Columns(ActiveCell.Column).ColumnWidth`
Autoajustar tamaño de todas las columnas y filas	`ActiveSheet.Cells.EntireRow.AutoFit` `ActiveSheet.Cells.EntireColumn.AutoFit`

1.5 Formato de encabezado y pie de página

Los encabezados y pies de página se componen de tres secciones cada uno: encabezado izquierda, encabezado central, encabezado derecha y pie izquierdo, pie central y pie derecho. Con los códigos que se muestran en la siguiente tabla podemos personalizar el encabezado o pie de página según nuestras necesidades.

Código	Descripción
&D	Muestra fecha actual
&T	Muestra hora actual
&F	Muestra nombre del libro
&A	Muestra el nombre de la hoja
&P	Muestra el número de página
&P+número	Muestra el número de página más el número especificado
&P-número	Muestra el número de página menos el número especificado
&N	Muestra el número total de páginas del libro
&Z	Muestra la ruta de ubicación del libro
&G	Inserta una imagen
&&	Muestra un ámpersand
&nn	Muestra los caracteres que siguen en la fuente especificada en puntos
&color	Muestra los caracteres en un color específico

&”nombre de fuente”	Muestra los caracteres en el tipo de fuente específica
&L	Alinea los caracteres a la izquierda
&C	Centra los caracteres
&R	Alinea los caracteres a la derecha
&B	Convierte los caracteres a negrita
&I	Convierte los caracteres a cursiva
&U	Convierte los caracteres a subrayado
&E	Convierte los caracteres a doble subrayado
&S	Tacha los caracteres
&X	Convierte los caracteres a superíndice
&Y	Convierte los caracteres a subíndice

Estos son algunos ejemplos de uso de los códigos de formato de encabezados y pies de página:

Formato	Declaración VBA
Encabezado de dos líneas con texto en negrita y en cursiva	`ActiveSheet.PageSetup.LeftHeader = "&BNombre de la empresa" & Chr(13) & "&INombre del departamento"`
Insertar en el pie, fecha de creación del libro	`ActiveSheet.PageSetup.RightFooter = "Creaado el: " & ActiveWorkbook.BuiltinDocumentProperties ("Fecha de creación")`
Poner contenido de una celda en el encabezado	`ActiveSheet.PageSetup.CenterHeader = ActiveSheet.Cells(2,2).value`
Insertar el nombre del archivo y su ruta en el pie	`ActiveSheet.PageSetup.CenterFooter = ActiveWorkbook.FullName`
Eliminar el texto y el formato del encabezado central	`ActiveSheet.PageSetup.CenterHeader = ""`

1.6 Formato de la apariencia de la celda

Como se mencionó anteriormente, los retoques de estilo (fuentes, color, bordes, sombreado, alineación, etc.) deben realizarse después de aplicar el formato a los números, fechas y horas. El formato de la celda hace que la hoja sea más fácil de leer e interpretar. Además, podemos llamar la atención del lector sobre información particularmente importante.

El objeto **Font** nos sirve para cambiar el formato de la fuente. Podemos aplicar fácilmente múltiples propiedades de formato usando el bloque de instrucciones **With … End With**, como se muestra a continuación:

```
Sub AplicaFormatoCelda()
    With ActiveSheet.Range("A1").Font
        .Name = "Tahoma"
        .FontStyle = "italic"
        .Size = 14
        .Underline = xlUnderlineStyleDouble
        .ColorIndex = 3
    End With
End Sub
```

La propiedad **ColorIndex** hace referencia a los 56 colores disponibles en la paleta de colores. Por ejemplo, el número 3 representa al color rojo. El siguiente procedimiento imprime una paleta de colores en la hoja activa:

```
Sub PaletaColores()
    Dim r As Integer
    Dim c As Integer
    Dim k As Integer

    k = 0
    For r = 1 To 8
        For c = 1 To 7
            Cells(r, c).Select
            k = k + 1
            ActiveCell.Value = k
            With Selection.Interior
                .ColorIndex = k
                .Pattern = xlSolid
            End With
        Next c
    Next r
End Sub
```

Para cambiar el color de fondo de una celda o de un rango de ellas, usamos una de las siguientes declaraciones:

```
Selection.Interior.Color = vbBlue
Selection.Interior.ColorIndex = 5
```

El color de la fuente lo cambiamos de la siguiente forma:

```
Selection.Font.Color = vbMagenta
```

La fuente puede ser Negrita, Cursiva o Subrayada, o una combinación de las tres. Para ahorrar algo de código encerramos los comandos en el bloque **With … End With**:

```
With Selection.Font
    .Italic = True
    .Bold = True
    .Underline = xlUnderlineStyleSingle
End With
```

Para aplicar bordes a celdas y rangos usamos la siguiente declaración:

```
Selection.BorderAround Weight:=xlMedium, ColorIndex:=3
Selection.BorderAround Weight:=xlThin, Color:=vbBlack
```

El método **BorderAround** del objeto **Range** coloca una línea alrededor de los bordes de las celdas seleccionadas. Pueden utilizarse las siguientes constantes para especificar el grosor del borde: **xlHairline**, **xlMedium**. **xlThick** y **xlDash**. Cuando se especifica el color del borde se puede usar **ColorIndex** o **Color** pero no ambos.

En lugar de especificar el grosor del borde, podemos utilizar **LineStyle** como en el siguiente ejemplo:

```
Selection.BorderAround LineStyle:=xlDashDotDot, Color:=vbBlack
```

Podemos utilizar cualquiera de las siguientes constantes de **xlLineStyle**:

- **xlContinuous**
- **xlDash**
- **xlDashDot**
- **xlDashDotDot**
- **xlDot**
- **xlDouble**
- **xlLineStyleNone**
- **xlSlantDashDot**

Si deseamos eliminar el borde de la celda usaremos **xlLineStyleNone**.

Como hemos visto anteriormente con VBA también podemos alinear el contenido de las celdas tanto vertical como horizontalmente:

```
Selection.HorizontalAlignment = xlCenter
Selection.VerticalAlignment = xlTop
```

1.7 Eliminar formatos de celdas

Para eliminar el formato de la celda usaremos el método `CleartFormats` del objeto `Range`. Este método restaura el formato al formato General original sin eliminar el contenido de la celda. Si lo que deseamos es eliminar el contenido, usaremos el método `ClearContents`.

2 Tareas avanzadas con formatos

Vamos a centrarnos ahora en cómo utilizar algunas técnicas avanzadas de formato de datos. Usaremos los formatos disponibles en el grupo **Estilos** de la ficha **Inicio** y en el grupo **Temas** de la ficha **Diseño de página**. Trabajaremos con la colección `FormatConditions` del objeto `Range` y exploraremos las herramientas del formato condicional: barras de datos, escalas de color y conjuntos de iconos. Luego veremos la creación y el uso de Minigráficos para mejorar la apariencia de la hoja de trabajo y la aplicación de temas a los documentos viendo cómo afectan a otros estilos de hoja.

2.1 Formato condicional con VBA

Para ayudarnos a resaltar automáticamente partes importantes de la hoja de trabajo, Excel cuenta con una herramienta conocida como **Formato condicional**. Esta herramienta nos permite establecer una condición (una regla de formato) y especificar el tipo de formato que debe aplicarse a las celdas y rangos cuando se cumple la condición. Por ejemplo, podemos usar el formato condicional para aplicar diferentes colores de fondo, fuentes o bordes a una celda en función de su valor.

El formato condicional proporciona a los usuarios varios tipos de reglas comunes y herramientas de formato como barras de datos, escalas de dolor y conjuntos de iconos que facilitan el resaltado de ciertos datos de la hoja. Por ejemplo, podemos resaltar el 10% superior o inferior de los valores, localizar información duplicada o única, o indicar valores por encima o por debajo de la media.

Es posible especificar un número ilimitado de formatos condicionales y hacer referencia a rangos de otras hojas de trabajo. El **Administrador de reglas de formato condicional** que se muestra en la Imagen 2.1 simplifica la creación, modificación y eliminación de las reglas condicionales. Se puede acceder a todas las funciones de formato condicional disponibles en la interfaz de usuario de Excel desde procedimientos VBA.

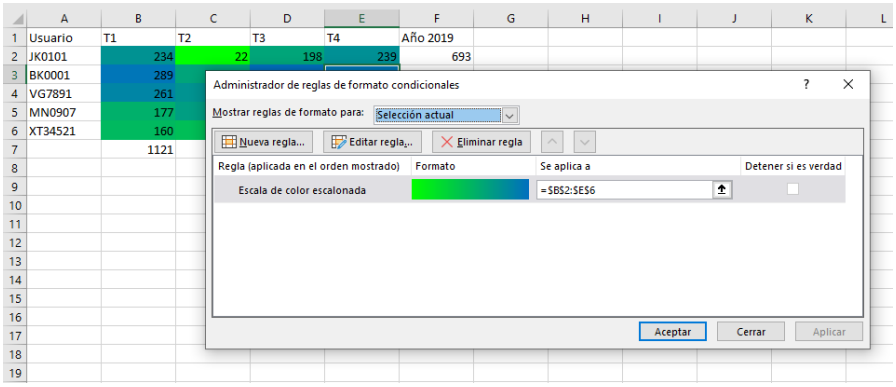


Imagen 2.1 Para activar el Administrador de reglas de formato condicional hacemos clic en la ficha Inicio > Formato condicional > Administrar reglas.

Para crear una nueva regla de formato condicional, hacemos clic en el botón **Nueva regla** en el **Administrador de reglas de formato condicional**. Veremos la lista de reglas predefinidas entre las que podemos seleccionar.

VBA utiliza el método `Add` de la colección `FormatConditions` para crear una nueva regla. Por ejemplo, para dar formato a las celdas que contienen "T" en la cadena de texto, introduciremos el siguiente procedimiento en un módulo estándar y a continuación, lo ejecutaremos.

```
Sub FormatoTextoT()
    With ActiveSheet.UsedRange
        .FormatConditions.Delete
        .FormatConditions.Add Type:=xlTextString, String:="T", _
        TextOperator:=xlContains
        .FormatConditions(1).Interior.Color = RGB(123, 130, 0)
    End With
End Sub
```

Observamos que antes de crear y aplicar un nuevo formato condicional a un rango de celdas, es una buena idea eliminar las condiciones existentes usando el método `Delete`. El método `Add` que se utiliza para añadir una nueva condición requiere al menos el argumento `Type`, que especifica si el formato condicional se base en el valor de la celda o en una expresión. Usaremos las constantes enumeradas en la siguiente tabla para establecer el tipo de condición. Por ejemplo, para dar formato a las celdas que contienen fechas usamos la constante `xlTimePeriod` en el argumento `Type` y especificamos el `DateOperator` utilizando alguna de las siguientes constantes: `xlToday`, `xlYestarday`, `xlTomorrow`, `xlLastWeek`, `xlThisweek`, `xlNextWeeb`, `xlLast7Days`, `xlLastMonth`, `xlThisMonth` o `xlNextMonth`.

```
Selection.FormatConditions.Add Type:=xlTimePeriod, _
DateOperator:=xlLast7Days
```

Constante	Descripción

`xlAboveAverageCondition`	Superior e inferior al promedio:
	`With Selection`
	`.FormatConditions.Delete`
	`.FormatConditions.AddAboveAverage`
	`.FormatConditions(1).AboveBelow = xlAboveAverage`
	`.FormatConditions(1).Font.Bold = True`
	`End With`

`xlBlanksCondition`	Da formato a las celdas en blanco:

Constante	Descripción

	`With Selection`
	`.FormatConditions.Add _`
	`Type:=xlBlanksCondition`
	`End With`

`xlCellValue`	Da formato según el valor de la celda:
	`With Selection`
	`.FormatConditions.Add Type:=xlCellValue, _`
	`Operator:=xlLess, Formula1:="=2000"`
	`.FormatConditions(1).NumberFormat = "#, ##0"`
	`End With`

`xlColorScale`	Da formato con escala de color:
	`If Selection.FormatConditions(1).Type = 3 Then`
	`MsgBox "Formateado con " & _`
	`"Escalas de color."`
	`End If`

`xlDataBar`	Da formato con barras de datos:
	`If Selection.FormatConditions(1).Type = 4 Then`
	`MsgBox "Formateado con " & _`
	`"Barras de datos."`
	`End If`

`xlErrorsCondition`	Da formato a las celdas que contienen errores:
	`Selection.FormatConditions.Add _`
	`Type:=xlErrorsCondition`

`xlExpression`	Utiliza una fórmula personalizada que identifica las celdas a las que se aplica el formato condicional. Por ejemplo, el siguiente procedimiento cambia el color de fondo de las filas alternativas en el rango:
	`Sub FilasAlternas()`
	`With ActiveSheet.UsedRange`

Constante	Descripción
	.FormatConditions.Add Type:=xlExpression, _ FormulaLocal:="=RESIDUO(FILA();2)=0" .FormatConditions(1).Interior.ColorIndex = 6 End With End Sub

xlIconSet	Da formato con conjuntos de iconos.
	If Selection.FormatConditions(1).Type = 6 Then
	MsgBox "Formateado con " & _
	"Conjuntos de iconos."
	End If

xlNoBlanksCondition	Da formato a celdas que no están vacías: Sub CeldasNoVacias() Range("A1:B12").Select Selection.FormatConditions.Add _ Type:=xlNoBlanksCondition With Selection.FormatConditions(1).Interior .ThemeColor = xlThemeColorAccent4 .TintAndShade = 0.399945066682943 End With End Sub
xlNoErrorsCondition	Da formato a celdas que no contienen errores. Sub NoErrores() Range("F1:F7").Select Selection.FormatConditions.Add _ Type:=xlNoErrorsCondition With Selection.FormatConditions(1).Interior .ThemeColor = xlThemeColorAccent4

Constante	Descripción
	.TintAndShade = 0.399945066682943 End With End Sub
xlTextString	Da formato a celdas con texto. With ActiveSheet.UsedRange .FormatConditions.Add Type:=xlTextString, _ String:="es", TextOperator:=xlContains .FormatConditions(1).Font.Bold = True End With
xlTimePeriod	Da formato a celdas con fechas. With ActiveSheet.UsedRange .FormatConditions.Add Type:=xlTimePeriod, _ DateOperator:=xlLastMonth .FormatConditions(1).Interior.ColorIndex = 6 End With
xlTop10	Da formato a los 10 más altos. With Selection .FormatConditions.AddTop10 .FormatConditions(1).TopBottom = xlTop10Top .FormatConditions(1).Value = 5 .FormatConditions(1).Percent = False .FormatConditions(1).Interior.Color = _ RGB(255, 0, 0) End With
xlUniqueValue	Da formato a valores únicos. With Selection .FormatConditions.AddUniqueValues .FormatConditions(1).DupeUnique = xlUnique

Constante	Descripción
	`End With`

2.2 Reglas de precedencia con formatos condicionales

Es posible aplicar múltiples formatos condicionales a una celda. Por ejemplo, podemos aplicar un formato para poner la celda en negrita y luego otro para dibujar un borde rojo. Como estos dos formatos no entran en conflicto entre sí, se pueden aplicar ambos a la misma celda. Sin embargo, si creamos otro formato que le diga a Excel que aplique un borde azul a la celda, esta regla no se aplicará porque entra en conflicto con la regla anterior (la del borde rojo). Para controlar las condiciones aplicadas a un rango de celdas, Excel utiliza reglas de precedencia.

Cuando las reglas entran en conflicto entre sí, Excel aplica la regla de mayor precedencia. Las reglas se evalúan en orden de precedencia (según se muestran en el **Administrador de reglas de formato condicional**). En VBA esto se controla mediante la propiedad `Priority` del objeto `FormatConditions`. Por ejemplo, para asignar la segunda prioridad a la primera regla usamos la siguiente declaración:

```
Range("B2:B17").FormatConditions(1).Priority = 2
```

También podemos hacer que la regla tenga la prioridad más baja:

```
Range("B2:B17").FormatConditions(1).SetLastPriority
```

2.3 Eliminar las reglas de formato condicional

Eliminar todas las reglas de formato condicional es sencillo. Solo tenemos que aplicar el método `Delete`:

```
Range("B2:B17").FormatConditions.Delete
```

En caso de que queramos eliminar una de ellas:

```
Range("B2:B17").FormatConditions(2).Delete
```

2.4 Las barras de datos

La herramienta de visualización de datos conocida como **Barras de datos** permite a los usuarios ver fácilmente cómo se relacionan entre sí determinados valores. Las barras de datos pueden añadirse mediante un formato condicional utilizando el cuadro de diálogo **Nueva regla de formato** (desde la ficha **Inicio > Formato condicional > Barras de datos > Más reglas**) o mediante un procedimiento VBA.

Excel dibuja las barras de datos proporcionalmente según sus valores. Gracias a esta característica, las barras de datos pueden ser usadas para comparar valores.

Podemos dar fácilmente formato a la apariencia de la barra con opciones de formato adicionales como rellenos y bordes sólidos. Gracias a esta característica, podemos ver qué celda tiene el valor más alto. Sin embargo, aplicar un relleno sólido a un dato puede hacer que algunas partes del texto sean más difíciles de leer, especialmente cuando se utilizan colores oscuros.

En VBA se puede crear una regla de barras de datos utilizando el método **AddDatabar** o **Add** de la colección **FormatConditions**, como se muestra a continuación:

```
Sub BarrasDatos()
    With Range("B2:E6").FormatConditions
        .AddDatabar
        .Add Type:=xlDatabar, _
        Operator:=xlGreaterEqual, Formula1:="200"
    End With
End Sub
```

Este procedimiento colocará una barra azul en las celdas de la hoja, como se ilustra en la Imagen 2.2. Observemos que la longitud de la barra de datos corresponde al valor de la celda. Si cambiamos o calculamos los datos de la hoja, la longitud de la barra de datos se ajusta a su valor.

En lugar de utilizar los valores más altos y más bajos para especificar las barras más largas o más cortas, podemos especificar que la barra se base en porcentajes, fórmulas o percentiles. Por ejemplo, podemos utilizar las siguientes instrucciones para cambiar el color, el tipo y los parámetros de umbral de la barra de datos:

```
Set mBar = Selection.FormatConditions.AddDatabar
mBar.MinPoint.Modify _
NewType:=xlConditionValuePercentile, NewValue:=20
mBar.MaxPoint.Modify _
NewType:=xlConditionValuePercentile, NewValue:=80
mBar.BarColor.ColorIndex = 7
```

	A	B	C	D	E	F	G
1	Usuario	T1	T2	T3	T4	Año 2019	
2	JK0101	234	22	198	239	693	
3	BK0001	289	199	300	287	1075	
4	VG7891	261	233	288	178	960	
5	MN0907	177	211	150	145	683	
6	XR34521	160	149	201	230	740	
7		1121	814	1137	1079	4151	
8							
9							
10							

Imagen 2.2 Una hoja a la que se han aplicado barras de datos mediante un procedimiento.

En el procedimiento anterior las propiedades **MinPoint** y **MaxPoint** del objeto **DataBar** se utilizan para establecer los valores de las barras más cortas y más largas del rango de datos, y la propiedad **BarColor** se utiliza para modificar el color de las barras.

2.5 Las escalas de color

También podemos crear otros efectos visuales en una hoja de trabajo seleccionando un rango de celdas y aplicando una escala de colores. La escala de color utiliza el fondo de las celdas para ayudarnos a comprender la variación de los datos. Cuando aplicamos un formato condicional de escala de color a través de la interfaz de usuario o desde un procedimiento, Excel usa los valores más bajos, más altos y los puntos intermedios del rango para determinar el degradado de color. Podemos aplicar una escala de dos o tres colores, como se muestra en la Imagen 2.3.

Para crear una regla de escala de color, utilizamos el método **`AddColorScale`** o **`Add`** de la colección **`FormatConditions`**.

	A	B	C	D	E	F
1	Usuario	T1	T2	T3	T4	Año 2019
2	JK0101	234	22	198	239	693
3	BK0001	289	199	300	287	1075
4	VG7891	261	233	288	178	960
5	MN0907	177	211	150	145	683
6	XR34521	160	149	201	230	740
7		1121	814	1137	1079	4151
8						
9						
10						

Imagen 2.3 Una hoja con formato de escala de color.

```
set cScale = Selection.FormatConditions.AddColorScale _
(ColorScaleType:=2)
```

La declaración anterior crea un objeto **`ColorScale`** de dos colores en las celdas de la hoja de trabajo seleccionadas. Si deseamos cambiar el color verde por el azul, podemos usar las siguientes declaraciones:

```
cScale.ColorScaleCriteria(1).FormatColor.Color = RGB(0, 255, 0)
cScale.ColorScaleCriteria(2).FormatColor.Color = RGB(0, 0, 255)
```

En caso de utilizar colores más oscuros, sería conveniente cambiar el color de la fuente a blanco:

```
Selection.Font.ColorIndex = 2
```

Al igual que con las barras de color, es posible cambiar los valores máximos y mínimos a porcentajes, fórmulas o percentiles.

Para crear formatos condicionales impactantes, podemos aplicar formatos de barras de color y escalas de color al mismo rango de datos.

2.6 Los conjuntos de iconos

Los conjuntos de iconos son otra forma de visualización de datos. Colocándolos en las celdas podemos hacer los datos más completos y visualmente atractivos. Los conjuntos de iconos permiten a los usuarios ver fácilmente la relación entre los valores de los datos, así como reconocer las tendencias.

Para ver las opciones de iconos disponibles, podemos hacer clic en la ficha **Inicio > Formato condicional > Conjuntos de iconos**.

Cada icono del conjunto representa un rango de valores. Por ejemplo, en el conjunto de tres iconos que se muestra en la Imagen 2.4, Excel usa el símbolo de la marca de verificación (check) en un círculo verde para los valores mayores o iguales al 67%, un signo de exclamación para los valores menores de 67% y mayores o iguales al 33% y un símbolo X para los valores menores del 33%.

Los conjuntos de iconos se han convertido en una herramienta de resaltado muy eficaz, gracias a la capacidad de aplicar los iconos solo a celdas específicas en lugar de a todo el rango de celdas. Podemos ocultar el icono para las celdas que cumplen criterios específicos seleccionando **Sin icono** en el menú desplegable de iconos.

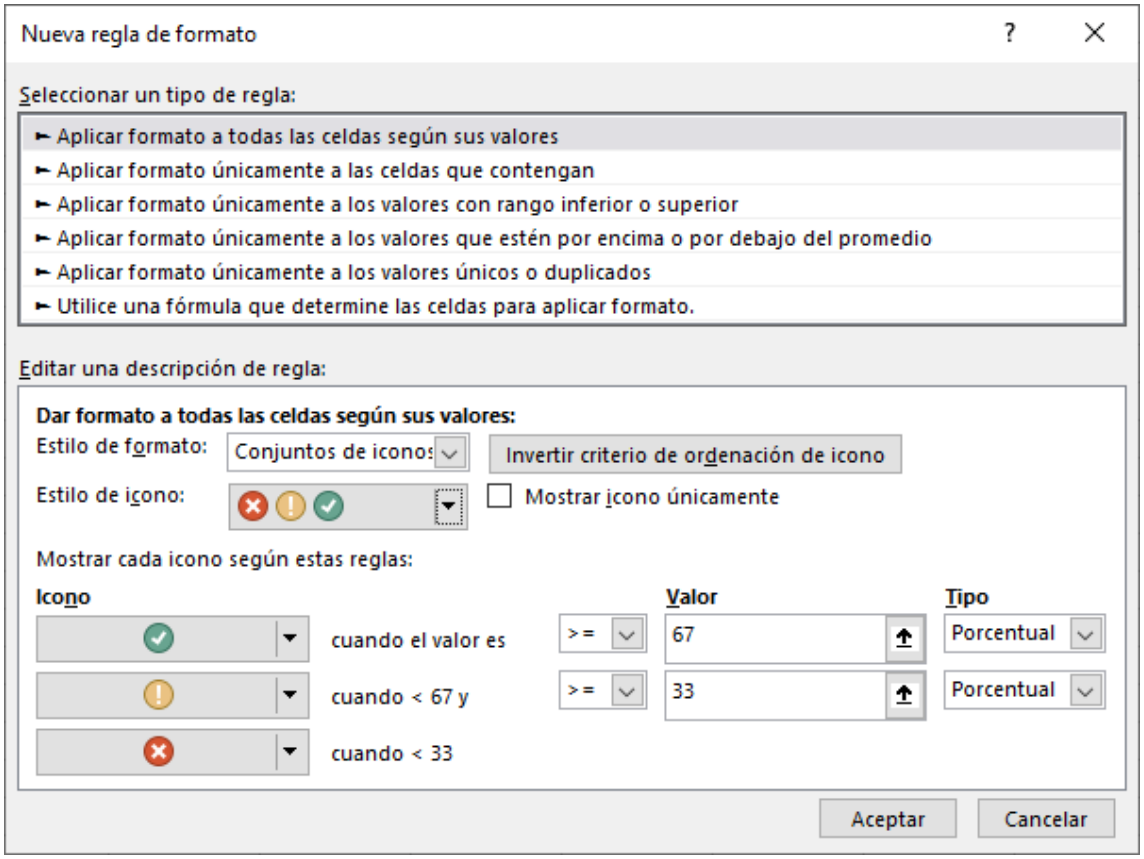


Imagen 2.4 Podemos visualizar los iconos según las reglas establecidas en el cuadro Nueva regla de formato.

El menú desplegable **Estilo de icono** también permite seleccionar conjuntos de cuatro o cinco iconos. Estos conjuntos muestran cada icono según el cuartil o quintil en el que se encuentra el

valor. Podemos modificar los valores máximo y mínimo y su tipo (número, porcentaje, fórmula o percentil) para cada icono editando la regla de formato o con un procedimiento.

En VBA el objeto `IconSet` de la colección `IconSets` representa un único conjunto de iconos. Para crear una regla de formato condicional que utilice conjuntos de iconos, utiliza el objeto `IconSetCondition`. Podemos agregar criterios a una regla de conjunto de iconos con la colección `IconCriteria`. El siguiente procedimiento aplica una regla a un rango de celdas:

1. Abre el archivo Capítulo 18 – Formatos.xlsm y selecciona la Hoja5.
2. Activa el editor de VBA e inserta un módulo nuevo.
3. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub ConjuntoIconos()
    Dim iSC As IconSetCondition

    Columns("C:C").Select
    With Selection
        .SpecialCells(xlCellTypeConstants, 23).Select
        .FormatConditions.Delete
        .NumberFormat = "$#,##0.00"
        Set iSC = Selection.FormatConditions.AddIconSetCondition
        iSC.IconSet = ActiveWorkbook.IconSets(xl3Symbols)
    End With
End Sub
```

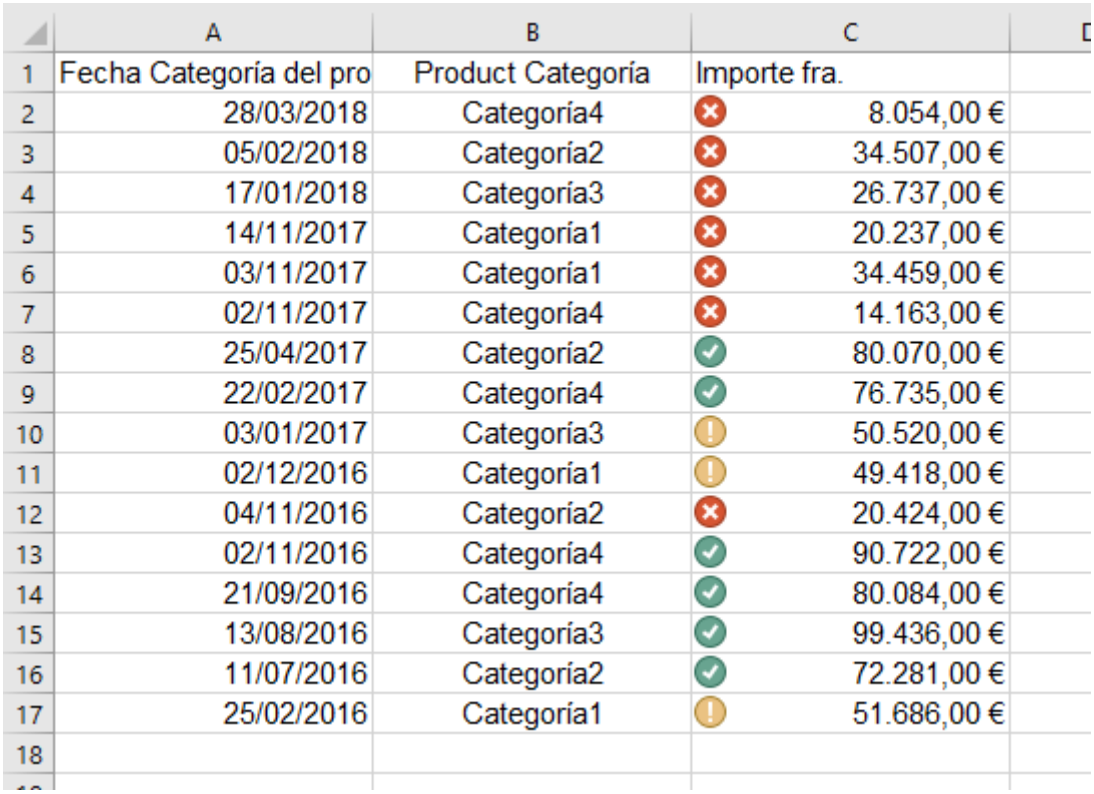
Este procedimiento aplica el formato de moneda a los valores de las celdas de la columna C y borra los formatos condicionales que pudieran haberse aplicado previamente. A continuación, se utiliza el método `AddIconSetCondition` para crear un formato condicional de conjunto de iconos para el rango de celdas seleccionado.

Sitúa el cursor del ratón en cualquier lugar dentro del procedimiento y presiona F8 para ejecutarlo paso por paso.

Como se mencionó anteriormente, puedes modificar las condiciones del conjunto de iconos mediante el cuadro de diálogo o con VBA. Imagina que en vez de utilizar los límites de >=67%, >=33% y <33%, quieres utilizar los criterios >=80.000, >=50.000 y <50.000.

Echa un vistazo al procedimiento modificado que cambia la regla de formato:

```
Sub ConjuntoIconos2()
    Dim iSC As IconSetCondition

    Columns("C:C").Select
```

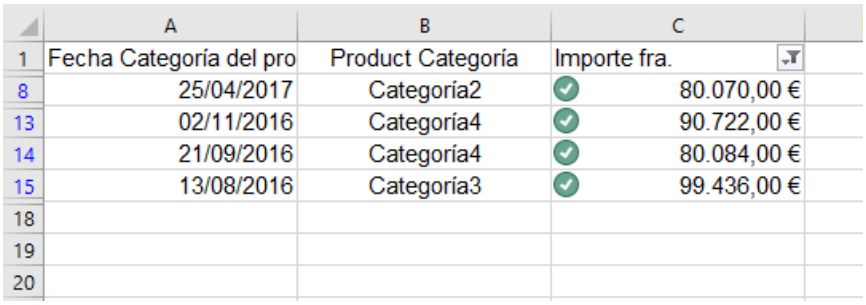


	A	B	C
1	Fecha Categoría del pro	Product Categoría	Importe fra.
2	28/03/2018	Categoría4	8.054,00 €
3	05/02/2018	Categoría2	34.507,00 €
4	17/01/2018	Categoría3	26.737,00 €
5	14/11/2017	Categoría1	20.237,00 €
6	03/11/2017	Categoría1	34.459,00 €
7	02/11/2017	Categoría4	14.163,00 €
8	25/04/2017	Categoría2	80.070,00 €
9	22/02/2017	Categoría4	76.735,00 €
10	03/01/2017	Categoría3	50.520,00 €
11	02/12/2016	Categoría1	49.418,00 €
12	04/11/2016	Categoría2	20.424,00 €
13	02/11/2016	Categoría4	90.722,00 €
14	21/09/2016	Categoría4	80.084,00 €
15	13/08/2016	Categoría3	99.436,00 €
16	11/07/2016	Categoría2	72.281,00 €
17	25/02/2016	Categoría1	51.686,00 €
18			

Imagen 2.5 Una columna con conjuntos de iconos.

```
        Selection.SpecialCells(xlCellTypeConstants, 23).Select
        With Selection
            .FormatConditions.Delete
            .AutoFilter
            .NumberFormat = "#,##0.00 €"
            Set iSC = Selection.FormatConditions.AddIconSetCondition
            iSC.IconSet = ActiveWorkbook.IconSets(xl3Symbols)
                With iSC.IconCriteria(2)
                    .Type = xlConditionValueNumber
                    .Value = 50000
                    .Operator = xlGreaterEqual
                End With
                With iSC.IconCriteria(3)
                    .Type = xlConditionValueNumber
                    .Value = 80000
                    .Operator = xlGreaterEqual
                End With
            .AutoFilter Field:=1, Criteria1:=iSC.IconSet.Item(3), _
            Operator:=xlFilterIcon
        End With
    End Sub
```

Ten en cuenta que al cambiar los criterios del formato condicional no es necesario especificar el tipo, el valor y el operador de `IconCriteria(1)`. Esta propiedad es de solo lectura. Excel determina por sí mismo los valores límite de `IconCriteria(1)`, y si intentas establecerlo en el código, obtendrás un error en tiempo de ejecución. Los comandos **Ordenar** y **Filtrar** te permiten ordenar o filtrar datos basados en el icono de la celda. El procedimiento anterior muestra cómo es posible aplicar el filtro desde VBA utilizando un icono. Los resultados del procedimiento se muestran en la Imagen 2.6.



	A	B	C
1	Fecha Categoría del pro	Product Categoría	Importe fra.
8	25/04/2017	Categoría2	80.070,00 €
13	02/11/2016	Categoría4	90.722,00 €
14	21/09/2016	Categoría4	80.084,00 €
15	13/08/2016	Categoría3	99.436,00 €
18			
19			
20			

Imagen 2.6 Después de ejecutar el procedimiento, se aplica un filtro a la columna Importe fra. para mostrar solo las celdas con valores >=80.000.

Imagina que solo quieres resaltar las celdas con valores inferiores a 50.000. El siguiente procedimiento crea una regla de formato que produce el resultado que se muestra en la Imagen 2.7.

```
Sub ConjuntosIconosOcultos()
    Dim iSC As IconSetCondition

    Columns("C:C").Select
    Selection.SpecialCells(xlCellTypeConstants, 23).Select
    With Selection
        .FormatConditions.Delete
        .NumberFormat = "#,##0.00 €"
        Set iSC = Selection.FormatConditions. _
        AddIconSetCondition
        iSC.IconSet = ActiveWorkbook.IconSets(xl3Symbols)
        .FormatConditions(1).IconCriteria(1). _
        Icon = xlIconRedCrossSymbol
        With iSC.IconCriteria(2)
            .Type = xlConditionValueNumber
            .Value = 50000
            .Operator = xlGreaterEqual
            .Icon = xlIconNoCellIcon
        End With
        With iSC.IconCriteria(3)
            .Type = xlConditionValueNumber
```

```
                .Value = 80000
                .Operator = xlGreaterEqual
                .Icon = xlIconNoCellIcon
            End With
        End With
    End Sub
```

2.7 Dar formatos con temas

Si necesitamos cambiar el aspecto de todo el libro, merecerá la pena dedicar un tiempo a familiarizarnos con los temas de los documentos. Un tema de documento consiste en un conjunto predefinido de fuentes, colores y efectos que podemos aplicar al libro y también compartir con otros documentos de Office.

Es importante señalar que un tema se aplica a todo el libro, no solo a la hoja activa. Hay numerosos temas de documentos disponibles en la interfaz de usuario (podemos verlos en la ficha **Diseño de página > Temas**) y también podemos crear temas personalizados mezclando diferentes elementos de otros temas utilizando los tres controles desplegables que se encuentran en el mismo lugar (Colores, Fuentes y Efectos). También se pueden descargar temas desde Office Online.

	A	B	C
1	Fecha Categoría del pro	Product Categoría	Importe fra.
2	28/03/2018	Categoría4	8.054,00 €
3	05/02/2018	Categoría2	34.507,00 €
4	17/01/2018	Categoría3	26.737,00 €
5	14/11/2017	Categoría1	20.237,00 €
6	03/11/2017	Categoría1	34.459,00 €
7	02/11/2017	Categoría4	14.163,00 €
8	25/04/2017	Categoría2	80.070,00 €
9	22/02/2017	Categoría4	76.735,00 €
10	03/01/2017	Categoría3	50.520,00 €
11	02/12/2016	Categoría1	49.418,00 €
12	04/11/2016	Categoría2	20.424,00 €
13	02/11/2016	Categoría4	90.722,00 €
14	21/09/2016	Categoría4	80.084,00 €
15	13/08/2016	Categoría3	99.436,00 €
16	11/07/2016	Categoría2	72.281,00 €
17	25/02/2016	Categoría1	51.686,00 €
18			

Imagen 2.7 En este escenario, se utiliza el conjunto de iconos para llamar la atención sobre las celdas con cantidades de factura inferiores a 50.000. Fijémonos que cuando no se aplican otros iconos es más fácil resaltar las celdas problemáticas.

Cuando cambiamos un tema, los selectores de fuente y color de la ficha **Inicio** y otras galerías como la de **Estilos de celdas** o **Estilos de tabla** se actualizan automáticamente para reflejar el nuevo tema. Por lo tanto, si buscamos aplicar un fondo de celda con un color en particular, y ese color no aparece en el selector de colores, aplicaremos un tema predefinido que contenga ese color, o crearemos nuestro propio color personalizado seleccionando la opción **Más colores** en el menú desplegable **Color de fuente**.

En la ficha **Disposición de página** (o **Diseño de página**, según la versión de Excel), el menú desplegable **Colores** muestra los grupos de colores de cada tema y nos da la opción de crear nuevos colores de tema. El menú desplegable **Fuentes** muestra una lista de fuentes para el tema. Las fuentes del tema contienen una fuente de encabezado y una fuente para el texto. Dichas fuentes se pueden cambiar con la opción **Crear nuevas fuentes del tema** en el menú desplegable. El menú desplegable **Efectos** muestra la línea y el relleno para cada uno de los temas y no nos da la opción de crear un nuevo conjunto de efectos.

Un esquema de color de un tema consiste en 12 colores de base como se ilustra en la Imagen 2.8. Cuando se aplica un color a una celda, el color se selecciona en el menú desplegable **Color de relleno** como se muestra en la Imagen 2.9.

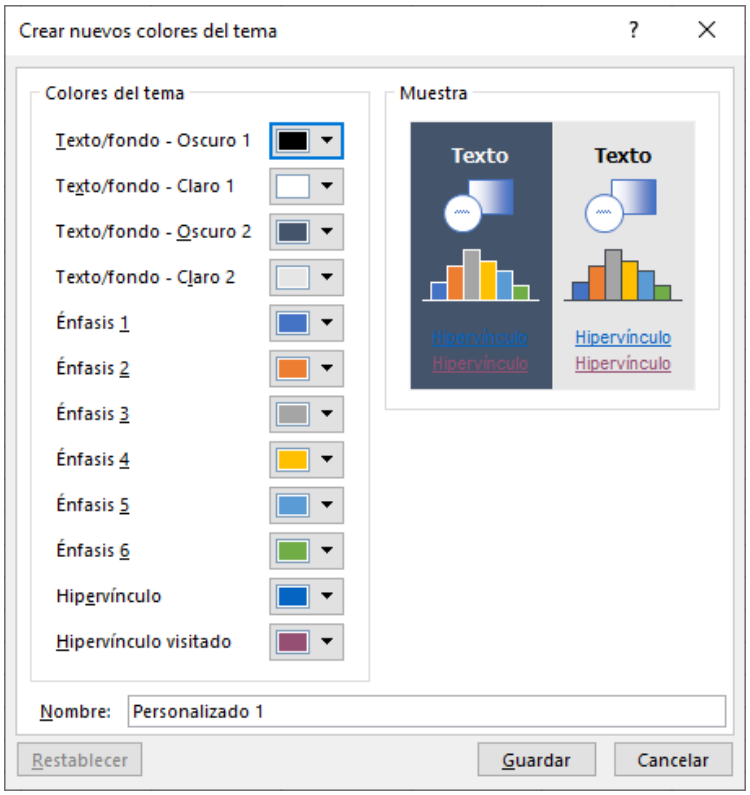


Imagen 2.8 Podemos crear nuestro conjunto de colores personalizado.

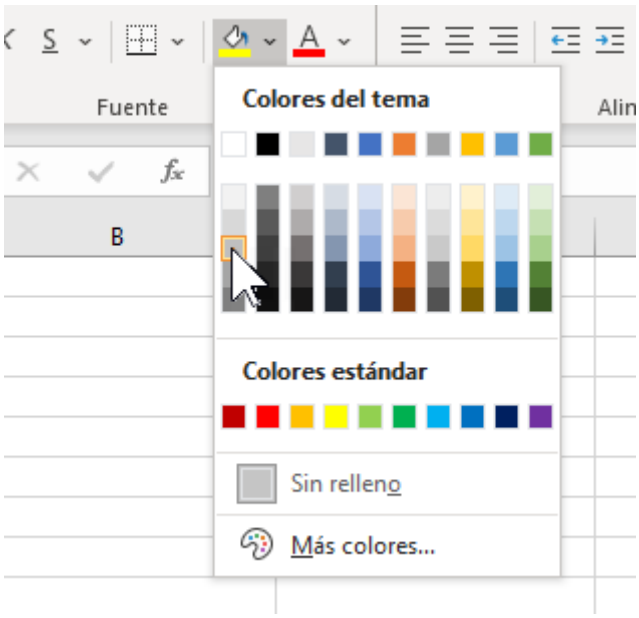


Imagen 2.9 El color de relleno se usa para establecer el color de fondo de la celda.

En la ficha **Inicio**, en el grupo **Fuente** encontramos el botón **Color de relleno**, que facilita la visualización de los colores del tema (ver Imagen 2.9). Al hacer clic en la flecha desplegable situada junto al botón **Color de relleno**, veremos una paleta de colores. La fila superior de la paleta muestra los 10 colores base en el tema de color actual. Las cinco filas de debajo muestran variaciones del color base. Un color puede ser más claro o más oscuro. El nombre del color se muestra en la información de la herramienta. Si grabamos una macro aplicando el color "Azul Énfasis 1 Claro 40%", obtendremos el siguiente código:

```
Sub Macro3()
'
' Macro3 Macro
'

'
    Range("D3").Select
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .ThemeColor = xlThemeColorAccent1
        .TintAndShade = 0.399975585192419
        .PatternTintAndShade = 0
    End With
End Sub
```

Las propiedades del patrón hacen referencia a los patrones de celda que pueden establecerse a través del cuadro de diálogo **Formato de celdas** (presiona **Ctrl + 1** y selecciona la pestaña **Relleno**). Estas propiedades pueden ser ignoradas si todo lo que se requiere es establecer el color de fondo de la celda. El procedimiento anterior puede modificarse de la siguiente forma:

```
Sub Macro3()
'
' Macro3 Macro
'

'
    Range("D3").Select
    With Selection.Interior
        .ThemeColor = xlThemeColorAccent1
        .TintAndShade = 0.399975585192419
    End With
End Sub
```

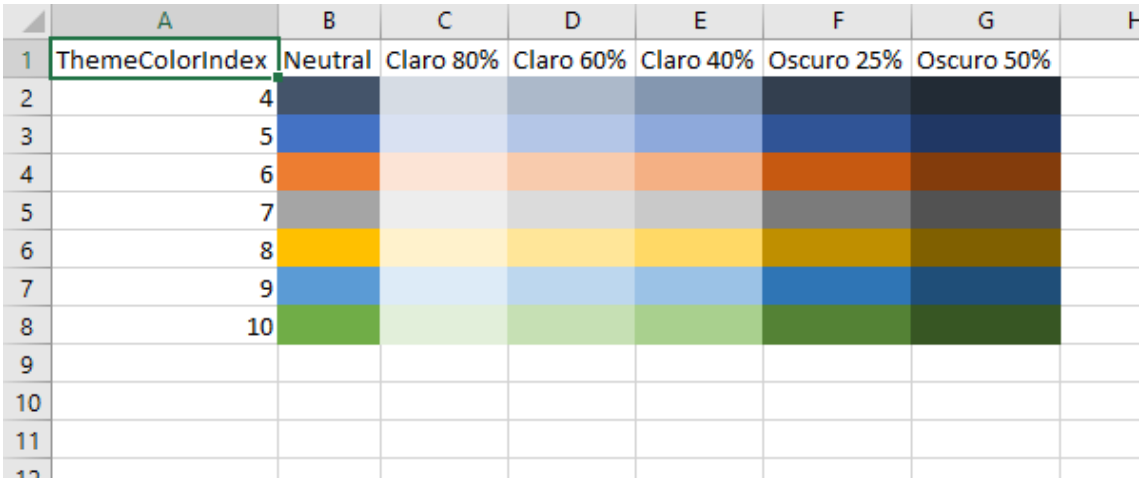
La propiedad `ThemeColor` especifica el color del tema que se usará.

La propiedad `TintAndShade` se utiliza para modificar el color seleccionado. Se utiliza un valor de -1 a 1. Si deseamos un color puro, establecemos la propiedad en 0. El valor -1 será el color negro y el 1 el blanco. Los valores negativos producirán colores más oscuros y los valores positivos los producirán más claros. El valor de 0.399975585192419 significa que se trata de un color un 40% más claro que el color base. Si cambiamos de signo (-0.399975585192419) obtendremos un color un 40% más oscuro.

El siguiente procedimiento hace un bucle sobre los colores de los temas 4 a 10 y escribe el índice del color y sus variaciones en un rango de celdas, como se muestra en la Imagen 2.10.

```
Sub Colores4a10()
    Dim tintshade As Variant
    Dim encabezados As Variant
    Dim cell As Range
    Dim themeC As Integer
    Dim r As Integer
    Dim c As Integer
    Dim i As Integer

    encabezados = Array("ThemeColorIndex", "Neutral", _
    "Claro 80%", "Claro 60%", "Claro 40%", _
    "Oscuro 25%", "Oscuro 50%")
    tintshade = Array(0, 0.8, 0.6, 0.4, -0.25, -0.5)
    i = 0
    For Each cell In Range("A1:G1")
        cell.Formula = encabezados(i)
        i = i + 1
    Next
```

```
        For r = 2 To 8
            themeC = r + 2
            For c = 1 To 7
                If c = 1 Then
                    Cells(r, c).Formula = themeC
                Else
                    With Cells(r, c)
                        With .Interior
                            .ThemeColor = themeC
                            .TintAndShade = tintshade(c - 2)
                        End With
                    End With
                End If
            Next c
        Next r
        ActiveSheet.Columns("A:G").AutoFit
    End Sub
```



	A	B	C	D	E	F	G
1	ThemeColorIndex	Neutral	Claro 80%	Claro 60%	Claro 40%	Oscuro 25%	Oscuro 50%
2	4						
3	5						
4	6						
5	7						
6	8						
7	9						
8	10						
9							
10							
11							

Imagen 2.10 La paleta ha sido generada por un procedimiento. Si se aplica un tema diferente, los colores serán reemplazados por los del nuevo tema.

El siguiente procedimiento aplica los colores del tema actual a un rango de celdas en la hoja de trabajo activa:

```
    Sub TemasColor()
        Dim tColorScheme As ThemeColorScheme
        Dim colorArray(10) As Variant
        Dim i As Long
        Dim r As Long

        Set tColorScheme = ActiveWorkbook.Theme.ThemeColorScheme
        For i = 1 To 10
```

```
            colorArray(i) = tColorScheme.Colors(i).RGB
            ActiveSheet.Cells(i, 1).Value = colorArray(i)
        Next i
        i = 0
        For r = 1 To 10
            ActiveSheet.Cells(r, 2).Interior.Color = _
            colorArray(i + 1)
            i = i + 1
        Next r
    End Sub
```

En el procedimiento anterior, el objeto **ThemeColorScheme** representa el color de esquema de un tema de Office. En el primer bucle **For … Next**, los colores y la propiedad RGB se usan para devolver un color específico. El valor del color se almacena en la variable **colorArray** y se introduce en la fila especificada de la primera columna de la hoja. El segundo bucle **For … Next** aplica el color de fondo a las celdas basado en los valores de color almacenados en **colorArray**.

El siguiente procedimiento hace más acciones relacionadas con colores, pero esta vez utilizando la propiedad **Interior.ThemeColor**:

```
    Sub AplicaColoresTema()
        Dim i As Integer

        For i = 1 To 10
            ActiveSheet.Cells(i, 3).Interior.ThemeColor = i
            ActiveSheet.Cells(i, 4).Value = i
        Next i
    End Sub
```

En este procedimiento, se usa la propiedad **Interior.ThemeColor** para establecer el color de fondo de las celdas de la tercera columna usando los colores del esquema de color actual. El valor del color se escribe en la celda correspondiente de la siguiente columna. La hoja de trabajo resultante se muestra en la Imagen 2.11.

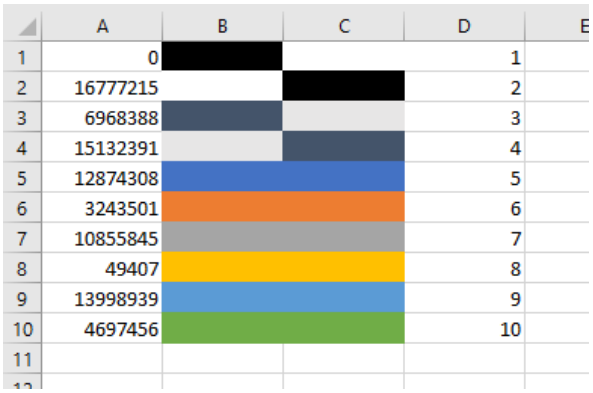


Imagen 2.11 Según el color del tema, el color de fondo de las celdas C1:C10 cambiará.

Cada libro que se crea lo hace con el tema predeterminado “Office”. La información del tema se almacena en un archivo diferente con la extensión .thmx. Cuando se cambia el tema en el libro, el archivo de tema se actualiza automáticamente con la nueva configuración. Cuando creamos un tema personalizado seleccionando nuevas fuentes, colores o efectos, debemos guardarlo en un archivo para que se pueda usar en cualquier documento de otra aplicación de Office o se pueda compartir con otros usuarios. Encontraremos el archivo del tema personalizado en la siguiente ubicación:

\Users\<nombre usuario>\AppData\Roaming\Microsoft\Templates\Document Themes

De forma predeterminada la carpeta **AppData** está oculta. Para acceder a ella debemos modificar las opciones de visualización de archivos de Windows.

Para crear un tema de prueba que se llame MiTema.thmx, debemos seleccionar **Disposición de página > Temas > Guardar tema actual**, cambiamos el nombre del tema a **MiTema** y presionamos en Guardar. Ahora que hemos creado un tema personalizado, podemos aplicarlo desde VBA al libro de trabajo mediante el método `ApplyTheme` del objeto `Workbook`.

Probémoslo ahora introduciendo la siguiente declaración en la ventana **Inmediato** (revisa la ruta para que coincida con la ubicación en tu equipo):

```
ActiveWorkbook.ApplyTheme
"C:\Users\Sergio\AppData\Roaming\Microsoft\Templates\Document
Themes\MiTema.thmx"
```

(Asegúrate de que lo introduces en una sola línea).

2.8 Formatos con formas (Shapes)

Podemos hacer nuestra hojas de cálculo más interesantes añadiendo varios tipos de formas, como el cilindro de la Imagen 2.12. Al dar formato a las formas, podemos usar los colores del tema del documento como se muestra en el siguiente procedimiento:

```
Sub FormaCilindro()
    Dim oShape As Shape

    Set oShape = ActiveSheet.Shapes.AddShape( _
    msoShapeCan, 54, 0, 54, 110)
    With oShape
        .Fill.ForeColor.ObjectThemeColor = msoThemeColorAccent4
        .Fill.Transparency = 0.5
        .Line.Visible = msoFalse
    End With
    Set oShape = Nothing
End Sub
```

En el procedimiento anterior declaramos una variable de objeto tipo `Shape` y luego utilizamos el método `AddShape` para añadir un nuevo objeto `Shape`. Este método tiene cinco argumentos obligatorios. El primero especifica el tipo de objeto `Shape` que deseamos crear. Se

trata de una de las constantes de **msoAutoShapeType** (ver ayuda de Office online). Excel ofrece un gran número de formas. Los dos argumentos siguientes le indican a Excel a qué distancia debe colocarse el objeto desde la esquina superior izquierda de la cuadrícula.

Los dos últimos argumentos especifican el ancho y el alto de la forma (en puntos). Para especificar el color del objeto **Shape**, establecemos la propiedad **ObjectThemeColor** del objeto **ColorFormat** del tema requerido. Para devolver el objeto **ColorFormat**, debemos usar la propiedad **ForeColor** del objeto **FillFormato**.

```
oShape.Fill.ForeColor.ObjectThemeColor = msoThemeColorAccent4
```

A continuación, establecemos el grado de transparencia para asegurarnos de que la forma no obstruya los datos. La última línea elimina el borde de la forma.

El procedimiento anterior coloca un objeto **Shape** sobre los datos de la columna B.

	A	B	C	D	E	F	G
1	Usuario	T1	T2	T3	T4	Año 2019	
2	JK0101	234	22	198	239	693	
3	BK0001	289	199	300	287	1075	
4	VG7891	261	233	288	178	960	
5	MN0907	177	211	150	145	683	
6	XR34521	160	149	201	230	740	
7		1121	814	1137	1079	4151	
8							
9							
10							

Imagen 2.12 Un objeto Shape colocado en la hoja de trabajo utiliza el esquema de color de la hoja.

Utilicemos el siguiente procedimiento para eliminar todas las formas de la hoja:

```
Sub EliminarFormas()
    Dim oShape As Shape
    Dim strShapeName As String

    With ActiveSheet
        For Each oShape In .Shapes
            strShapeName = oShape.Name
            oShape.Delete
            Debug.Print "El objeto Shape " _
            & strShapeName & " ha sido eliminado."
        Next oShape
    End With
End Sub
```

Atención

Aunque podemos buscar las propiedades y métodos del objeto **`Shape`** en la ayuda en línea, no debemos olvidar la herramienta de programación más útil de Excel: la grabadora de macros. Utilízala para grabar acciones con los objetos **`Shape`** (insertar, dar formato, etc.) para comenzar a escribir rápidamente el código VBA y familiarizarte con la terminología.

2.9 Formatos con Minigráficos

Los Minigráficos (Sparklines en inglés) son pequeños gráficos que se insertan en las celdas para resaltar tendencias importantes de los datos y aumentar la comprensión lectora. Hay tres tipos de minigráficos en Excel: Línea, Columna y Pérdidas y ganancias. Se pueden crear desde el botón correspondiente en el grupo **Minigráficos** de la ficha **Insertar** (ver Imagen 2.13).

Imagen 2.13 El grupo Minigráficos de la ficha Insertar ofrece tres botones para crear pequeños gráficos en las celdas.

Para insertar un minigráfico de forma manual, hacemos clic en la celda en la que deseamos que aparezca y seleccionamos el botón correspondiente al tipo de minigráfico. Excel mostrará el cuadro de diálogo **Crear minigráficos** (ver Imagen 2.14), donde podemos elegir o introducir el rango de datos que vamos a incluir como fuente de datos para el minigráfico y el rango de ubicación donde deseamos colocarlo.

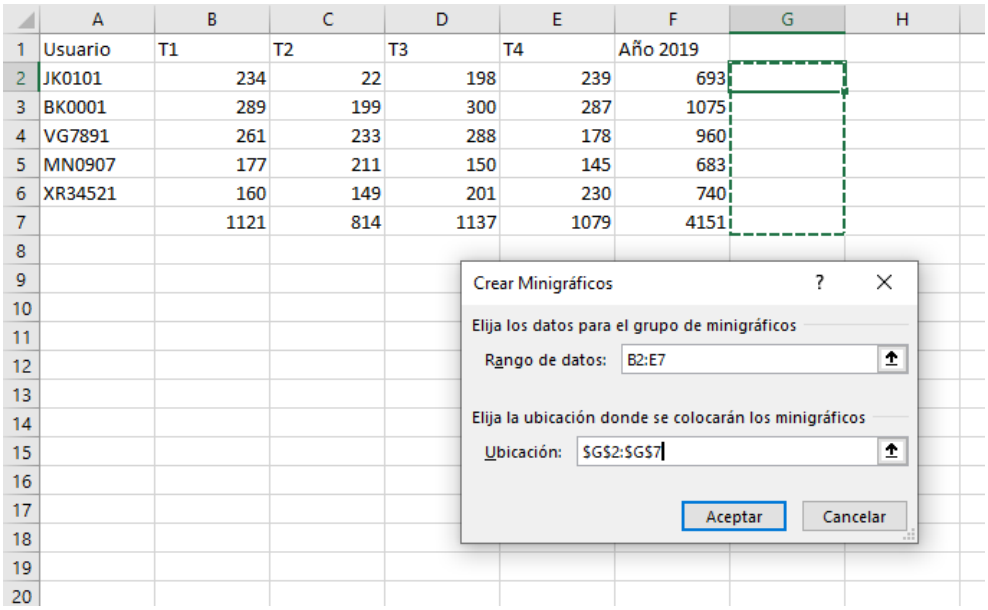
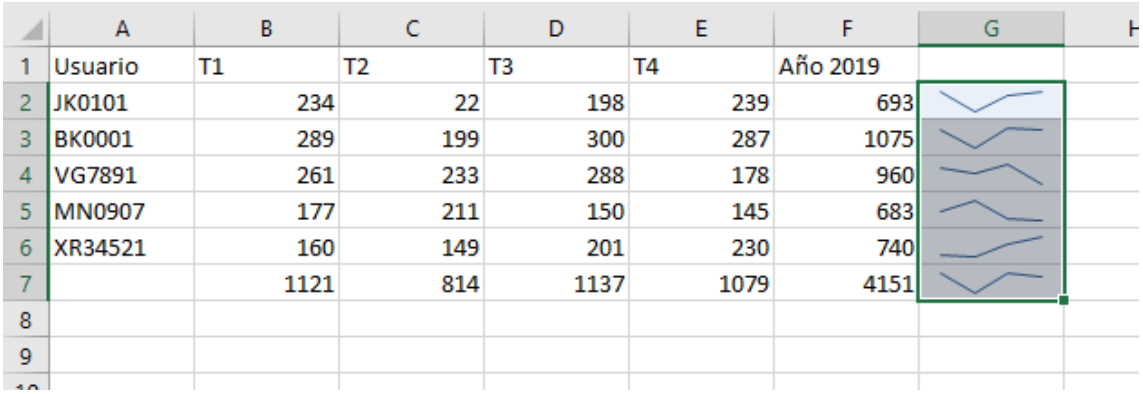


Imagen 2.14 Seleccionando la fuente de los datos y la localización de los minigráficos.

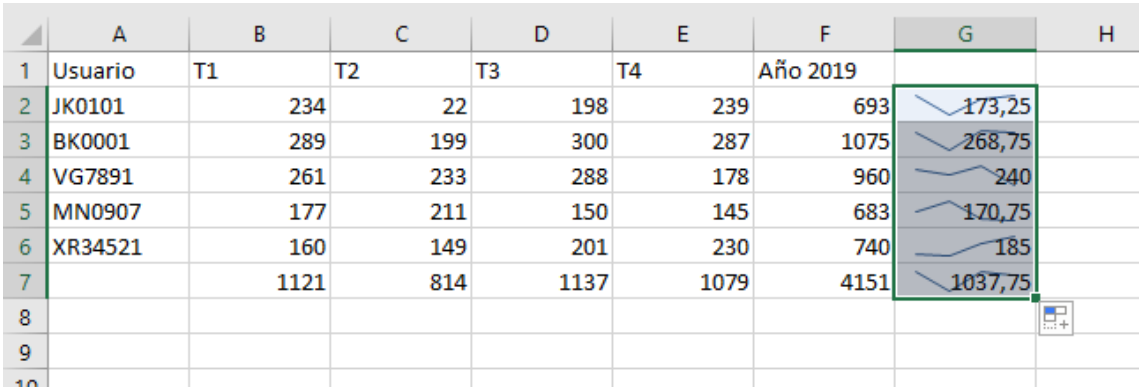
En la Imagen 2.14, nos gustaría comparar el rendimiento de los usuarios en cada trimestre de 2019. Después de seleccionar los datos en el cuadro de diálogo hacemos clic en **Aceptar** para que Excel cree los minigráficos y active la ficha contextual **Minigráficos**, desde la que podemos dar formato a los minigráficos mostrando puntos y marcadores, cambiando los colores de las líneas o marcadores, cambiando los tipos de minigráficos, etc. La Imagen 2.15 muestra el resultado de insertar y aplicar el formato de líneas.



	A	B	C	D	E	F	G
1	Usuario	T1	T2	T3	T4	Año 2019	
2	JK0101	234	22	198	239	693	
3	BK0001	289	199	300	287	1075	
4	VG7891	261	233	288	178	960	
5	MN0907	177	211	150	145	683	
6	XR34521	160	149	201	230	740	
7		1121	814	1137	1079	4151	
8							
9							

Imagen 2.15 Los minigráficos de la columna G son ideales para identificar tendencias.

Los minigráficos son dinámicos. Esto significa que se ajustan automáticamente cuando cambian los datos de las celdas que los componen. Pueden copiarse, cortarse y pegarse como las fórmulas. Si deseamos obtener minigráficos más grandes, simplemente debemos aumentar el ancho de la columna o el alto de la fila. Para eliminar un minigráfico, nos dirigiremos a la ficha contextual y presionaremos el botón **Borrar**. Como los minigráficos forman parte del fondo de la celda, podemos mostrar texto o fórmulas encima, como se muestra en la Imagen 2.16:



	A	B	C	D	E	F	G	H
1	Usuario	T1	T2	T3	T4	Año 2019		
2	JK0101	234	22	198	239	693	173,25	
3	BK0001	289	199	300	287	1075	268,75	
4	VG7891	261	233	288	178	960	240	
5	MN0907	177	211	150	145	683	170,75	
6	XR34521	160	149	201	230	740	185	
7		1121	814	1137	1079	4151	1037,75	
8								
9								

Imagen 2.16 Los minigráficos pueden compartir el espacio de las celdas con valores.

2.10 Minigráficos y VBA

El modelo de objetos de Excel contiene objetos, propiedades, métodos y eventos que permiten a los desarrolladores utilizar VBA para crear y modificar minigráficos. Para tener una idea de donde se encuentran los minigráficos en el modelo de objetos, echemos un vistazo a la Imagen 2.17.

Cada minigráfico está representado por un objeto **Sparkline**, que a su vez es miembro del objeto **SparklineGroup**. La propiedad **SparklineGroup** del objeto **Range** devuelve un objeto **SparklineGroups** que representa a un grupo de minigráficos en un rango específico.

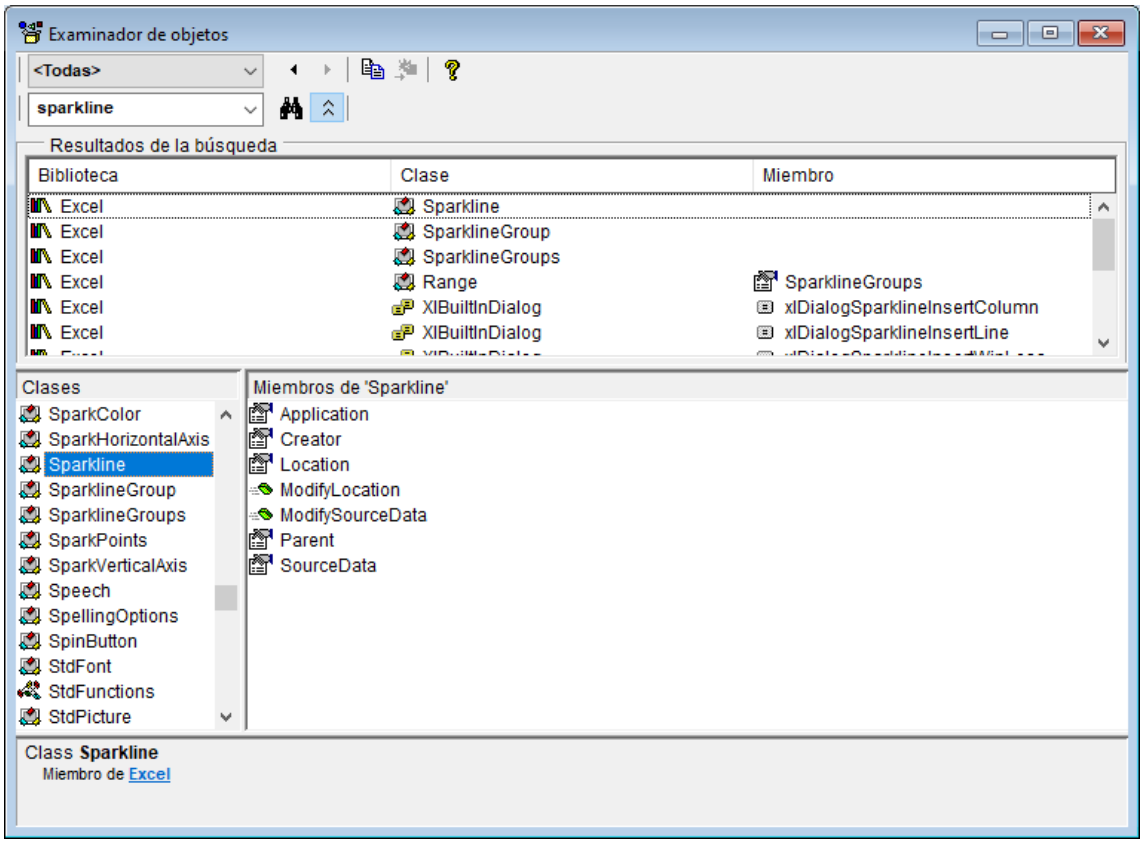


Imagen 2.17 En el Examinador de objetos podemos localizar rápidamente los objetos, métodos y propiedades que se utilizan para programar minigráficos.

El siguiente ejemplo muestra cómo usar VBA para leer los datos de minigráficos en una hoja:

1. En el libro Capítulo 18 – Formularios.xlsm, activa el editor de VBA y crea un módulo nuevo.
2. En la ventana **Código** del módulo introduce el siguiente procedimiento:

```
Sub InfoMinigraficos()
    Dim spGrp As SparklineGroup
    Dim spCount As Long
    Dim i As Long

    spCount = Cells.SparklineGroups.Count
    If spCount <> 0 Then
        For i = 1 To spCount
            Set spGrp = Cells.SparklineGroups(i)
            Debug.Print "Tipo de minigráficos:" & i
            Select Case spGrp.Type
                Case 1
```

```
                    Debug.Print "Tipo: Línea"
                Case 2
                    Debug.Print "Tipo: Columna"
                Case 3
                    Debug.Print "Tipo: Pérdidas y ganancias"
            End Select
            Debug.Print "Localización:" & spGrp.Location.Address
            Debug.Print "Fuente de datos: " & spGrp.SourceData
        Next i
    Else
        MsgBox "No hay minigráficos en la hoja activa."
    End If
End Sub
```

3. Activa la hoja que contiene los minigráficos y ejecuta el procedimiento anterior. En la ventana **Inmediato** se mostrará información similar a esta:

Tipo de minigráficos:1
Tipo: Línea
Localización:$G$2:$G$7
Fuente de datos: B2:E7

El siguiente ejercicio creará un informe con minigráficos de Pérdidas y ganancias. Este tipo de minigráficos se utilizan para comparar diversas cantidades:

1. Introduce los siguientes procedimientos debajo del procedimiento `InfoMinigraficos`:

```
Sub InformeMinigraficos()
    Dim spGrp As SparklineGroup
    Dim sht As Worksheet
    Dim cell As Range
    Dim spLocation As Range

    Workbooks.Add
    Set sht = ActiveSheet
    IntroduceDatos sht, 3, "Mes", "Gastos €", "Ingresos €", _
    "Diferencia"
    IntroduceDatos sht, 4, "Enero", "234000", "250000", "=C4-B4"
    IntroduceDatos sht, 5, "Febrero", "211000", "180000", _
    "=C5-B5"
    IntroduceDatos sht, 6, "Marzo", "304000", "370000", "=C6-B6"
    Range("B4:D6").Style = "Currency"
    Columns("A:D").AutoFit
```

```
    Range("A1").Value = "Balance"
    Set spLocation = sht.Range("B1")
    Set spGrp = spLocation.SparklineGroups _
    .Add(xlSparkColumnStacked100, "D4:D6")
    spGrp.SeriesColor.ThemeColor = 2
    spLocation.SparklineGroups.Item(1) _
    .Axes.Horizontal.Axis.Visible = True
End Sub
Sub IntroduceDatos(sht As Worksheet, rowNum As Integer, _
ParamArray myValues() As Variant)
    Dim j As Integer
    Dim count As Integer

    count = UBound(myValues()) + 1
    j = 1
    For j = j To count
        sht.Range(Cells(rowNum, 1), Cells(rowNum, count)) = _
        myValues()
    Next
End Sub
```

2. Ejecuta el procedimiento `InformeMinigraficos`.
   Tras ejecutarlo debería aparecer el informe de Pérdidas y Ganancias en un libro nuevo como se muestra en la Imagen 2.18. Observa que el minigráfico compara datos de gastos y ventas durante los tres primeros meses del año. La fuente de datos es la columna D.

	A	B	C	D
1	Balance			
2				
3	Mes	Gastos €	Ingresos €	Diferencia
4	Enero	234.000,00 €	250.000,00 €	16.000,00 €
5	Febrero	211.000,00 €	180.000,00 €	- 31.000,00 €
6	Marzo	304.000,00 €	370.000,00 €	66.000,00 €
7				
8				

Imagen 2.18 Un informe creado íntegramente con VBA.

2.11 Formatos con estilos de celda

Mucha gente utiliza la herramienta **Copiar formato** de la ficha **Inicio** para copiar rápidamente el formato a otras celdas. Sin embargo, cuando se crean hojas de trabajo complejas con diferentes tipos de formato, es una buena idea guardar todos los ajustes de formato en un archivo para poder reutilizarlos siempre que los necesitemos. Esto se puede hacer a través de

la función **Estilos**. Los estilos de celdas pueden contener opciones de formato, como Número, Alineación, Fuente, Bordes, Relleno y Protección. Si cambiamos el estilo después de haberlo aplicado a la hoja de trabajo, Excel actualizará automáticamente las celdas que han sido formateadas usando ese estilo.

Los estilos son más fáciles de encontrar y aplicar, gracias a la existencia de galerías como las que se ilustran en la Imagen 2.19. Para aplicar un estilo a una celda, simplemente seleccionamos las celdas que deseamos formatear con el estilo, y hacemos clic en el estilo apropiado en la galería (se muestra al hacer clic en el botón **Estilos de celda**). Los estilos se basan en el tema actual. También podemos aplicarlos a tablas, tablas dinámicas, gráficos y formas. Excel ofrece un gran número de estilos predefinidos que podemos modificar, duplicar o eliminar, y además podemos añadir los nuestros propios: simplemente hacemos clic con el botón derecho del ratón en el estilo de la galería y seleccionamos la opción que deseemos.

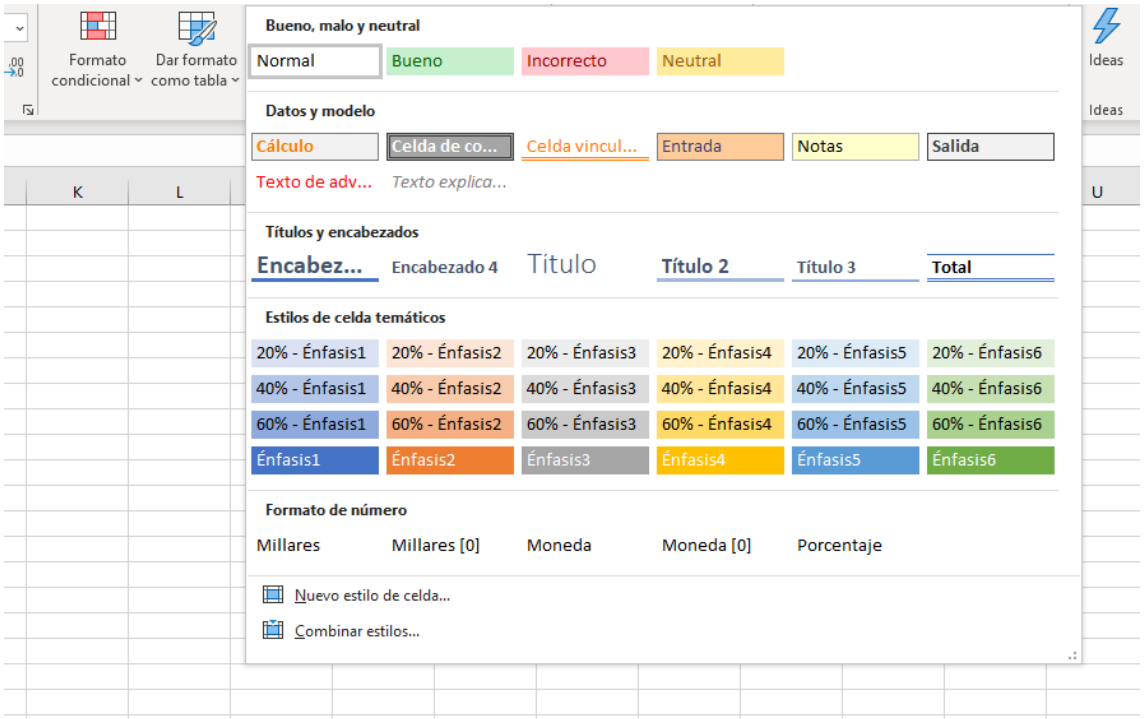


Imagen 2.19 Es posible previsualizar el estilo antes de aplicarlo. Si el estilo predefinido no se ajusta a nuestras necesidades, podemos crearlos de forma personalizada.

Para saber el número de estilos en el libro de trabajo, usaremos la siguiente declaración:

```
MsgBox "Número de estilos=" & ActiveWorkbook.Styles.Count
```

Excel nos dice que hay 47 estilos predefinidos en un libro de Excel 2019. Utilizaremos la colección `Styles` y el objeto `Style` para controlar los estilos del libro. Para obtener una lista de los nombres de estilos hagamos un bucle en la colección `Styles`:

```
Sub NombresEstilos()
    Dim i As Integer

    For i = 1 To ActiveWorkbook.Styles.count
        Debug.Print "Estilo " & i & ":" & _
```

```
        ActiveWorkbook.Styles(i).Name
        Next i
    End Sub
```

El procedimiento anterior imprime los nombres de todos los estilos de celda en la ventana **Inmediato**. Los nombres de los estilos se muestran alfabéticamente.

Para agregar un estilo, utilizamos el método `Add`, como se muestra en el siguiente procedimiento:

```
    Sub NuevoEstilo()
        Dim newStyleName As String
        Dim curStyle As Variant
        Dim i As Integer

        newStyleName = "Formato simple"
        i = 0
        For Each curStyle In ActiveWorkbook.Styles
            i = i + 1
            If curStyle.Name = newStyleName Then
                MsgBox "This style " & "(" & newStyleName & _
                ") ya existe. " & Chr(13) & _
                "Es el estilo " & i & " en la colección Styles."
                Exit Sub
            End If
        Next
        With ActiveWorkbook.Styles.Add(newStyleName)
            .Font.Name = "Arial Narrow"
            .Font.Size = "12"
            .Borders.LineStyle = xlThin
            .NumberFormat = "#,##0 €_);[Red](#,##0 €)"
            .IncludeAlignment = False
        End With
    End Sub
```

El procedimiento anterior añade un estilo a la colección de estilos siempre que el nombre sea único. El procedimiento comienza comprobando si el nombre del estilo ya ha sido definido. Si el libro ya tiene un estilo con el nombre especificado, el procedimiento finaliza después de mostrar un mensaje al usuario. Si el nombre del estilo no existe, el procedimiento crea el estilo con el formato dado.

El estilo personalizado se agrega a la colección de estilos. Para averiguar el número de índice del estilo recién agregado simplemente vuelve a ejecutar este procedimiento.

Para aplicar un estilo a una o varias celdas mediante un procedimiento, podemos ejecutar la siguiente instrucción en la ventana **Inmediato** (o incluirla en una macro):

```
Selection.Style = "Formato Simple"
```

Para comprobar los ajustes de un estilo específico, seleccionamos el rango de celda y hacemos clic en la ficha **Inicio > Estilos** para mostrar la galería de etilos. Hacemos clic con el botón derecho del ratón en el nombre del estilo y seleccionamos **Modificar**. Excel muestra la ventana de la Imagen 2.20.

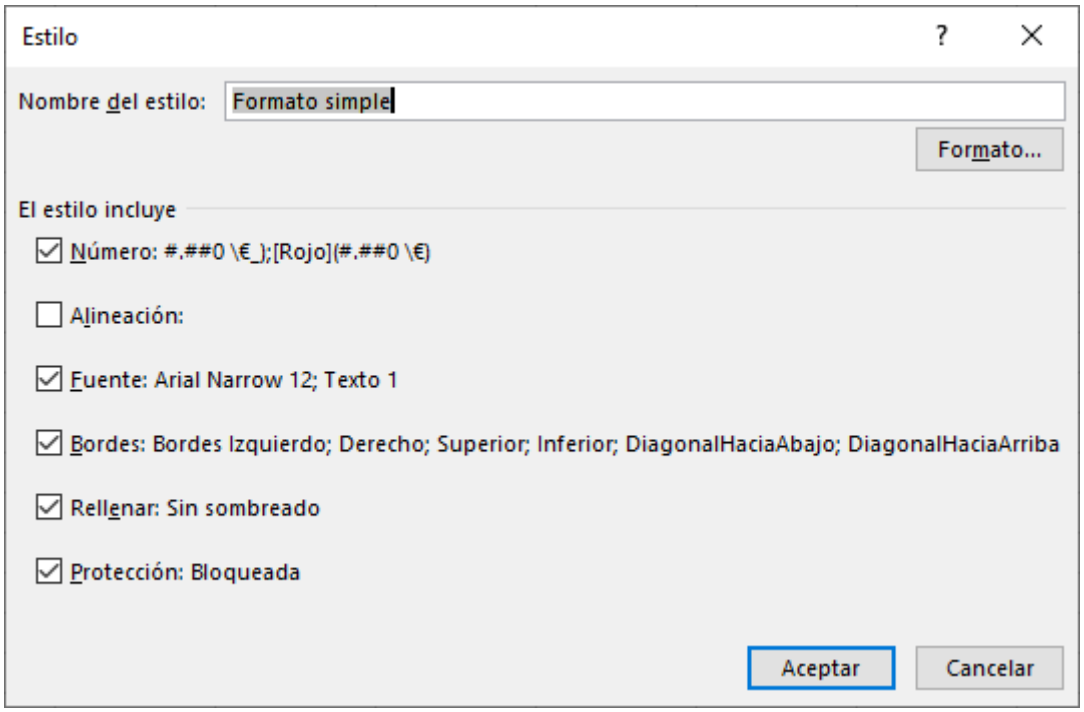


Imagen 2.20 Este cuadro de diálogo muestra los ajustes del estilo Formato simple que fue creado anteriormente.

El siguiente código elimina el formato aplicado al rango seleccionado:

```
Selection.ClearFormats
```

La declaración anterior devuelve el formato de la celda a su estado habitual pero no elimina el estilo de la colección. Si deseamos borrar un estilo de un libro, utilizamos la siguiente declaración:

```
ActiveWorkbook.Styles("Formato simple").Delete
```

Si una celda ya contiene un estilo que hemos aplicado manualmente, es posible crear un estilo nuevo a partir de esa configuración:

```
Sub EstiloSeleccion()
    Dim nuevoNombreEstilo As String

    nuevoNombreEstilo = "Importe Factura"
    ActiveWorkbook.Styles.Add Name:=nuevoNombreEstilo, _
    BasedOn:=ActiveCell
End Sub
```

De forma predeterminada, Excel crea un nuevo estilo basado en el estilo Normal. Sin embargo, si ya hemos aplicado formato a una celda en concreto y queremos guardar la configuración en un estilo, utilizamos el argumento opcional **BasedOn** de la colección **Styles**.

Los estilos personalizados que crees pueden ser reutilizados en otros libros. Para ello, debes copiar la información de estilo de un libro a otro. En VBA esto se puede hacer usando el método **Merge**:

```
ActiveWorkbook.Styles.Merge "Ventas2020.xlsx"
```

La línea anterior copia los estilos del libro actual en el libro que especifiquemos entre comillas.

3 Resumen

En este capítulo, has aprendido a utilizar VBA para aplicar algunas características básicas de formato a una hoja de Excel, con el fin de facilitar la lectura e interpretación de los datos. También has aprendido características de formato avanzadas como el formato condicional y a utilizar herramientas como barras de datos, conjuntos de iconos, escalas de color, formas y minigráficos, así como temas y estilos de celdas.

El próximo capítulo se centra en la cinta de opciones y los menús contextuales.

Capítulo 19
Personalización de la cinta de opciones y los menús contextuales

Los usuarios de aplicaciones informáticas esperamos encontrarnos con formas fáciles de utilizar cualquier aplicación de Windows ¿verdad? Una vez que hemos escrito procedimientos que dan soluciones a problemas específicos de automatización de hojas de cálculo deberíamos dedicar algo de tiempo a añadir alguna función que haga que nuestros libros sean rápidos y fáciles de usar.

En Excel existen dos zonas en las que los usuarios interactúan de una forma intensa: los menús contextuales y la cinta de opciones. Acceder a un comando específico debería ser fácil e intuitivo para cualquier persona.

De eso trata este capítulo. En él aprenderás a trabajar con los menús contextuales y la cinta de opciones mediante programación VBA.

1 Los menús contextuales

Un menú contextual aparece cuando se hace clic con el botón derecho del ratón en un objeto de la ventana de Excel. Es posible personalizar los menús contextuales predefinidos utilizando el objeto **`CommandBar`**. Esta sección se centra en el uso de las propiedades y métodos del objeto **`CommandBar`** para crear, modificar o deshabilitar menús contextuales en función de las necesidades de la aplicación.

Cada objeto de la colección **`CommandBars`** se llama **`CommandBar`**. El término **`CommandBar`** se usa para referirse a un solo menú contextual. Este objeto cuenta con la propiedad especial **`Type`**, que se puede usar para devolver el tipo de menú (ver siguiente tabla).

Tipo de objeto	Índice	Constante
Barra de herramientas	0	msoBarTypeNormal

Tipo de objeto	Índice	Constante
Barra de menú	1	msoBarTypeMenuBar
Menú contextual	2	msoBarTypePoput

Atención

En las versiones anteriores a Excel 2007, el objeto `CommandBar` se utilizaba para trabajar con las barras de menú y barras de herramientas. Desde que se introdujo la cinta de opciones, el objeto `CommandBar` solo puede utilizarse con menús contextuales. Más adelante en este capítulo aprenderemos a personalizar la cinta de opciones.

1.1 Modificar un menú contextual predefinido

Excel cuenta con 67 menús contextuales con diferentes conjuntos de opciones. Escribamos un procedimiento que imprima los nombres de los menús contextuales en la ventana **Inmediato**.

1. Crea un libro nuevo y guárdalo en la carpeta C:\Archivos Manual VBA, con el nombre Capítulo 19 – Menús contextuales y cinta de opciones.xlsm.
2. Activa el editor de VBA e inserta un módulo nuevo.
3. Cambia el nombre al módulo. Lo llamarás `MenusContextuales`
4. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub MenusContextuales()
    Dim myBar As CommandBar

    Dim counter As Integer
    For Each myBar In CommandBars
    If myBar.Type = msoBarTypePopup Then
    counter = counter + 1
    Debug.Print counter & ": " & myBar.Name
    End If
    Next
End Sub
```

   Observa el uso de la constante `msoBarTypePopup` para identificar el menú contextual en la colección `CommandBars`.

5. Ejecuta el procedimiento.
   El resultado será una lista de los menús contextuales mostrados en la ventana **Inmediato**.

Ahora que conoces los nombres de los menús contextuales de Excel, puedes añadir fácilmente otros comandos frecuentes a cualquiera de ellos. Veamos cómo añadir el comando **Insertar imagen** al menú contextual que se activa al hacer clic con el botón derecho en una hoja.

1. En la misma ventana **Código** introduce el siguiente procedimiento:

```
Sub AgregarMenuCelda()
    With Application.CommandBars("Cell")
        .Reset
        .Controls.Add(Type:=msoControlButton, _
        Before:=2).Caption = "Insertar imagen..."
        .Controls("Insertar imagen...").OnAction =
"InsertarImagen"
    End With
End Sub
Sub InsertarImagen()
    CommandBars.ExecuteMso ("PictureInsertFromFile")
End Sub
```

El método `Reset` del objeto `CommandBar` evita que se vuelva a colocar la misma opción en el menú cuando ejecutas el procedimiento más de una vez.

Para añadir un control predefinido o personalizado a un menú contextual, utiliza el método `Add` con la siguiente sintaxis:

```
CommandBar.Controls.Add(Type, Id, Parameter, Before,
Temporary)
```

`CommandBar` es el objeto al que se quiere añadir el control. `Type` es una constante que determina el tipo de control personalizado que quieres añadir. Puedes seleccionar uno de los siguientes tipos:

Control	Valor
msoControlButton	1
msoControlPopup	10
msoControlEdit	2
msoControlDropDown	3
msoControlComboBox	4

`Id` es un número entero que especifica el número de control predefinido que deseas añadir. `Parameter` se utiliza para enviar información a un procedimiento o para almacenar información sobre un control. El argumento `Before` es el número de índice

del control ante el cual se añadirá un nuevo control. Si se omite, VBA añade el control al final de la barra de comandos especificada.

El argumento `Temporary` es un valor lógico (`True` o `False`) que determina cuando será borrado el control. Cuando se establece este argumento en `True`, el control se eliminará automáticamente cuando se cierre la aplicación Excel.

Los controles del objeto `CommandBar` tienen una serie de propiedades que ayudan a especificar la apariencia y funcionalidad de un control. Por ejemplo, la propiedad `Caption` especifica el texto que se mostrará en el control. En el procedimiento anterior aparece la opción "Insertar imagen…" en el menú contextual **Cell**. Ten en cuenta que es costumbre agregar puntos suspensivos (…) al final del texto para indicar que la opción activará un cuadro de diálogo en el que el usuario tendrá que hacer más selecciones. La propiedad `OnAction` especifica el nombre de un procedimiento VBA que se ejecutará cuando se seleccione la opción del menú. En este ejemplo, al seleccionar la opción **Insertar imagen…**, se llamará al procedimiento `InsertarImagen`. Este procedimiento utiliza el método `ExecuteMso` del objeto `CommandBar` para ejecutar el comando `PictureInsertFromFile` de la cinta de opciones.

2. Ejecuta el procedimiento `MenusContextuales`.
3. Activa la ventana de Excel y haz clic con el botón derecho del ratón en cualquier celda. A continuación, selecciona **Insertar imagen…** (ver Imagen 1.1).

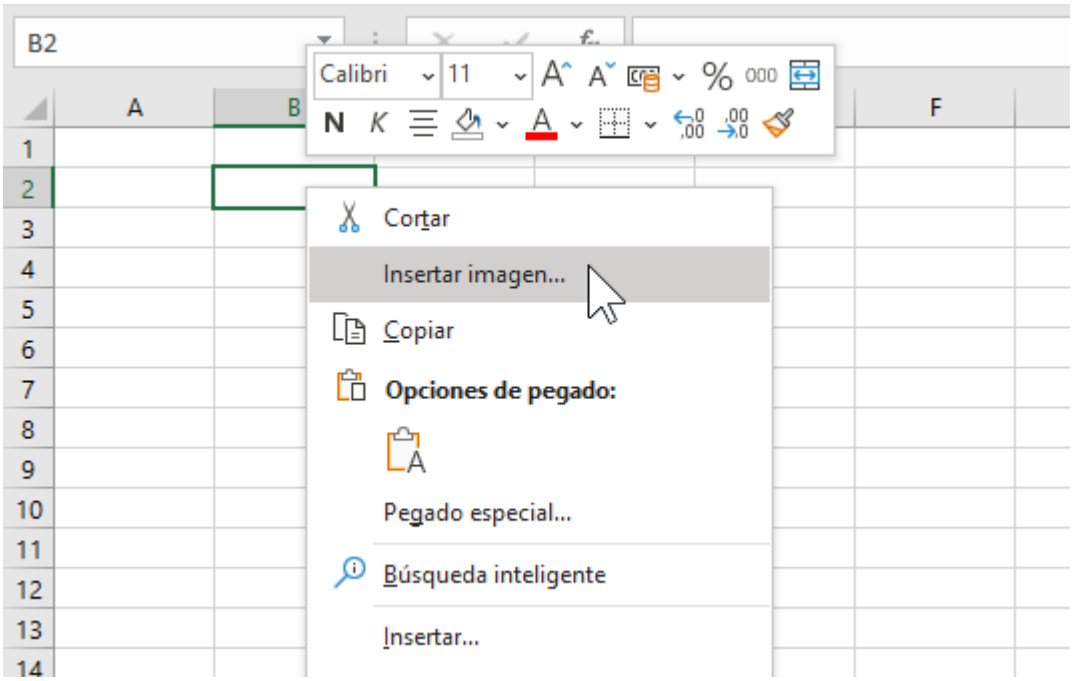


Imagen 1.1 El menú contextual Cell predefinido muestra un nuevo elemento (Insertar imagen...).

Excel muestra el cuadro de diálogo **Insertar imagen**, desde el que se puede insertar una imagen desde un archivo. Es el mismo cuadro que aparece cuando se hace clic en **Imagen**, en la ficha **Insertar**.

Atención

Los elementos agregados a los menús contextuales de Excel están disponibles en todos los libros de trabajo abiertos. No importa qué libro de trabajo se utilizó para añadir un elemento personalizado. Por esta razón es una buena práctica asegurarse de que el elemento del menú contextual se elimine cuando se cierre el libro de trabajo.

Observemos que algunas opciones del menú contextual van precedidas de una pequeña imagen. Escribamos otra versión del procedimiento `AgregarMenuCelda` para incluir una imagen junto al comando **Insertar Imagen…**

4. Introduce el siguiente procedimiento en el módulo:

```
Sub AgregarMenuCelda2()
    Dim ct As CommandBarButton
    With Application.CommandBars("Cell")
        .Reset
        Set ct = .Controls.Add(Type:=msoControlButton, _
        Before:=11, Temporary:=True)
    End With
    With ct
        .Caption = "Insertar Imagen..."
        .OnAction = "InsertarImagen"
        .Picture = Application.CommandBars. _
        GetImageMso("PictureInsertFromFile", 16, 16)
        .Style = msoButtonIconAndCaption
    End With
End Sub
```

En este procedimiento, se le indica a VBA que añada el elemento del menú en la posición 11 del menú contextual. También se especifica que esta opción de menú se elimine automáticamente al salir de Excel. Esto se logra estableciendo el valor del parámetro `Temporary` en True. A continuación, se utiliza el bloque de instrucciones `With … End With` para establecer un par de propiedades para el objeto de control recién creado (`ct`). Además de establecer dos propiedades estándar (`Caption` y `OnAction`), se asigna la imagen `imageMso` a la propiedad `Picture` del nuevo control. Para que se muestre la imagen debes usar el método `CommandBars.GetImageMso` y especificar el nombre de la imagen y su tamaño (ancho y alto). El tamaño de la imagen se especifica en 16 x 16 píxeles. La propiedad `Style` se utiliza aquí para especificar que el botón de control debe mostrar tanto el icono como su título.

1. Ejecuta el procedimiento `AgregarMenuCelda2`.
2. Activa la ventana de Excel, haz clic con el botón derecho del ratón en cualquier celda de la hoja y busca el comando **Insertar imagen…** (ver Imagen 1.2).

Observa que los comandos del menú contextual tienen una letra de su título subrayada. Es la tecla de acceso directo. Para invocar una opción del menú, basta con pulsar la letra subrayada después de abrir el menú.

3. Agrega una tecla de acceso rápido a la nueva opción personalizada modificando la propiedad `Caption` en el procedimiento anterior de esta manera:

```
.Caption = "In&sertar Imagen..."
```

El símbolo “&” delante de la letra S indica que la “s” mayúscula servirá como tecla de acceso directo. Recuerda que las teclas de acceso directo son únicas; no puedes usar una letra que esté en uso en otro elemento del mismo menú.

4. Después de volver a ejecutar el procedimiento modificado, activa a la ventana de Excel, haz clic con el botón derecho del ratón en cualquier celda de la hoja y presiona “s” minúscula.

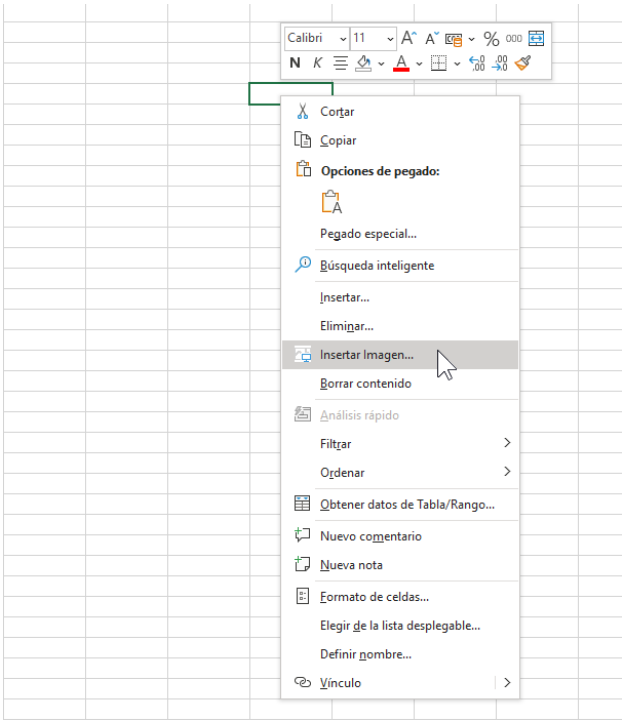


Imagen 1.2 El elemento personalizado Insertar imagen contiene ahora una imagen.

1.2 Eliminar un elemento personalizado del menú contextual

Cuando modificamos menús contextuales, las personalizaciones no desaparecerán cuando se cierra el libro. Únicamente reiniciar Excel eliminará los cambios del menú contextual, ysolo si se ajusta el valor del parámetro `Temporary` a `True` cuando se añade el elemento. Para asegurarnos de que el elemento personalizado se elimina del menú, consideremos la

posibilidad de escribir un procedimiento de eliminación similar al que se muestra a continuación:

```
Sub EliminarInsertarImagen()
    Dim c As CommandBarControl

    On Error Resume Next
    Set c = CommandBars("Cell").Controls("In&sertar Imagen...")
    c.Delete
End Sub
```

Para eliminar de forma automática el elemento del menú, llamaremos al procedimiento desde el procedimiento del evento `Workbook_BeforeClose`:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Call EliminarInsertarImagen
End Sub
```

Este procedimiento debe introducirse en el código del módulo `ThisWorkbook` para que se ejecute justo antes de que el libro se cierre.

De igual forma si queremos que el elemento personalizado se muestre automáticamente cuando abramos el libro, podemos introducir el siguiente procedimiento en el código de `ThisWorkbook`:

```
Private Sub Workbook_Open()
    Call AgregarMenuCelda2
End Sub
```

1.3 Deshabilitar y ocultar elementos de un menú contextual

Para impedir el uso de un elemento en particular en un menú contextual, es posible que desees desactivarlo u ocultarlo.

Cuando un elemento está desactivado, su título aparece atenuado. Cuando un elemento del menú está oculto, simplemente no aparece. Para deshabilitar el elemento del menú, establecemos la propiedad `Enable` del control en `False`. Por ejemplo, la siguiente declaración deshabilitará el comando **Insertar imagen** que hemos añadido anteriormente:

```
Application.CommandBars("Cell"). _
Controls("In&sertar Imagen...").Enabled = False
```

Para habilitar un elemento previamente deshabilitado, simplemente escribe el valor `True` en la propiedad `Enable`.

Para ocultar un menú, establecemos en `False` la propiedad `Visible`:

```
Application.CommandBars("Cell"). _
Controls("Insert Pict&ure...").Visible = False
```

Para hacer visible el elemento oculto, establecemos la propiedad Visible en True.

Un buen lugar para usar los comandos anteriores son los procedimientos de eventos **Worksheet_Activate** y **Worksheet_Deactivate**. Por ejemplo, para desactivar un elemento específico del menú solo cuando la Hoja1 esté activa, escribimos los siguientes procedimientos de eventos en el módulo de la **Hoja1**:

```
Private Sub Worksheet_Activate()
    Application.CommandBars("Cell").Controls("Ordenar"). _
    Enabled = False
End Sub
Private Sub Worksheet_Deactivate()
    Application.CommandBars("Cell").Controls("Ordenar"). _
    Enabled = True
End Sub
```

1.4 Agregar un menú contextual a un botón de comando

Cuando diseñemos formularios personalizados, puede que queramos añadir menús contextuales a algunos controles situados dentro. El siguiente conjunto de procedimientos muestra cómo al hacer clic con el botón derecho del ratón en un botón de comando, se puede ofrecer a los usuarios un conjunto de opciones entre las que elegir.

1. Introduce los siguientes procedimientos en el módulo **MenusContextuales**:

```
Sub CrearMenuContextual()
    Dim sm As Object

    Set sm = Application.CommandBars.Add _
    ("Mi ordenador", msoBarPopup)
    With sm
        .Controls.Add(Type:=msoControlButton). _
        Caption = "Sistema operativo"
        With .Controls("Sistema operativo")
            .FaceId = 1954
            .OnAction = "SistemaOperativo"
        End With
        .Controls.Add(Type:=msoControlButton). _
        Caption = "Impresora activa"
        With .Controls("Impresora activa")
            .FaceId = 4
            .OnAction = "ImpresoraActiva"
        End With
        .Controls.Add(Type:=msoControlButton). _
        Caption = "Libro activo"
        With .Controls("Libro activo")
```

```
            .FaceId = 247
            .OnAction = "LibroActivo"
        End With
        .Controls.Add(Type:=msoControlButton). _
        Caption = "Hoja activa"
        With .Controls("Hoja activa")
            .FaceId = 18
            .OnAction = "HojaActiva"
        End With
    End With
End Sub
```

Este procedimiento crea un menú contextual personalizado llamado "**Mi Ordenador**" y le agrega cuatro comandos. Fíjate que a cada comando se le asigna un icono. Al seleccionar un comando de este menú contextual, se ejecutará uno de los procedimientos siguientes:

2. En la ventana **Código** del módulo `MenusContextuales` introduce los siguientes procedimientos:

```
Sub SistemaOperativo()
MsgBox Application.OperatingSystem, , "Sistema operativo"
End Sub
Sub ImpresoraActiva()
MsgBox Application.ActivePrinter
End Sub
Sub LibroActivo()
MsgBox Application.ActiveWorkbook.Name
End Sub
Sub HojaActiva()
MsgBox Application.ActiveSheet.Name
End Sub
```

3. Ejecuta el procedimiento `CrearMenuContextual`.
   Para probar el menú personalizado utiliza el método `ShowPopup` del siguiente paso.
4. Introduce la siguiente instrucción en la ventana Inmediato y presiona **Intro**:

```
CommandBars("Mi Ordenador").ShowPopup 0, 0
```

   El método `ShowPopup` del objeto `CommandBar` acepta dos argumentos opcionales (x, y) que determinan la ubicación del menú contextual en la pantalla. En el ejemplo anterior, el menú contextual que añadió aparecerá en la esquina superior izquierda de la pantalla.

Hagamos nuestro menú contextual más amigable adjuntándolo a un botón de comando colocado en un `UserForm`.

5. En la pantalla del editor de VBA haz clic en **Insertar – UserForm** para agregar un nuevo formulario.
6. Utilizando el control **Botón de comando**, crea uno en cualquier lugar del formulario que acabas de insertar
   Usa la ventana **Propiedades** para cambiar la propiedad `Caption` del botón a "Información del sistema". Es posible que necesites modificar el tamaño del botón para que se adapte a la longitud del texto.
7. Activa la ventana **Código** del formulario haciendo doble clic en el botón **Código** en el **Explorador de proyectos** o haciendo doble clic dentro del formulario.
8. Introduce el siguiente procedimiento:

```
Private Sub CommandButton1_MouseDown(ByVal Button _
As Integer, _
ByVal Shift As Integer, _
ByVal X As Single, _
ByVal Y As Single)
    If Button = 2 Then
        Call MostrarMenuContextual
    Else
        MsgBox "Debes hacer clic con el botón derecho."
    End If
End Sub
```

Este procedimiento llama al procedimiento `MostrarMenuContextual` (ver paso 9) cuando el usuario hace clic con el botón derecho del ratón en el botón colocado en el formulario. VBA tiene dos procedimientos de eventos que se ejecutan en respuesta a un clic del ratón. Cuando se hace clic en un botón, VBA ejecuta el procedimiento `MouseDown`. Cuando se suelta el botón, se produce el evento `MouseUp`. Los dos procedimientos requieren los siguientes argumentos:

- `Object`: especifica el objeto. En este ejemplo, es el nombre del botón de comando.
- `Button`: es el valor entero que especifica qué botón del ratón se ha pulsado.

Valor del argumento Button	Descripción
1	Botón izquierdo
2	Botón derecho
3	Botón central

- `Shift`: Determina si el usuario estaba presionando las teclas Mayús, Ctrl o Alt cuando ocurrió el evento.

Valor del argumento Shift	Descripción
1	Mayús
2	Ctrl
3	Mayús + Ctrl
4	Alt
5	Alt + Mayús
6	Alt + Ctrl
7	Alt + Mayús + Ctrl

9. En la ventana **Código** del módulo `MenusContextuales`, introduce el siguiente procedimiento:

```
Sub MostrarMenuContextual()
    Dim shortMenu As Object

    Set shortMenu = Application.CommandBars("Mi ordenador")
    With shortMenu
        .ShowPopup
    End With
End Sub
```

10. En el **Explorador de proyectos**, haz doble clic en `UserForm1` y presiona **F5** para ejecutarlo. Haz clic con el botón derecho del ratón en el botón **Información del sistema** y selecciona uno de los elementos del menú contextual.
    Observa que el método `ShowPopup` utilizado en el procedimiento no incluye la ubicación en la pantalla. Por lo tanto, el menú aparece donde hiciste clic con el ratón.
11. Para eliminar el menú contextual, introduce y ejecuta el siguiente procedimiento en la ventana **Código** del módulo `MenusContextuales`.

1.5 Encontrar el valor FaceID de una imagen

Al crear nuestros propios menús contextuales, es posible que queramos incluir un icono o imagen junto al texto. La colección `CommandBars` tiene cientos de imágenes listas para usar. Cada botón de control de la barra de comandos tiene un **FaceID** que determina su aspecto. Pero, ¿cómo podemos saber qué ID pertenece a cada botón? El siguiente procedimiento realiza un bucle por la colección `CommandBars` y escribe en un libro nuevo una lista de botones con su número ID.

Si quieres ver este procedimiento en acción, introduce el código que se muestra a continuación en la ventana **Código** del módulo `MenusContextuales` y luego ejecútalo:

```
Sub Imagenes()
    Dim i As Integer
    Dim j As Integer
    Dim total As Integer
    Dim buttonId As Integer
    Dim buttonName As String
    Dim myControl As CommandBarControl
    Dim bar As CommandBar

    On Error GoTo GestionErrores
    Workbooks.Add
    Range("A1").Select
    With ActiveCell
        .Value = "Image"
        .Offset(0, 1) = "Índice"
        .Offset(0, 2) = "Nombre"
        .Offset(0, 3) = "FaceID"
        .Offset(0, 4) = "Nombre CommandBar"
    End With
    For j = 1 To Application.CommandBars.Count
        Set bar = CommandBars(j)
        total = bar.Controls.Count
        With bar
        For i = 1 To total
            buttonName = .Controls(i).Caption
            buttonId = .Controls(i).ID
            Set myControl = _
            CommandBars.FindControl(ID:=buttonId)
            myControl.CopyFace ' aquí podría procucirse un error
            ActiveCell.Offset(1, 0).Select
            Sheets(1).Paste
            With ActiveCell
                .Offset(0, 1).Value = buttonId
                .Offset(0, 2).Value = buttonName
                .Offset(0, 3).Value = myControl.FaceId
                .Offset(0, 4).Value = bar.Name & " (" & j & ")"
            End With
StartNext:
        Next i
        End With
```

```
        Next j
        Columns("A:E").EntireColumn.AutoFit
        Exit Sub
GestionErrores:
        Resume StartNext
End Sub
```

Debido a que no es posible copiar la imagen de un icono desactivado, VBA encuentra un error cuando intenta copiar el icono en el portapapeles. El procedimiento atrapa este error con la sentencia `On Error GoTo GestionErrores`. De esta forma, cuando VBA encuentre el error, saltará a la etiqueta `GestionErrores` y ejecutará las instrucciones que están debajo de ella. Esto asegurará que se salte ese botón y el procedimiento pueda continuar sin interrupción. En la Imagen 2.1 se muestra un resultado parcial del procedimiento.

2 Introducción a la cinta de opciones

La cinta de opciones está compuesta por la barra de título, la barra de herramientas de acceso rápido y las fichas. Cada ficha de la cinta de opciones da acceso a funciones y comandos relacionados con una tarea concreta. Por ejemplo, podemos utilizar la ficha **Insertar** para insertar rápidamente tablas, imágenes, gráficos, enlaces o texto (ver la Imagen 2.2). Los comandos y botones dentro de una pestaña se organizan en grupos. Este tipo de organización facilita la localización de un comando concreto.

	A	B	C	D	E	F
72		464	&Ocultar det	464	PivotTable (12)	
73		462	&Mostrar de	462	PivotTable (12)	
74		459	&Actualizar (	459	PivotTable (12)	
75		20857	&Incluir eler	20857	PivotTable (12)	
76		5920	&Mostrar sie	5920	PivotTable (12)	
77		460	&Configurac	460	PivotTable (12)	
78		3790	&Mostrar list	3790	PivotTable (12)	
79		917	Objeto &sele	917	Chart (13)	
80		439	&Leyenda	439	Chart (13)	
81		987	&Tabla de da	987	Chart (13)	
82		989	Series en &fi	989	Chart (13)	
83		988	Series en &c	988	Chart (13)	
84		1014	Á&ngulo des	1014	Chart (13)	
85		1013	Ángulo &asc	1013	Chart (13)	
86		1589	Nue&vo com	1589	Reviewing (14)	
87		1590	&Comentari	1590	Reviewing (14)	
88		1591	&Comentari	1591	Reviewing (14)	
89		1593	M&ostrar u c	1593	Reviewing (14)	

Imagen 2.1 Una lista de iconos con sus correspondientes valores, generados por el procedimiento.

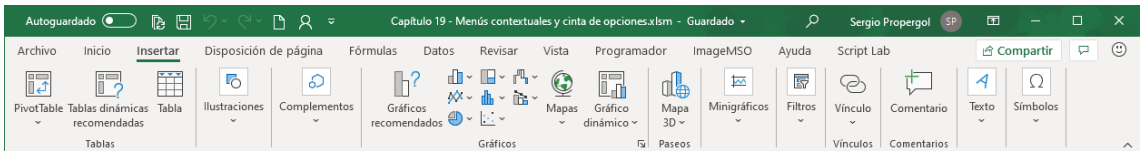


Imagen 2.2 En la cinta de opciones, cada ficha contiene grupos y comandos relacionados entre sí.

Los botones de los programas pueden ser grandes o pequeños. Un botón grande indica un comando de uso frecuente, mientras que un botón pequeño muestra una característica específica del comando principal con la que puede que queramos trabajar. Algunos de estos botones incluyen listas desplegables con otros comandos más especializados. Por ejemplo, el botón "Más funciones" situado en el grupo **Biblioteca de funciones** de la ficha **Fórmulas** contiene otros tipos de funciones con las que podemos trabajar: Estadística, Ingeniería, Cubo, etc.

Algunos controles de la cinta no muestran comandos accionables. En su lugar, nos ofrecen una ayuda visual del resultado que podemos esperar cuando se selecciona una opción específica. Estos tipos de controles se conocen como galerías. El control de galería se utiliza a menudo para presentar opciones de formato, como los ajustes de margen que se muestran en la Imagen 2.3.

Como se mencionó anteriormente, los comandos de las fichas de la cinta están organizados en grupos para facilitar la navegación. Algunos grupos tienen "lanzadores de cuadros de diálogo" en la esquina inferior derecha que muestran un cuadro de diálogo en el que se pueden establecer varias opciones avanzadas a la vez.

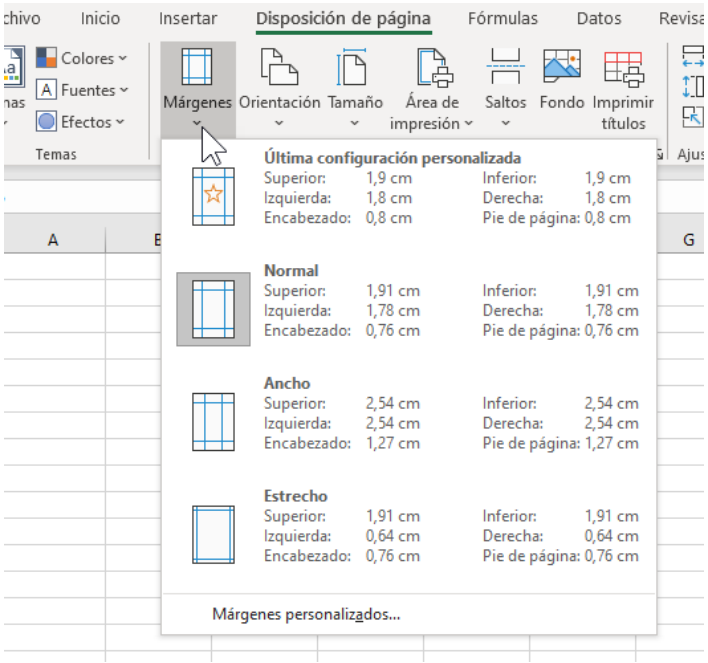


Imagen 2.3 Los diseños de márgenes se muestran en un control de galería.

Además de las fichas principales, también hay fichas contextuales, que contienen comandos que se aplican al objeto con el que estamos trabajando. Cuando seleccionamos un objeto, en

la cinta se muestra una de estas fichas. Por ejemplo, cuando seleccionamos una imagen en una hoja, la cinta de opciones muestra una ficha contextual llamada **Herramientas de imagen**, como se muestra en la Imagen 2.4.

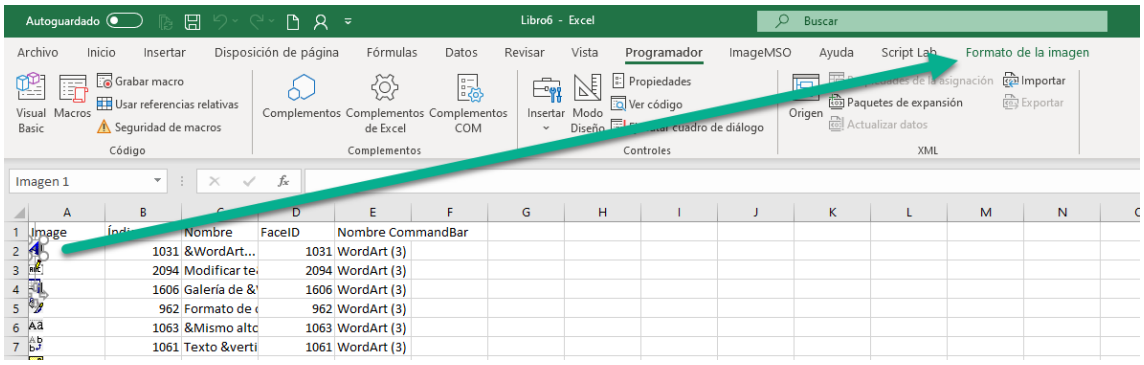


Imagen 2.4 Las fichas contextuales aparecen mientras estamos trabajando con determinados objetos como imágenes, tabla dinámicas o gráficos.

Al hacer clic en la imagen se activa la ficha **Formato de la imagen**, que tiene comandos para trabajar con objetos de imagen. La ficha contextual desaparece cuando se deja de seleccionar el objeto. Dicho de otra forma, si seleccionas una celda de la hoja, la ficha **Formato de la imagen** desaparecerá.

Al detener el ratón encima de un comando aparecerá un cuadro con ayuda, el método abreviado del teclado y una descripción de lo que hace el comando (ver Imagen 2.5).

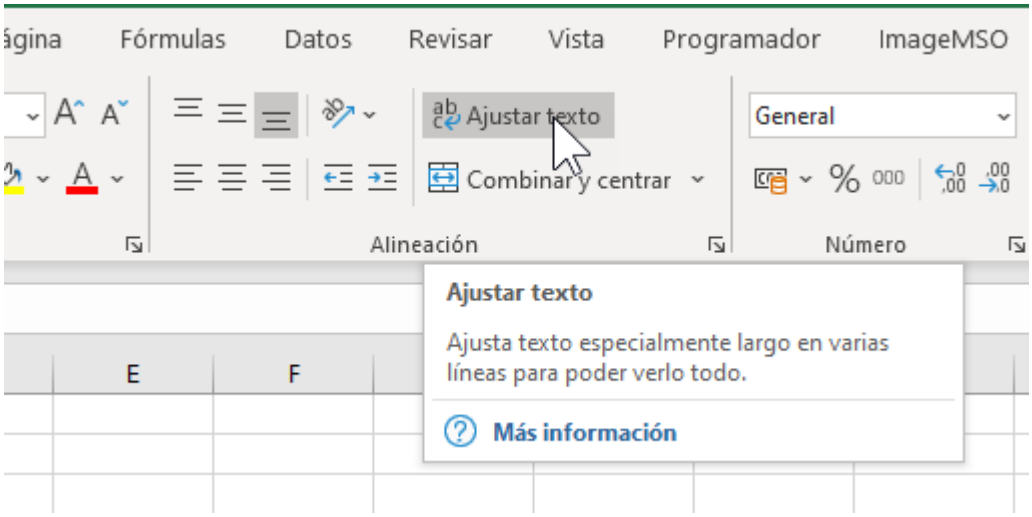


Imagen 2.5 La ayuda de las herramientas nos dan más información sobre el comando seleccionado.

Todos los comandos de la cinta de opciones y de la barra de herramientas de acceso rápido tienen su método abreviado de teclado. Basta con presionar la tecla **Alt** del teclado para mostrar recuadros sobre las pulsaciones que debemos ir haciendo hasta llegar al lugar deseado. Cada comando tiene su propia tecla. Por ejemplo, para acceder a la ficha Archivo, presiona Alt y luego la letra A. Dentro de los menús veremos otros consejos clave para cada comando. Para ver los consejos de esas teclas para los comandos de una ficha en particular, primero seleccionamos la tecla de acceso a esa ficha. Para ocultar las sugerencias de teclas, volvemos a presionar la tecla Alt. Cuando trabajemos en el modo de comandos (después de

presionar la tecla Alt), también podemos utilizar la tecla de tabulación y las teclas de flechas para desplazarnos por la cinta.

Ahora que hemos visto las principales características de la interfaz de la cinta de opciones, veamos cómo se pueden ampliar con nuestras propias fichas y controles.

3 Programación de la cinta de opciones con XML y VBA

Los componentes de la cinta de opciones pueden ser manipulados mediante programación usando XML u otros lenguajes de programación. Todas las aplicaciones de Office que utilizan esta interfaz se basan en el modelo de programación conocido como *Ribbon Extensibility* (RibbonX). En esta sección nos introduciremos en la personalización de la interfaz de usuario (UI) de Microsoft 2019 Office Fluent utilizando XML. Aunque no se requieren herramientas especiales para realizar personalizaciones en la cinta de opciones, es mucho más rápido y fácil trabajar con la aplicación Custom UI Editor. Por lo tanto, en los siguiente ejemplos, utilizaremos esta herramienta gratuita para crear las personalizaciones de la cinta de opciones.

En los archivos que acompañan al manual encontrarás OfficeCustomUIEditorSetup.zip

Importante

Al descomprimir el archivo OfficeCustomUIEditorSetup.zip, deberías obtener el archivo instalador con extensión .msi. Ejecuta el archivo de instalación para trabajar con las personalizaciones de la cinta de opciones.

Podemos encontrar los nombres de los controles de la cinta de opciones descargando las tablas de identificadores de comandos Office Fluent User Interface Control Identifiers:

https://www.microsoft.com/en-us/download/details.aspx?id=50745

3.1 Crear una personalización XML

Para hacer cambios personalizados en la interfaz de usuario de la cinta de opciones debemos preparar un archivo XML que especifique todas estas personalizaciones. El contenido del archivo XML que utilizaremos en el siguiente ejercicio se muestra en la Imagen 3.1 y el resultado obtenido en la Imagen 3.2.

1. Abre Excel y crea un libro llamándolo Capítulo 19 – Modificación cinta.xlsm y guárdalo en la carpeta C:\Archivos Manual VBA asegurándote de que el formato es adecuado para macros.
2. Cierra el archivo y sal de Excel.
3. Abre la aplicación **Custom UI Editor For Microsoft Office**.
4. Haz clic en **Archivo (File) – Abrir (Open)**.
5. Selecciona el archivo que creaste en el paso 1 y haz clic en **Abrir**.

Imagen 3.1 Este código define dos grupos en una ficha nueva de la cinta de opciones.

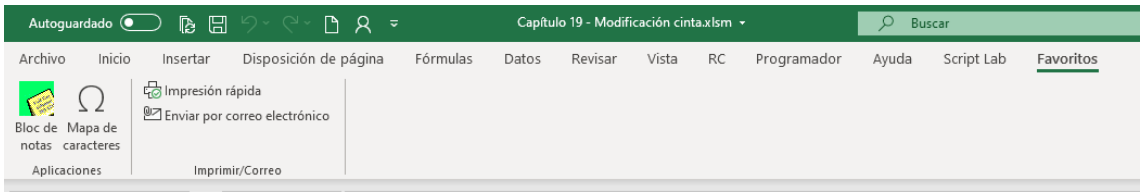


Imagen 3.2 La ficha Favoritos está creada a partir de un archivo XML.

6. Haz clic en el menú **Insertar – Office 2010 Custom UI Part**, como se muestra en Imagen 3.3. Esto crea el archivo CustomUI14.xml en el libro.

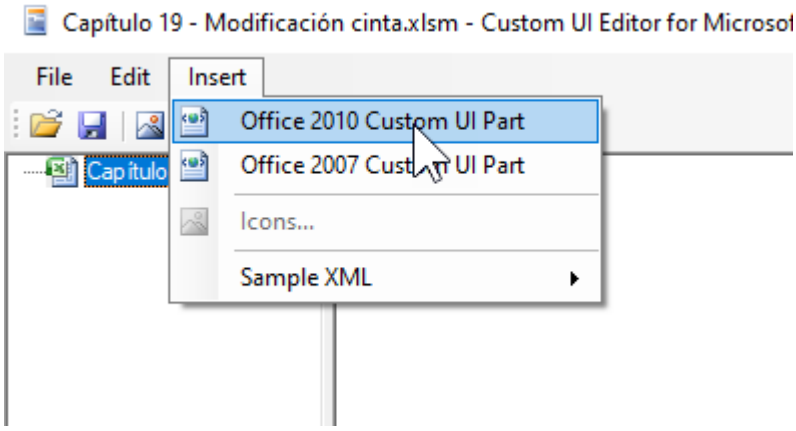


Imagen 3.3 La opción Office 2010 Custom UI Part funciona para todas las versiones de Excel de 2010 en adelante.

7. En el panel derecho de la aplicación, introduce el siguiente código (ver Imagen 3.3.

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
    <ribbon startFromScratch="false">
        <tabs>
```

```
            <tab idMso="TabHome">
                <group idMso="GroupStyles" visible="false" />
            </tab>
            <tab id="TabJK1" label="Favoritos">
                <group id="GroupJK1" label="Aplicaciones">
                    <button id="btnNotes" label="Bloc de notas"
image="Note1"
                      size="large"
                      onAction="OpenNotepad" screentip="Abre el Bloc de
notas de Windows"
                      supertip="Es recomendable guardar tus notas sobre
este libro en un archivo de texto."/>
                    <button id="btnCharMap" label="Mapa de caracteres"
                      imageMso="SymbolInsert" size="large"
                      onAction="OpenCharmap" />
                </group>
                <group id="GroupJK2" label="Imprimir/Correo" >
                    <button idMso="FilePrintQuick" size="normal" />
                  <button idMso="FileSendAsAttachment" size="normal" />
                </group>
            </tab>
        </tabs>
    </ribbon>
</customUI>
```

XML es sensible a mayúsculas, por lo que debes introducir el código de la misma forma en que aparece arriba.

8. Haz clic en el botón **Validar** de la barra de herramientas para verificar que el código XML no contiene errores. Debería aparece un mensaje indicándolo. Si hay errores, debes corregirlos para asegurarte de que el XML está bien creado.

Echemos un vistazo al código. Un archivo XML consiste en un determinado número de elementos llamados nodos. En cualquier documento debe haber un nodo raíz, o un elemento de nivel superior. En nuestro código, la etiqueta raíz es **<customUI>**. El objetivo es especificar el espacio de nombres XML:

```
<customUI xmlns="http://schemas.microsoft.com/office/2009/07/customui">
```

Los nombres de espacios se utilizan para identificar de forma exclusiva los elementos de los documentos XML y evitar conflictos de nombres cuando se combinan elementos con el mismo nombre en el mismo documento.
El atributo **xmlns** de la etiqueta <**customUI**> contiene el nombre del espacio predeterminado que se utilizará en la personalización de la cinta. Observemos que el elemento raíz encierra todos los demás elementos del documento: cinta de opciones, fichas, grupos y botones. Cada elemento consta de una etiqueta de inicio y una de fin.

Para crear una nueva ficha en la cinta, usamos la etiqueta <tabs>. Cada elemento de la ficha se define con la etiqueta <tab>. El atributo label de cada elemento de la ficha especifica el nombre de la ficha personalizada. El nombre del atributo id se utiliza para identificar la ficha:

```
<tabs>
    <tab id="TabJK1" label="Favoritos">
```

Las fichas de la cinta contienen controles organizados en grupos. Podemos definir un grupo para los controles creando una etiqueta <group>. En nuestro archivo hemos definido dos grupos:

```
<group id="GroupJK1" label="Aplicaciones">
<group id="GroupJK2" label="Imprimir/correo">
```

Al igual que el nodo de la ficha, los nodos de grupo del documento XML también contienen los atributos id y label. Colocar controles en grupos es fácil. El grupo **Aplicaciones** tiene dos controles de botón personalizados, identificados por los elementos <button>. El grupo denominado **Impimir/Correo** también contiene dos botones; sin embargo, a diferencia del grupo **Aplicaciones**, los botones que se colocan aquí son controles predefinidos del sistema de Office en lugar de controles personalizados. Podemos determinar esto rápidamente mirando el atributo id del control. Cualquier atributo que termine con "mso" se refiere a un elemento de Office.

```
<button idMso="FilePrintQuick" size="normal" />
```

Los botones de la cinta de opciones pueden ser grandes o pequeños. Podemos definir el tamaño del botón estableciendo el atributo size como large o normal. Los botones pueden tener otros atributos adicionales:

```
<button id="btnNotes" label="Bloc de notas" image="Note1"
size="large"
onAction="OpenNotepad" screentip="Abre el Bloc de notas de
Windows"
supertip="Es recomendable guardar tus notas sobre este libro en
un archivo de texto."/>
<button id="btnCharMap" label="Mapa de caracteres"
       imageMso="SymbolInsert" size="large"
       onAction="OpenCharmap" />
```

Los atributos screentip y supertip permiten especificar un texto corto y otro largo que aparecen cuando se coloca el puntero del ratón sobre el botón.

El atributo imageMso indica el nombre del icono de Office existente. Podemos utilizar las imágenes proporcionadas por cualquier aplicación de Office.

Los controles realizan las tareas designadas mediante procedimientos de llamada. Por ejemplo, el atributo onAction de un botón contiene el nombre del procedimiento a ejecutar cuando se hace clic en el botón.

Antes de finalizar el documento XML debemos asegurarnos de que hemos incluido todas las etiquetas de cierre.

```
    </tab>
  </tabs>
 </ribbon>
</customUI>
```

Como esta personalización incluye una imagen personalizada, vamos a agregarla al archivo.

9. Copia la carpeta **Imágenes** incluida con el manual, en la carpeta C:\Archivos Manual VBA.
10. En el editor de UI de Office, selecciona **CustomUI14.xml** en el panel izquierdo y haz clic en el menú **Insert – Icons**.
11. Cambia el filtro de los archivos a mostrar para poder visualizar los archivos .gif y selecciona **Note.gif**. Haz clic en **Abrir**.
12. En el panel izquierdo haz clic en el signo más (+) en la parte izquierda del archivo CustomUI.xml. Debería aparecer la imagen Nota. Cambia el nombre de la imagen a Note1 para hacerla coincidir con el nombre del atributo `image` en el archivo XML:

```
<button id="btnNotes" label="Bloc de notas" image="Note1"
 size="large"
 onAction="OpenNotepad" screentip="Abre el Bloc de notas de
Windows"
 supertip="Es recomendable guardar tus notas sobre este libro en
un archivo de texto."/>
```

13. Selecciona **File – Save** para guardar el archivo. A continuación sal del Custom UI Editor.

Hemos completado la primera parte de la personalización. En la siguiente parte abriremos el archivo de Excel y veremos la ficha personalizada que acabamos de crear. También escribiremos algunos procedimientos de llamada que realicen algunas acciones.

3.2 Cargar las personalizaciones de la cinta

Vamos a finalizar los pasos necesarios para integrar las personalizaciones en el libro de Excel:

1. Abre el archivo Capítulo 19 – Modificación cinta.xlsm. Observa que se muestra la nueva ficha **Favoritos**.
2. Activa el editor de VBA e inserta un módulo nuevo.
3. En el módulo introduce los siguientes procedimientos:

```
Public Sub OpenNotepad(ctl As IRibbonControl)
    Shell "Notepad.exe", vbNormalFocus
End Sub
Public Sub OpenCharmap(ctl As IRibbonControl)
    Shell "Charmap.exe", vbNormalFocus
End Sub
```

`OpenNotepad` y `OpenCharmap` son los nombres de los procedimientos que se especificaron en el atributo `onAction` del botón. Como se ha mencionado anteriormente, un procedimiento de devolución de llamada ejecuta alguna acción y luego notifica a la cinta que la tarea ha sido completada. La devolución de llamada `onAction` se gestiona mediante un procedimiento VBA. La devolución de la llamada incluye el parámetro `lRibbonControl` que es el control en el que se ha hecho clic. Este control se pasa al código VBA por la cinta.

```
Sub OpenNotepad(ctl as IRibbonControl)
Sub OpenCharmap(ctl as IRibbonControl)
```

Para que VBA reconozca este parámetro, debemos asegurarnos de que el cuadro de diálogo **Referencias** (Herramientas – Referencias) tenga una referencia a la biblioteca de objetos de Microsoft Office 16.0.

Los procedimientos `OpenNotepad` y `OpenCharmap` le indican a Excel que utilice la función `Shell` para abrir el Bloc de notas de Windows o el Mapa de caracteres. Observemos que el nombre del archivo ejecutable del programa está entre comillas. El segundo argumento de la función `Shell` es opcional. Especifica el estilo de la ventana, es decir, como aparecerá el programa una vez sea ejecutado. La constante `vbNormalFocus` abrirá la aplicación en una ventana de tamaño normal con foco.

4 Personalizar la ficha Archivo

La ficha **Archivo** es el punto de entrada a la interfaz de Office. También es conocida como "Vista backstage". Esta vista está diseñada específicamente para trabajar con libros. Contiene comandos que proporcionan acceso rápido a las funciones más comunes con libros como guardar, abrir o cerrar. Aquí también se encuentra los comandos **Salir** y **Opciones**, para personalizar numerosas características de Excel. Además de los comandos, la barra de navegación en el lado izquierdo incluye varias pestañas que agrupan las tareas relacionadas. Por ejemplo, al hacer clic en la ficha **Imprimir** de la barra de navegación, se muestra toda la información relacionada con impresoras y permite acceder y cambiar fácilmente muchos ajustes de impresión. Una gran área en la vista backstage de impresión se utiliza para la **Vista preliminar de impresión**. La pestaña **Información** organiza las tareas relacionadas con los permisos del libro de trabajo, versiones de archivos, intercambios y otras muchas propiedades.

La vista backstage es el lugar perfecto para incluir personalizaciones que presentan resúmenes de procesos o flujos de trabajo. En esta sección haremos un par de personalizaciones sencillas para pasar más tarde a tareas más avanzadas.

El código XML correspondiente a la vista backstage debe introducirse entre los elementos `<backstage></backstage>` dentro de las etiquetas `<customui></customui>` y debajo de cualquier personalización de la cinta. El siguiente código XML añade un botón personalizado llamado **Sincronizar** y una pestaña personalizada llamada "Posibilidades sin fin" (puedes descargarlo del archivo Capítulo 19 – XML Backstage.txt).

Atención

Para una introducción avanzada a la vista Backstage, puede que queramos descargar los siguientes documentos de Microsoft (los documentos son de versiones de Excel 2010 que son aplicables a cualquier versión de Excel posterior).

Customizing the Office 2010 Backstage View for Developers from

http://msdn.microsoft.com/en-us/library/ee815851(printer).aspx

Dynamically Changing the Visibility of Groups and Controls in the Office 2010 Backstage View

http://msdn.microsoft.com/en-us/library/ff645396(printer).aspx

```
<backstage>
    <button id="btnSync" label="Sincronizar" imageMso="SyncNow"
isDefinitive="true"
     insertBeforeMso="FileClose"
onAction="onActionCopyToArchive" />
    <tab id="mySpecialTab" label="Posibilidades sin fin"
insertAfterMso="TabRecent">
     <firstColumn>
       <group id="grp01" label="Grupo inicio" helperText="Este
es el texto de ayuda del grupo 1">
            <topItems>
               <button id="myButton1" label="Mi botón" />
            </topItems>
      </group>
      <group id="gr02" label="Hoja de trucos">
            <topItems>
               <button id="myButton2" label="Propuestas de
trucos (ideas)" />
            </topItems>
            <bottomItems>
              <layoutContainer id="set1"
layoutChildren="horizontal" >
                 <editBox id="item1" label="Truco 1"  />
                 <editBox id="item2" label="Truco 2"  />
              </layoutContainer>
            </bottomItems>
      </group>
    </firstColumn>
    <secondColumn>
```

```
      <group id="grpHyperlinks" label="Webs de acceso frecuente"
visible="true">
        <primaryItem>
         <button id="top1" label="Botón principal"
imageMso="HyperlinkProperties" />
        </primaryItem>
       <topItems>
           <button id="msft" label="Microsoft"
onAction="onActionExecHyperlink" />
           <layoutContainer id="set2" layoutChildren="vertical"
>
              <hyperlink id="YouTube"
label="https://www.YouTube.com"
                  onAction="onActionExecHyperlink" />
              <hyperlink id="amazon"
label="https://www.amazon.com"
                  onAction="onActionExecHyperlink" />
              <hyperlink id="ayu" label="https://ayudaexcel.com"
                  onAction="onActionExecHyperlink" />
           </layoutContainer>
       </topItems>
      </group>
     </secondColumn>
  </tab>
 </backstage>
```

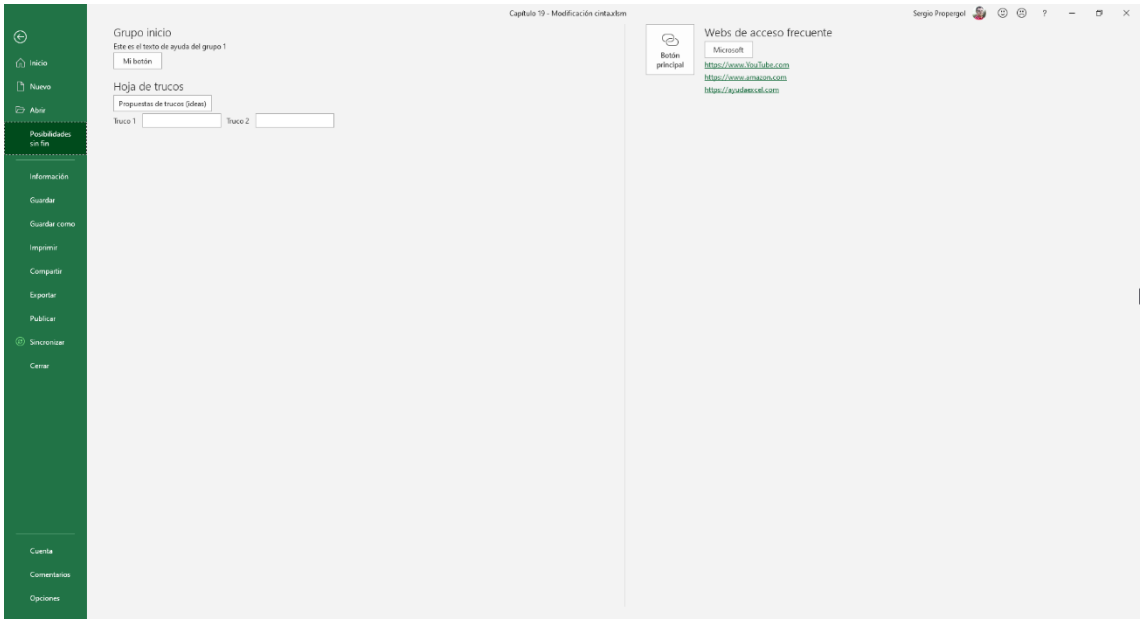


Imagen 4.1 La vista Backstage es muy personalizable. La pantalla se creó agregando un poco de código XML en el archivo de personalización de la cinta.

En el ejemplo anterior, el elemento **<button>** se utiliza para incorporar en la barra de navegación de la vista backstage un comando personalizado llamado **Sincronizar**:

```
<button id="btnSync" label="Sincronizar" imageMso="SyncNow"
isDefinitive="true"
     insertBeforeMso="FileClose"
onAction="onActionCopyToArchive" />
```

El elemento <**button**> contiene el atributo **isDefinitve**. Cuando este atributo se establece en True, al hacer clic en el botón se activará el procedimiento definido en el atributo **onAction** y luego se cerrará automáticamente la vista backstage y se volverá a la hoja.

A continuación, se muestra la llamada de **onAction** del botón **Sincronizar**. Observa que llama al procedimiento **CopyToArchive**. Este procedimiento nos permite hacer una copia del archivo actual del libro de trabajo en una carpeta elegida por nosotros. Debemos asegurarnos de introducir el código del procedimiento en el módulo estándar del libro Capítulo 19 – Modificación cinta.xlsm.

```
Sub onActionCopyToArchive(ctl As IRibbonControl)
Archivo
End Sub
Sub Archivo()
    Dim nombreCarpeta As String
    Dim miUnidad As String
    Dim CopiaSeguridad As String

    Application.DisplayAlerts = False
    On Error GoTo GestionErrores
    nombreCarpeta = ActiveWorkbook.Path
    If nombreCarpeta = "" Then
        MsgBox "No es posible copiar el archivo. " & Chr(13) _
        & "Este archivo no ha sido guardado.", _
        vbInformation, "Archivo"
    Else
        With ActiveWorkbook
            If Not .Saved Then .Save
                miUnidad = InputBox("Introduce la ruta:" & _
                Chr(13) & "(Por ejemplo: " & _
                "C:\Archivos\, etc.)", _
                "Localización del archivo?", "C:\")
                If miUnidad <> "" Then
                    If Right(miUnidad, 1) <> "\" Then
                        miUnidad = miUnidad & "\"
                    End If
```

```
                    CopiaSeguridad = miUnidad & .Name
                    .SaveCopyAs Filename:=CopiaSeguridad
                    MsgBox .Name & " fue copiado en: " _
                    & miUnidad, , "Fin del archivado"
                End If
            End With
        End If
        GoTo FinProcedimiento
    GestionErrores:
        MsgBox "VBA no puede encontrar la " & _
        "ruta especificada (" & miUnidad & ")" & Chr(13) & _
        "para el archivo. Por favor, inténtalo de nuevo.", _
        vbInformation + vbOKOnly, "Unidad de disco o " & _
        "carpeta no existe"
    FinProcedimiento:
        Application.DisplayAlerts = True
    End Sub
```

El código XML anterior también añade a la barra de navegación de la vista backstage una pestaña personalizada llamada **Personalización sin fin**. Cada elemento <`tab`> puede tener una o más columnas. Nuestro ejemplo contiene dos columnas. Cada ficha puede contener múltiples elementos <`group`>. Aquí tenemos dos grupos en la primera columna y un grupo en la segunda. Estos grupos pueden tener diferentes tipos de controles. Podemos agrupar los controles en tres tipos de secciones que se enumeran a continuación:

<primaryItem>	Se utiliza para especificar el elemento más importante del grupo. El control del elemento principal puede ser un botón o un menú con botones, botones de alternancia, casillas de verificación u otro menú.
<topItems>	Este elemento define los controles que aparecerán en la parte superior del grupo.
<bottomItems>	Este elemento define los controles que aparecerán en la parte inferior del grupo.

La disposición de los controles en la vista backstage se define mediante el elemento <`layoutContainer`>. El atributo `layoutChildren` de este elemento puede definir la disposición de los controles como horizontal o vertical.

La segunda columna de nuestro ejemplo XML utiliza el siguiente procedimiento de devolución de llamada para el botón etiquetado como Microsoft y los tres hipervínculos. Introduce el siguiente hipervínculo en el módulo:

```
Sub onActionExecHyperlink(ctl As IRibbonControl)
```

```
        Select Case ctl.ID
        Case "YouTube"
        ThisWorkbook.FollowHyperlink Address:= _
        "https://www.YouTube.com", NewWindow:=True
        Case "amazon"
        ThisWorkbook.FollowHyperlink Address:= _
        "https://www.amazon.com", NewWindow:=True
        Case "ayu"
        ThisWorkbook.FollowHyperlink Address:= _
        "https://ayudaexcel.com", NewWindow:=True
        Case "msft"
        ThisWorkbook.FollowHyperlink Address:= _
        "https://www.Microsoft.com", NewWindow:=True
        Case Else
        MsgBox "You clicked control id " & ctl.ID & _
        " that has not been programmed!"
        End Select
    End Sub
```

Atención

Algunas consideraciones al personalizar la vista backstage:

- El número máximo de pestañas permitidas es de 255.
- No se pueden reordenar las pestañas predefinidas.
- Podemos agregar nuestras pestañas antes o después de las pestañas predefinidas.
- No se puede modificar el diseño de las columnas de ninguna ficha predefinida.
- No se pueden ordenar los grupos predefinidos; sin embargo podemos especificar el orden de los grupos que creemos.

5 Resumen

En este capítulo has aprendido a utilizar VBA para trabajar con los menús contextuales predefinidos y a personalizar la cinta de opciones y la vista backstage utilizando una combinación de XML y VBA. Mientras trabajabas con los menús contextuales, aprendiste varias propiedades y métodos del objeto `CommandBar`. También aprendiste a utilizar la aplicación **Custom UI Editor for Microsoft Office** para crear el código XML necesario para incluir controles en la cinta de opciones y escribiste varios procedimientos que los conectaban con Excel.

El conocimiento y la experiencia que has adquirido en este capítulo puede utilizarse para realizar personalizaciones similares en todas las aplicaciones de Microsoft Office que utilizan la interfaz de cinta de opciones

En el siguiente capítulo, nos centramos en escribir código VBA que maneje la impresión y el envío de correos electrónicos.

Capítulo 20
Imprimir documentos y enviar correos desde Excel

Después de crear una hoja de cálculo, querrás que la gente la vea.

Excel proporciona comandos y botones fáciles de usar para imprimir y enviar por correo electrónico los libros de trabajo. Además, los programadores pueden controlar estas tareas con código VBA. Excel ofrece un acceso fácil a todas las funciones de impresión. Desde el punto de vista del usuario, se puede acceder a todas las opciones de impresión en la vista backstage, seleccionando **Archivo > Imprimi**r. Cuando se selecciona el comando imprimir, el lado izquierdo de la vista backstage muestra todas las opciones disponibles para imprimir, así como el botón **Imprimir** para ejecutar la impresión. Automáticamente aparece la vista previa de impresión de la hoja en la columna derecha de la ventana.

En este capítulo trabajarás con las funciones de impresión desde VBA. Aprenderás los métodos para acceder y configurar las opciones de impresión y para mostrar la vista previa de impresión utilizando declaraciones VBA. Estas declaraciones te permitirán también mostrar el cuadro de diálogo **Impresión** y la ventana **Vista previa de impresión**. Usarás estas declaraciones para configurar automáticamente las impresoras y las opciones de impresión en tus macros. De manera similar a la cinta de opciones, las características de impresión que se muestran en la vista backstage no se pueden modificar con VBA.

Además de la impresión, este capítulo también muestra cómo puedes automatizar las funciones de envío de correo electrónico de Excel (incluido el envío de correos masivos) con VBA.

1 Controlar la configuración de la página

Podemos controlar el aspecto de nuestras hojas de cálculo desde la ficha **Disposición de página** (o **Diseño de página** en algunas versiones de Excel). La ficha **Disposición de página**, que se muestra en la Imagen 1.1, está dividida en grupos que incluyen ajustes relacionados con la

configuración de la página (márgenes, orientación y tamaño), la escala y las opciones de la hoja.

Imagen 1.1 La ficha Disposición de página permite especificar los márgenes de la página, la orientación, el tamaño del papel y las opciones de escalado de hoja junto con otros ajustes.

Podemos acceder desde VBA a estos ajustes desde el cuadro de diálogo **Configurar página**, utilizando las propiedades del objeto `PageSetup`. Para mostrar el cuadro de diálogo **Configurar página**, escribe la siguiente declaración en la ventana **Inmediato** y presiona **Intro**:

```
Application.Dialogs(xlDialogPageSetup).Show
```

La declaración anterior usa el método `Show` del objeto `Dialogs` para mostrar el cuadro de diálogo predeterminado **Configurar página**. Podemos introducir una lista de argumentos después del método `Show`. Para establecer los valores iniciales en el cuadro de diálogo **Configurar página**, utiliza los argumentos que se enumeran en la siguiente tabla:

Argumento	Descripción
`Arg1`	Head
`Arg2`	Foot
`Arg3`	Left
`Arg4`	Right
`Arg5`	Top
`Arg6`	Bot
`Arg7`	Hdng
`Arg8`	Grid
`Arg9`	H_cntr
`Arg10`	V_cntr
`Arg11`	Orient
`Arg12`	Paper_size
`Arg13`	Scale
`Arg14`	Pg_number
`Arg15`	Pg:order

Argumento	Descripción
`Arg16`	Bw_cells
`Arg17`	Quelity
`Arg18`	Head_margin
`Arg19`	Foot_margin
`Arg20`	Notes
`Arg21`	Draft

Si no especificamos los ajustes iniciales, aparece el cuadro de diálogo **Configurar página** con sus ajustes predeterminados. Pero ¿cómo se deben utilizar los argumentos anteriores? Si deseamos mostrar el cuadro con la orientación de la página en horizontal, utilizamos la siguiente declaración:

```
Application.Dialogs(xlDialogPageSetup).Show Arg11:=2
```

Excel utiliza 1 para la orientación vertical y 2 para la horizontal.

La siguiente declaración muestra el cuadro de diálogo **Configurar página** en el que la opción **Centrar en la página horizontal** está seleccionada en la pestaña **Márgenes** (`Arg9=1`), y la pestaña **Página** tiene la opción **Orientación** establecida en Vertical (`Arg11:=1`):

```
Application.Dialogs(xlDialogPageSetup).Show Arg9:=1, Arg11:=1
```

También podemos establecer los valores iniciales en el cuadro de diálogo **Configurar página** utilizando el objeto `PageSetup` con sus propiedades. Por ejemplo, para establecer la orientación de la página como apaisada, vamos a escribir las siguientes declaraciones en una sola línea en la ventana **Inmediato** y pulsar **Intro**:

```
ActiveSheet.PageSetup.Orientation = 2 :
Application.Dialogs(xlDialogPageSetup).Show
```

Estas dos declaraciones se ejecutan una después de la otra. Los dos puntos indican el final de la primera declaración y el comienzo de la segunda. Es un truco práctico que podemos usar en la ventana **Inmediato** para ejecutar un bloque de código. En la siguientes secciones se describen varios ajustes de página que tal vez deseemos especificar antes de imprimir una hoja de trabajo.

1.1 Configuración de la ficha Disposición de página

Los ajustes de la ficha **Disposición de página** (Diseño de página) se agrupan en cinco áreas principales: Temas, Configuración de página, Ajustar área de impresión, Opciones de hoja y Organizar.

Los ajustes de orientación en el grupo **Configuración de página** indican si la página se imprimirá en vertical u horizontal (propiedad `Orientation`). El ajuste **Tamaño** permite seleccionar uno de los tamaños de papel más comunes. Las opciones del grupo **Ajustar área de impresión** permiten ajustar la impresión según nuestras necesidades. Podemos reducir o ampliar la hoja de trabajo mediante el control **Escala**. Excel puede escalar automáticamente para ajustar la impresión a un número determinado de páginas.

Para hacer esto con VBA	Utilizamos esta declaración
Establecer la disposición horizontal para la Hoja1	`Worksheets("Hoja1").PageSetup.Orientation = xlLandscape`
Escalar la Hoja1 para imprimirla al 200%	`Worksheets("Hoja1").PageSetup.Zoom = 200`

Escalar la Hoja 1 para imprimirla en una sola hoja	`With Worksheets("Hoja1").PageSetup`
	`.FitToPagesTall = 1`
	`.FitToPagesWide = 1`
	`End With`

Establecer el tamaño del papel a “oficio”	`Worksheets("Hoja1").PageSetup.PaperSize = xlPaperLegal`
Averiguar la configuración actual de la calidad de la impresión vertical y horizontal	`Debug.Print "Horizontal Print Quality = " & Worksheets("Hoja1").PageSetup.PrintQuality(1)` `Debug.Print "Vertical Print Quality = " & Worksheets("Hoja1").PageSetup.PrintQuality(2)`

1.2 Configuración de los márgenes

Los ajustes disponibles en la pestaña **Márgenes** del cuadro de diálogo **Configuración de página** que se muestran en la Imagen 1.2, permiten especificar el ancho de los márgenes superior, inferior, izquierdo y derecho (propiedades `TopMargin`, `BottomMargin`, `LeftMargin` y `RightMargin`).

La configuración de encabezado y pie de página nos permite determinar hasta dónde queremos que se imprima el encabezado o el pie de página desde la parte superior o inferior de la página (propiedades `HeaderMargin` y `FooterMargin`). El área de impresión se puede centrar en la página horizontal y verticalmente (propiedades `CenterHorizontally` y `CenterVertically`).

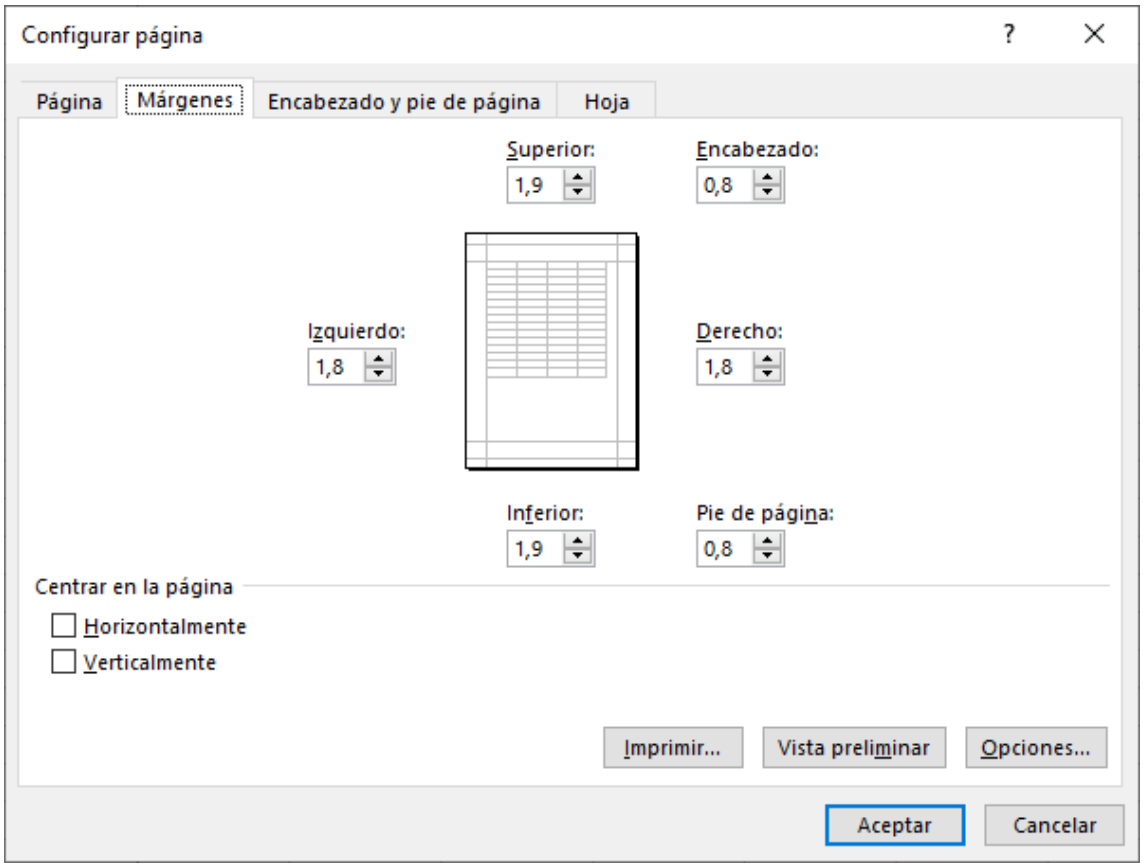


Imagen 1.2 Los ajustes de la pestaña Márgenes del cuadro de diálogo Configurar página determinan los márgenes alrededor del área de impresión y la forma en la que debe centrarse el contenido en la página.

Para hacer esto con VBA	Utilizamos esta declaración

Establecer todos los márgenes de la página a 1,5 pulgadas	`With Worksheets("Hoja1").PageSetup`
	`.LeftMargin = Application.InchesToPoints(1.5)`
	`.RightMargin = Application.InchesToPoints(1.5)`
	`.TopMargin = Application.InchesToPoints(1.5)`
	`.BottomMargin = Application.InchesToPoints(1.5)`
	`End With`

Establecer los márgenes del encabezado y el pie de página en 0,5 pulgadas	`With Worksheets("Hoja1").PageSetup`
	`.HeaderMargin = Application.InchesToPoints(0.5)`
	`.FooterMargin = Application.InchesToPoints(0.5)`
	`End With`

Para hacer esto con VBA	Utilizamos esta declaración

Centrar la Hoja1 horizontalmente	`With Worksheets("Hoja1").PageSetup`
	`.CenterHorizontally = True`
	`.CenterVertically = False`
	`End With`

1.3 Configuración del encabezado y el pie de página

La Imagen 1.3 muestra la configuración de la pestaña **Encabezado y pie de página** que nos permite agregar encabezados y pies de página tanto predeterminados como personalizados. Podemos usar los botones **Encabezado personalizado** y **Pie de página personalizado** para diseñar nuestros propios formatos.

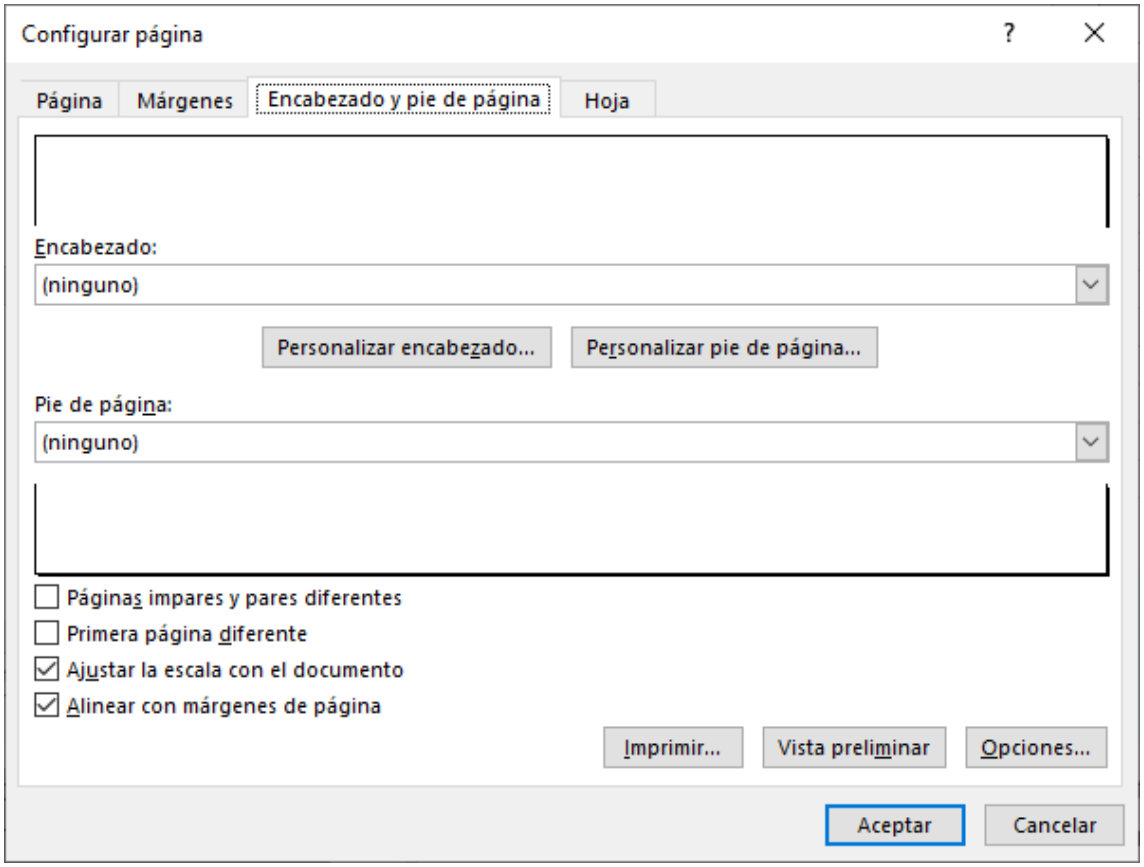


Imagen 1.3 La pestaña Encabezado y pie de página del cuadro de diálogo Configurar página nos permite seleccionar uno de los encabezados o pies de página predeterminados o crear nuestros propios formatos.

El objeto `PageSetup` tiene las siguientes propiedades para configurar y controlar la creación de encabezados y pies de página: `RightHeader`, `LeftHeader`, `RightFooter`, `LeftFooter`, `CenterHeader`, `CenterFooter` `RightHeaderPicture`, `RightFooterPicture`, `LeftHeaderPicture`, `LeftFooterPicture`, `CenterHeaderPicture` y `CenterFooterPicture`.

Los siguientes ajustes están disponibles en la pestaña **Encabezado y pie de página** del cuadro de diálogo **Configurar página**:

- **Páginas impares y pares diferentes**: Se utiliza la propiedad `PageSetup.OddAndEvenPagesHeaderFooter`. Esta propiedad devuelve `True` si el objeto `PageSetup` tiene diferentes encabezados y pies de página para páginas pares e impares.
- **Primera página diferente**: Se utiliza la propiedad `PageSetup.DifferentFirstPageHeaderFooter`. Esta propiedad devuelve `True` si se usa un encabezado o pie de página diferente en la primera página.
- **Ajustar la escala con el documento**: Utiliza la propiedad `PageSetup.ScaleWithDocHeaderFooter`. Esta propiedad devuelve `True` si el encabezado y el pie de página deben usar el mismo tamaño de fuente y escala que la hoja de trabajo.
- **Alinear con márgenes de página**: Se utiliza la propiedad `PageSetUp.AlignMarginsHeaderFooter`. Esta propiedad devuelve `True` si el encabezado y el pie de página tienen los márgenes alineados con el contenido.

Para obtener algo de práctica en el uso de los códigos anteriores, probemos las siguientes declaraciones:

1. Imprime la ruta completa del libro de trabajo en la esquina superior derecha de cada página cuando se imprima la Hoja 1:

```
Worksheets("Hoja1").PageSetup.RightHeader = _
ActiveWorkbook.FullName
```

2. Imprime la fecha, el número de página y el número total de páginas a la izquierda en la parte inferior de cada página cuando se imprima la Hoja1:

```
Worksheets("Hoja1").PageSetup.LeftFooter = "&D Página &P de &N"
```

3. Muestra una marca de agua en la sección central del encabezado de Hoja1:

```
Sub MarcaDeAgua()
    With Worksheets("Hoja1").PageSetup.CenterHeaderPicture
        .Filename = "C:\Archivos Manual VBA\cd.bmp"
        .Height = 75
        .Width = 75
        .Brightness = 0.25
        .ColorType = msoPictureWatermark
        .Contrast = 0.45
    End With
    ' Muestra la imagen en el centro del encabezado.
    ActiveSheet.PageSetup.CenterHeader = "&G"
End Sub
```

Los parámetros disponibles en la pestaña **Hoja** del cuadro de diálogo **Configurar página** (que se muestra en la Imagen 1.4) determinan qué tipos de datos deseamos incluir en la impresión y el orden en que Excel debe imprimir los rangos de datos si la impresión abarca varias páginas.

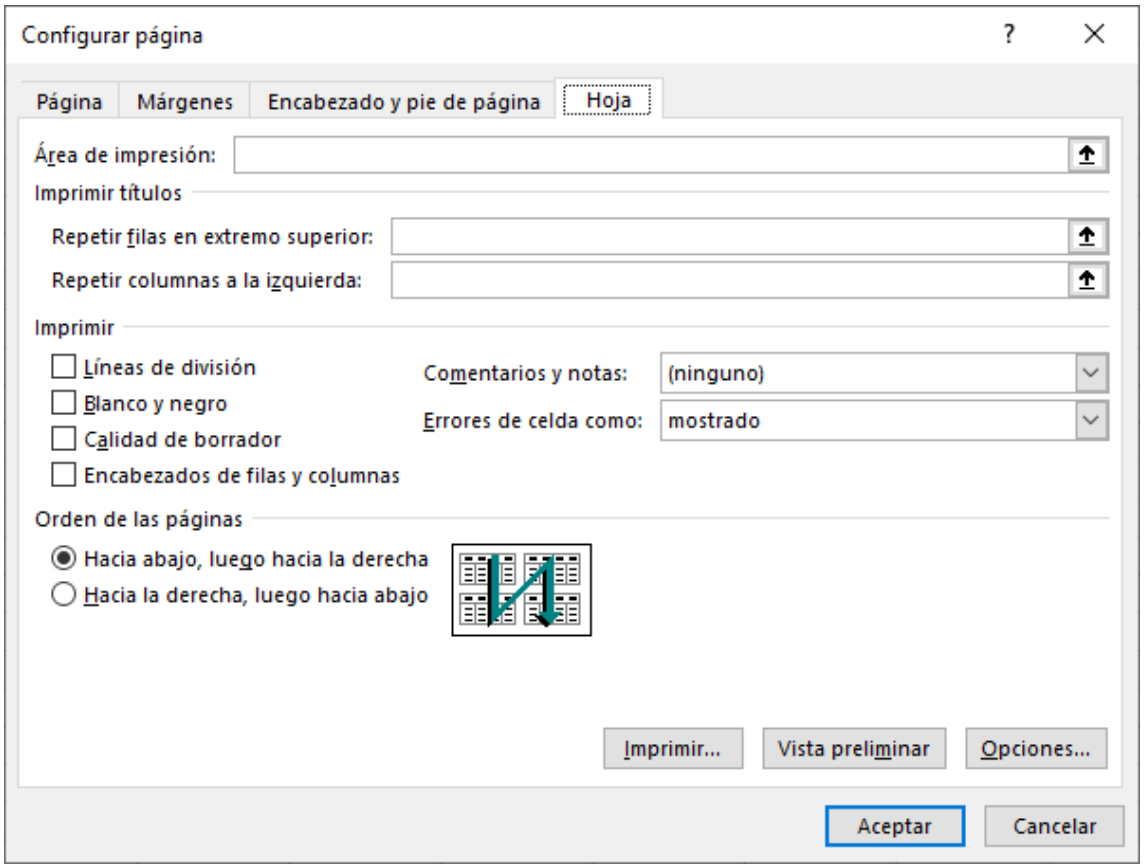


Imagen 1.4 La pestaña Hoja del cuadro Configurar página permite especificar los encabezados y los rangos de datos y ajustar la apariencia de cada página.

El área de impresión es un rango especial que define las celdas que queremos imprimir. Podemos decidir la cantidad de datos de trabajo que queremos imprimir. Excel imprime toda la hoja de cálculo de forma predeterminada. Podemos imprimir solo lo que realmente necesitamos definiendo un área de impresión. Podemos especificar la dirección del rango a imprimir en el ajuste **Área de impresión**. Si no especificamos un área de impresión, Excel imprimirá todos los datos de la hoja de trabajo actual. Si especificamos el rango de celdas que se va a imprimir en la opción **Área de impresión** Excel imprimirá la selección actual de celdas de la hoja de trabajo en lugar del rango de celdas especificado en la opción Área de impresión.

La propiedad `PrintArea` del objeto `PageSetup` se utiliza para devolver o establecer el rango de celdas que se va a imprimir. Si se establece la propiedad `PrintArea` en False o en una cadena vacía (“”), el área de impresión se establecerá en toda la hoja.

El área **Imprimir títulos** de la ficha **Hoja** permite especificar las filas del libro de trabajo que deben imprimirse en la parte superior de cada página o las columnas del libro que se imprimirán en el lado izquierdo de cada página. Estos ajustes son especialmente útiles para imprimir hojas muy grandes. De forma predeterminada, Excel imprime los títulos de la fila y la

columna solo en la primera página, lo que hace muy difícil entender los datos de las páginas siguientes. Para solucionar este problema, podemos decirle a Excel que imprima los títulos de las filas y columnas especificadas en cada página. Especificaremos las filas que deben repetirse en la parte superior de la página (propiedad **PrintTitleRows**) y las columnas que se repetirán en la parte izquierda (propiedad **PrintTitleColumns**). Debemos especificar ambos ajustes para hojas de trabajo extremadamente grandes. Para desactivar los títulos de filas y columnas estableceremos la propiedad correspondiente (**PrintTitleRows** o **PrintTitleColumns**) en False o en una cadena de texto vacía (“”).

La configuración de impresión controla el aspecto de las página impresas. Para imprimir la hoja de trabajo con líneas de cuadrícula, marcamos la casilla **Líneas de división** (propiedad **PrintGridlines**). Para imprimir en blanco y negro, seleccionamos la casilla de verificación correspondiente (propiedad **BlackAndWite**). La impresión con calidad de borrador será más rápida, ya que Excel no imprime la cuadrícula y suprime algunos gráficos en este modo. Para mostrar los encabezados de fila y columna en las páginas impresas, marcamos el cuadro **Encabezados de fila y columna** (propiedad **PrintHeadings**). Excel identificará las filas con números y las columnas con letras. Si la hoja de trabajo contiene comentarios podemos indicar la posición en la página impresa en la que deseamos que se impriman eligiendo una opción del cuadro desplegable **Comentarios** (propiedad **PrintComments**). Si la hoja contiene errores, podemos suprimir la visualización de los valores de error cuando la imprimamos, seleccionando la opción adecuada en el cuadro desplegable **Errores de celda** (propiedad **PrintErrors**). Al utilizar la propiedad **PrintErrors** especificaremos cómo se tienen que mostrar los errores con una de las siguientes constantes:

XlPrintErrorsBlank

XlPrintErrorsDash

XlPrintErrorsDisplayed

xlPrintErrorsNA

Los ajustes en la sección **Orden de las páginas** de la pestaña **Hoja** permiten especificar cómo se deben imprimir y numerar las páginas cuando se imprimen hojas grandes. El orden de impresión por defecto es de arriba abajo. Podemos solicitar que este orden se cambie de izquierda a derecha, lo cual es una forma mejor de imprimir tablas anchas. Utilizaremos la propiedad **OrderProperty** de **PageSetup** para establecer o devolver el orden de impresión.

El orden de las páginas puede ser **xlDownThenOver** o **xlOverThenDown**.

De nuevo, aquí tenemos algunos ejemplos de declaraciones:

1. Establecer el área de impresión en las celdas A2:D10 de la Hoja1:

```
Worksheets("Hoja1").PageSetup.PrintArea = "$A$2:$D$10"
```

2. Especificar la fila 1 como título de fila y las columnas A y B como títulos de columnas:

```
ActiveSheet.PageSetup.PrintTitleRows = _
ActiveSheet.Rows(1).Address
ActiveSheet.PageSetup.PrintTitleColumns =  _
ActiveSheet.Columns("A:B").Address
```

3. Imprimir las líneas de división y los encabezados de columna en la Hoja1:

```
With Worksheets("Hoja1").PageSetup
.PrintHeadings = True
.PrintGridlines = True
End With
```

4. Numerar e imprimir la hoja completa de izquierda a derecha:

```
Worksheets("Sheet1").PageSetup.Order = xlOverThenDown
```

1.4 Recuperación de los valores actuales del cuadro de diálogo Configurar página

Ahora que estamos familiarizados con los muchos ajustes disponibles en el cuadro de diálogo **Configurar página** y conocemos los nombres de las propiedades correspondientes que se pueden utilizar en VBA para escribir el código que configura las hojas para la impresión, es hora de ver un procedimiento completo. El siguiente procedimiento imprime algunos ajustes de configuración de página en la ventana **Inmediato**:

1. Crea un libro nuevo, llámalo Capítulo 20 – Impresión-Correo.xlsm y guárdalo en la carpeta C:\Archivos Manual VBA.
2. En la Hoja1 del archivo, introduce los datos que se muestran en la Imagen 1.5. Los datos de la columna Bonus están calculados multiplicando la columna Meses *3. En la celda D2 introduce =C2*3 y arrastra el resultado hasta la última celda.

	A	B	C	D	E
1	Nombre	Apellido	Meses	Bonus	
2	María	Martín	10	30	
3	José	Jiménez	14	42	
4	Sergio	Santander	34	102	
5	Manuel	Molina	27	81	
6	Pablo	Pinto	18	54	
7	Cristina	Crespo	60	180	
8	Isabel	Ibarra	16	48	
9					
10					
11					

Imagen 1.5 Hoja de ejemplo.

3. Presiona **Alt + F11** para acceder al editor de VBA. Selecciona el proyecto correspondiente al libro que acabas de crear y haz clic en el menú **Insertar – Módulo**.
4. En la ventana **Código** del Módulo1, introduce el siguiente procedimiento:

```
Sub MostrarConfiguracionPagina()
    With ActiveSheet.PageSetup
        Debug.Print "Orientación = "; .Orientation
        Debug.Print "Tamaño papel = "; .PaperSize
```

```
            Debug.Print "Imprimir líneas de división = "; _
            .PrintGridlines
            Debug.Print "Calidad de impresión = "; .PrintQuality(1)
            Debug.Print "Área de impresión = "; .PrintArea
        End With
    End Sub
```

5. Ejecuta el procedimiento.
   Los resultados se imprimirán en la ventana **Inmediato**. Dado que no has cambiado ningún parámetro del cuadro de diálogo **Configurar página**, los valores que aparecen son los predeterminados.
6. Modifica el procedimiento de la siguiente manera:

```
Sub MostrarConfiguracionPagina2()
    With ActiveSheet.PageSetup
        Debug.Print "Orientación = "; .Orientation
        Debug.Print "Tamaño papel = "; .PaperSize
        Debug.Print "Imprimir líneas de división = "; _
        .PrintGridlines
        Debug.Print "Calidad de impresión = "; .PrintQuality(1)
        Cells(1, 1).Select
        .PrintArea = ActiveCell.CurrentRegion.Address
        Debug.Print "Print Area = "; .PrintArea;
        .CenterHeader = Chr(10) & "Informe Bonus"
    End With
    Application.Dialogs(xlDialogPrintPreview).Show
End Sub
```

7. Ejecuta el procedimiento `MostrarConfiguracionPagina2`.
   Ahora, además de escribir los ajustes seleccionados en la ventana **Inmediato**, la hoja de trabajo se formatea y se muestra en la ventana **Vista previa de impresión**.
   Cuando se imprimen hojas de trabajo que contienen un gran número de filas, es una buena idea establecer por separado los títulos de impresión y el área de impresión para que cada página se imprima con los títulos de las columnas.

   El siguiente procedimiento muestra este caso particular. Observa cómo la propiedad `CurrentRegion` de la colección `Range` se utiliza junto con las propiedades `Offset` y `Resize` para redimensionar el área de impresión de forma que no incluya la fila de encabezado (fila 1). El procedimiento establece la fila de encabezado utilizando la propiedad `PrintTitleRows` del objeto `PageSetup`.

```
Sub FormatoHoja()
    Dim curReg As Range
```

```
    Set curReg = ActiveCell.CurrentRegion
    With ActiveSheet.PageSetup
        .PrintTitleRows = "$1:$1"
        Cells(1, 1).Select
        .PrintArea = curReg.Offset(1, 0).Resize _
        (curReg.Rows.Count - 1, curReg.Columns.Count).Address
        Debug.Print "Área de impresión = "; .PrintArea;
        .CenterHeader = Chr(10) & "Informe Bonus"
        .PrintGridlines = True
    End With
    Application.Dialogs(xlDialogPrintPreview).Show
End Sub
```

2 Vista previa de la hoja

Como podemos ver en la Imagen 2.1, la vista **Diseño de página** (Vista > Diseño de página) facilita la visualización de la hoja de trabajo junto con los encabezados y pies de página.

Para agregar un encabezado, simplemente hacemos clic en el área superior de la hoja y escribimos el texto o hacemos clic en los botones apropiados de la ficha contextual **Encabezados y pies de página**. Por ejemplo, para agregar la fecha de hoy, hacemos clic en el botón **Fecha actual**. Para añadir el pie de página, hacemos clic en el botón **Ir al pie de página** en el grupo **Navegación** de la ficha **Encabezado y pie de página**.

C:\Archivos Manual VBA\Capítulo 20 - Impresión-Correo.xlsm
Informe Bonus

Nombre	Apellido	Meses	Bonus
María	Martín	10	30
José	Jiménez	14	42
Sergio	Santander	34	102
Manuel	Molina	27	81
Pablo	Pinto	18	54
Cristina	Crespo	60	180
Isabel	Ibarra	16	48

Imagen 2.1 Para ver la hoja tal como se vería al imprimirse, debemos hacer clic en la ficha Vista > Diseño de página.

Para activar la vista **Diseño de página** se utiliza la siguiente declaración:

```
Application.Dialogs(xlDialogPrintPreview).Show
```

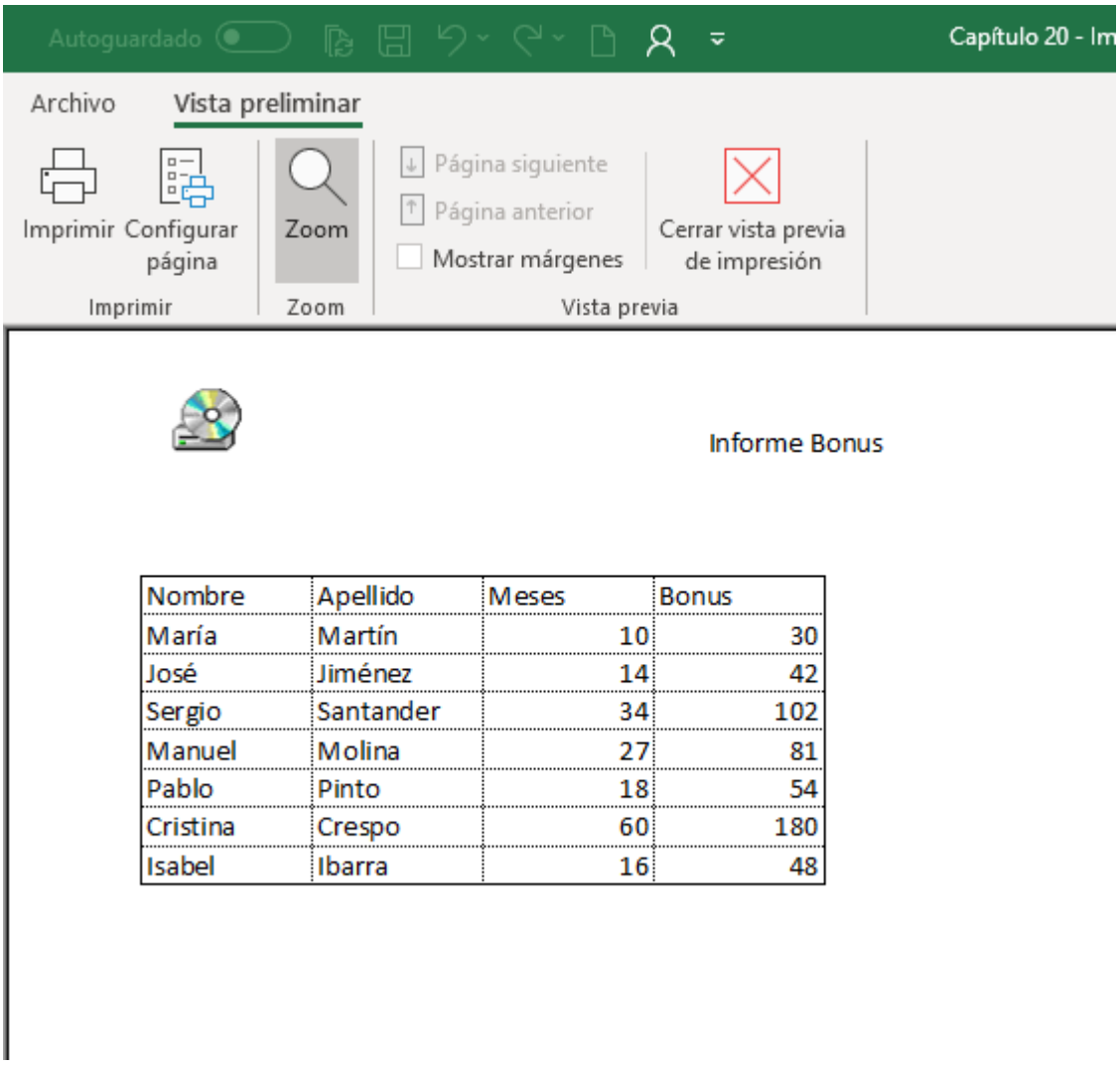
o también:

```
Worksheets("Hoja1").PrintPreview
```

Las declaraciones anteriores no funcionarán si la hoja no contiene datos.

Como podemos ver en la Imagen 2.2, hay botones en la parte superior de la ventana **Vista preliminar** que permiten que nos movamos entre las diferentes páginas de la impresión, hacer ajustes en la configuración de la página y los márgenes, ver más cerca los datos o cualquier parte de la impresión (botón de zoom) e imprimir la hoja. Si la hoja tiene más de una página, podemos mostrarlas en la **Vista preliminar** haciendo clic en los botones **Siguiente** o **Anterior** o utilizando el teclado (flecha hacia abajo, flecha hacia arriba y teclas Inicio y Fin).

Podemos usar el botón **Zoom** de la ventana **Vista preliminar** para cambiar a otra vista más amplia y podemos ajustar los márgenes de la página y el ancho de las columnas visualmente con el ratón (para ello debemos seleccionar la casilla de verificación **Mostrar márgenes** en el grupo **Vista previa**). También podemos imprimir utilizando el botón **Imprimir** y acceder fácilmente al cuadro de diálogo **Configurar página** para realizar cambios en el diseño de impresión.



Nombre	Apellido	Meses	Bonus
María	Martín	10	30
José	Jiménez	14	42
Sergio	Santander	34	102
Manuel	Molina	27	81
Pablo	Pinto	18	54
Cristina	Crespo	60	180
Isabel	Ibarra	16	48

Imagen 2.2 La ventana Vista preliminar muestra una versión reducida de las páginas de la hoja. Observemos que se trata de una ventana totalmente diferente, que se abre mediante el comando Archivo > Imprimir.

A veces podemos querer evitar que los usuarios modifiquen la configuración de la página o que impriman desde la ventana **Vista preliminar**. Esto se puede hacer de forma automática. Podemos desactivar los botones **Mostrar márgenes** y **Configurar página** en la ventana **Vista preliminar** de una de las siguiente formas:

```
Application.Dialogs(xlDialogPrintPreview).Show False
```

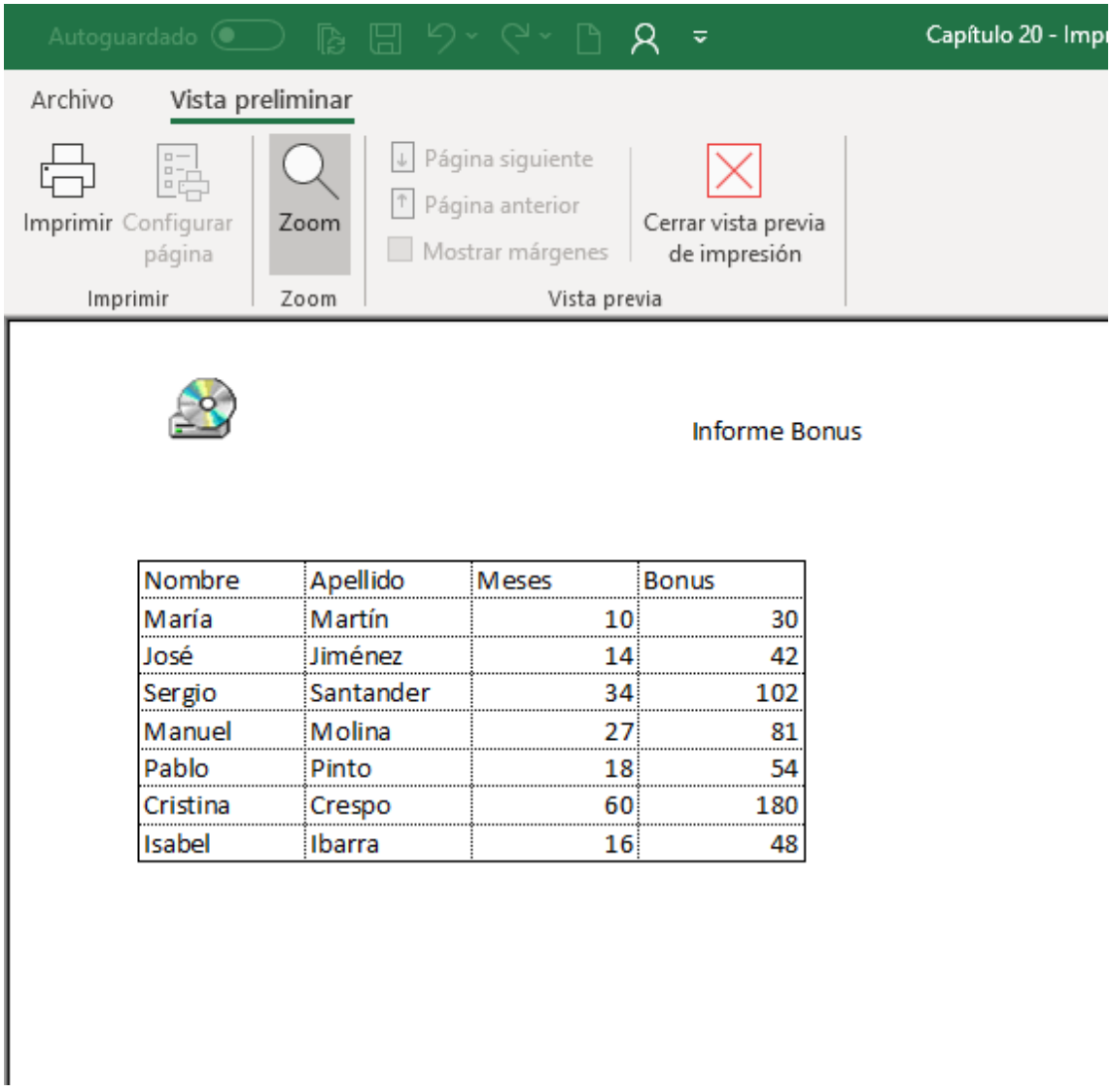
o:

```
Worksheets("Hoja1").PrintPreview EnableChanges:=False
```

o también:

```
Worksheets("Hoja1").PrintPreview False
```

En la Imagen 2.3 se muestra la ventana **Vista preliminar** con la opción **Mostrar márgenes** y el botón **Configurar página** desactivados.



Nombre	Apellido	Meses	Bonus
María	Martín	10	30
José	Jiménez	14	42
Sergio	Santander	34	102
Manuel	Molina	27	81
Pablo	Pinto	18	54
Cristina	Crespo	60	180
Isabel	Ibarra	16	48

Imagen 2.3 La ventana Vista preliminar con las opciones Configurar página y Mostrar márgenes desactivadas.

3 Cambiar la impresora activa

Antes de imprimir, tal vez queramos mostrar la lista de impresoras disponibles para que los usuarios las seleccionen. El cuadro de diálogo **Configurar impresora** se muestra en la Imagen 3.1. Este cuadro se puede mostrar con la siguiente declaración:

```
Application.Dialogs(xlDialogPrinterSetup).Show
```

Para conocer el nombre de la impresora activa, utilizamos la propiedad `ActivePrinter` del objeto `Application`.

```
MsgBox Application.ActivePrinter
```

Para cambiar la impresora activa usamos la siguiente declaración, reemplazando el nombre y el puerto de la impresora por alguno disponible en nuestro equipo:

```
Application.ActivePrinter = "Brother HL-5370DW series Printer
en Ne04:"
```

Podemos decirle a Excel que establezca una impresora predeterminada al abrir el libro de trabajo, introduciendo la declaración anterior dentro del procedimiento de eventos `Workbook_Open`.

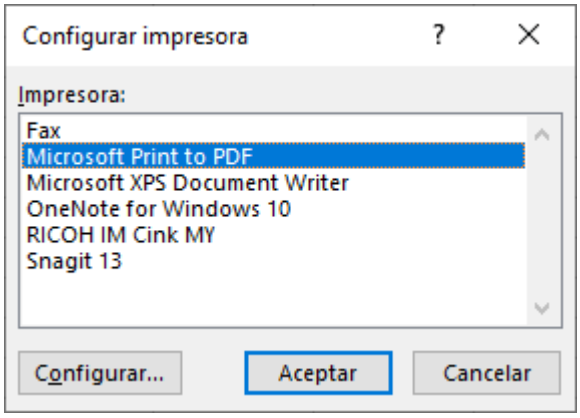


Imagen 3.1 El cuadro de diálogo Configurar impresora.

1. En el libro Capítulo 20 – Impresión-Correo.xlsm, activa el editor de VBA y Haz doble clic en el módulo `ThisWorkbook`.
2. En la ventana **Código**, introduce el siguiente procedimiento, que reemplazará la impresora actual por una impresora en concreto:

```
Private Sub Workbook_Open()
    Application.ActivePrinter = "Microsoft Print to PDF en Ne03"
    MsgBox Application.ActivePrinter
End Sub
```

Atención

La impresora debe estar conectada.

3. Guarda el libro y ciérralo.
4. Abre el archivo de nuevo. Excel ejecutará el procedimiento `Workbook_Open` y establecerá la impresora activa.

4 Imprimir una hoja con VBA

Antes de imprimir una hoja puede que deseemos configurar las opciones de impresión como los rangos a imprimir o el número de copias. Esto se hace fácilmente configurando las opciones adecuadas en el cuadro de diálogo **Imprimir** (ver Imagen 4.1).

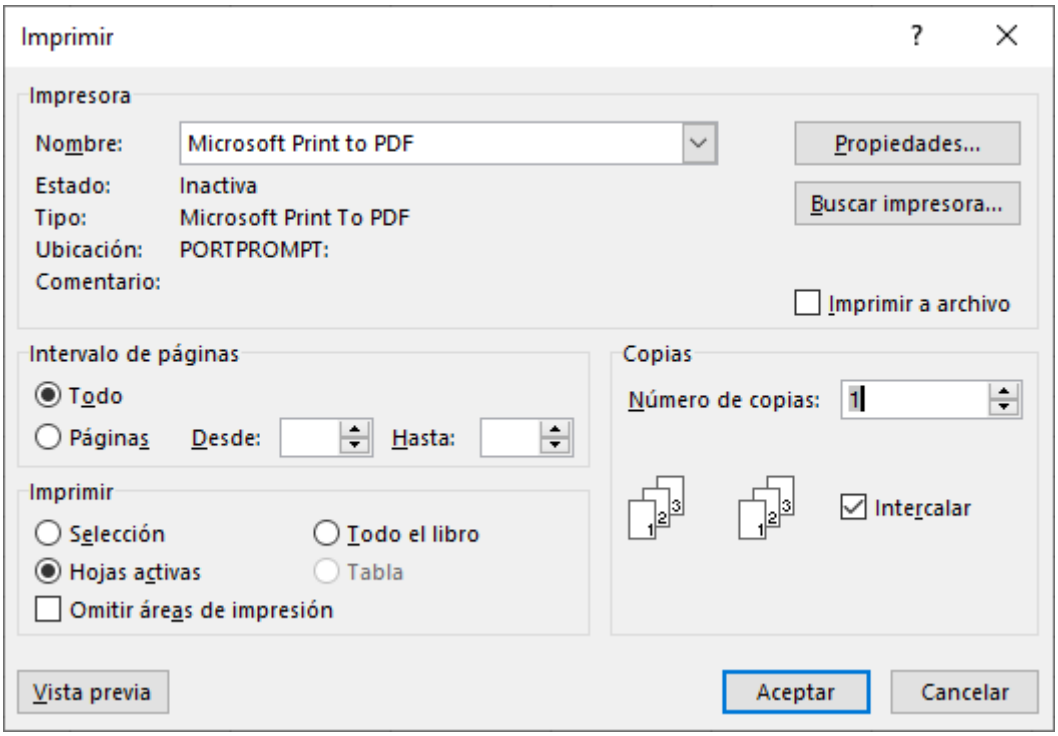


Imagen 4.1 El cuadro de diálogo Imprimir permite especificar varias opciones de impresión.

Para mostrar este cuadro de diálogo, podemos usar la siguiente declaración:

```
Application.Dialogs(xlDialogPrint).Show
```

Podemos incluir algunos argumentos después del método `Show`. Para establecer los valores iniciales, se utilizan los argumentos de la siguiente tabla:

Argumento	Descripción
Arg1	Range_num
Arg2	From
Arg3	To
Arg4	Copies
Arg5	Draft

Argumento	Descripción
Arg6	Preview
Arg7	Print_what
Arg8	Color
Arg9	Feed
Arg10	Quality
Arg11	Y_resolution
Arg12	Selection
Arg13	Printer_text
Arg14	Print_to_file
Arg15	Collate

Por ejemplo, la siguiente declaración imprimirá las páginas 1 y 2 de la hoja de trabajo activa (suponiendo que la hoja activa conste de dos o más páginas):

```
Application.Dialogs(xlDialogPrint).Show Arg1:=2, _
Arg2:=1, Arg3:=2
```

El primer argumento selecciona el botón de opción **Páginas** en el área **Rango de impresión** del cuadro de diálogo **Imprimir**. Para seleccionar la opción **Todo** debemos configurar el `Arg1` en `1`.

El segundo y tercer argumento especifican las páginas que se quieren imprimir (debemos introducir el número de la página inicial y el de la última página a imprimir).

Para enviar esta configuración directamente a la impresora (sin pasar por el cuadro de diálogo **Impresión** de la ficha **Archivo**), utilizamos la siguiente declaración:

```
ActiveSheet.PrintOut
```

El método `PrintOut` puede tener los siguientes argumentos:

Argumento	Descripción
`From`	El número de la primera página a imprimir. Si se omite, la impresión comenzará en la página 1.
`To`	El número de la última página a imprimir. Si se omite, la impresión finalizará en la última página.

Argumento	Descripción
`Copies`	El número de copias a imprimir. Si se omite, se imprimirá una copia.
`Preview`	Si se establece en True, Excel mostrará la vista previa de impresión. En caso de omitirse o establecerse en False, la impresión comenzará inmediatamente.
`ActivePrinter`	Establece el nombre de la impresora activa.
`PrintToFile`	Si se establece en `True`, la hoja se enviará a un archivo.
`Collate`	Si se establece en `True`, se intercalarán las copias.
`PrToFileName`	Especifica el nombre del archivo donde se hará la impresión en caso de que el argumento `PrintToFile` sea `True`.

5 Eventos relacionados con la impresión

Antes de que se imprima el libro de trabajo (y antes de que aparezca el cuadro de diálogo **Imprimir**), Excel activa el evento `Workbook_BeforePrint`. Este evento se puede utilizar para realizar ciertas tareas de formateo o cálculo antes de la impresión o para impedir la impresión y la vista previa de la impresión por completo cuando se solicitan estas funciones. El código para el procedimiento del evento `Workbook_BeforePrint` debe colocarse en la ventana de código de `ThisWorkbook` (Imagen 5.1).

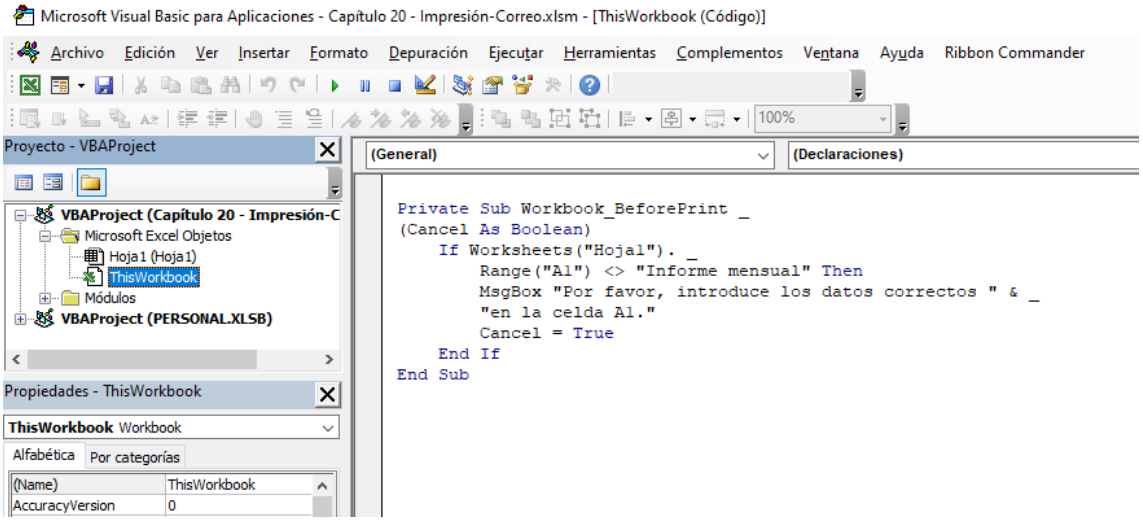


Imagen 5.1 Escribiendo el procedimiento Workbook_BeforePrint en el código de ThisWorkbook.

Se puede acceder a la ventana **Código** de `ThisWorkbook` haciendo doble clic en el objeto correspondiente en el **Explorador de proyectos**. A continuación, en la parte superior de la ventana, seleccionamos `Workbook` en la lista desplegable **Objeto**. La lista de la derecha mostrará los nombres de los eventos a los que puede responder el objeto `Workbook`. Seleccionemos el nombre del evento `BeforePrint`, Excel colocará la estructura de este procedimiento en la ventana **Código**. Escribimos el código VBA entre las dos líneas. La próxima vez que imprimamos, Excel ejecutará primero este código y luego procederá a imprimir la hoja.

El código del evento `Workbook_BeforePrint` se activa tanto si hemos solicitado la impresión mediante las herramientas de Excel como si hemos escrito un procedimiento VBA para controlar la impresión.

Las siguientes tareas se pueden realizar desde el código VBA colocado en el procedimiento `Workbook_BeforePrint`:

- Desactivar la impresión y la vista previa de impresión.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    If Weekday(Date, vbSunday) = 7 Then Cancel = True
End Sub
```

  Cuando establecemos el argumento `Cancel` en `True`, la hoja no se imprime cuando finaliza el procedimiento. El procedimiento anterior no permite imprimir los sábados (el séptimo día de la semana). La función `Weekday` especifica que el domingo es el primer día de la semana.

- Colocar el nombre completo del libro en el pie de página.
- Cambiar el formato de la hoja de cálculo antes de la impresión.
- Validar los datos al imprimir.

```
Private Sub Workbook_BeforePrint _
(Cancel As Boolean)
    If Worksheets("Hoja1"). _
        Range("A1") <> "Informe mensual" Then
        MsgBox "Por favor, introduce los datos correctos " & _
        "en la celda A1."
        Cancel = True
    End If
End Sub
```

- Calcular todas las hojas en el libro activo.

```
Private Sub Workbook_BeforePrint(Cancel As Boolean)
    Dim sh As Variant

    For Each sh In Worksheets
        sh.Calculate
```

```
    Next
End Sub
```

Ya hemos trabajado con los eventos a nivel de aplicación en el capítulo 15. El siguiente ejemplo muestra cómo hacer que Excel imprima automáticamente en el pie de la página la ruta completa del archivo en todos los libros abiertos y nuevos.

Comenzarás el ejercicio creando el libro Personal.xlsb. Las macros guardadas en este archivo están disponibles siempre que trabajemos con Excel. El libro de macros personales (Personal.xlsb) se almacena en la carpeta XLStart. Si este libro no existe todavía (es habitual si no hemos intentado guardar nada en él), Excel lo crea cuando grabamos una macro y seleccionamos la opción de almacenarlo en el libro de macros personal.

1. En la ventana de Excel, selecciona la ficha **Programador (Desarrollador) > Grabar macro.**
2. En el cuadro de diálogo **Grabar macro**, selecciona **Libro de macros personal** en la lista desplegable **Guardar macro en**.
   El libro de macros personales se carga automáticamente cada vez que inicias Excel.
3. Haz clic en **Aceptar** para iniciar la grabación.
   En este ejercicio no grabarás nada.
4. Haz clic en el botón **Detener la grabación** para detener la grabadora de macros.
5. Activa el editor de VBA.
6. En el **Explorador de proyectos**, selecciona VBAProject (PERSONAL.XLSB).
   Con el proyecto seleccionado, y utilizando la ventana **Propiedades**, cambia el nombre del proyecto.
7. Haz doble clic de nuevo en el **Libro de macros personal**.
8. En la carpeta **Módulos**, haz clic con el botón derecho del ratón en el Módulo1 y selecciona **Quitar Módulo1**.
9. Ahora haz clic en el menú **Insertar – Módulo de clase**.
   Excel inserta un módulo de clase llamado Clase1.
10. En la ventana **Propiedades** cambia el nombre de Clase1 a clsFooter.
11. Introduce la siguiente declaración y el procedimiento de eventos en la ventana **Código** de `clsFooter`:

```
Public WithEvents objApp As Application
Private Sub objApp_WorkbookBeforePrint(ByVal Wb As Workbook, _
Cancel As Boolean)
    With Wb.ActiveSheet
        .PageSetup.RightFooter = Wb.FullName
    End With
End Sub
```

Recuerda en el Capítulo 15 que la palabra clave `WithEvents` se utiliza en un módulo de clase para declarar una variable de objeto que apunta al objeto `Application`. En este procedimiento, `objApp` es el nombre de la variable del objeto `Application`. La declaración `Public` antes de la palabra `WithEvents` permite que todos los módulos del proyecto VBA accedan a la variable `objApp`.

Una vez declarada la variable objeto, puedes seleccionar `ObjApp` en la lista desplegable **Objeto** de la ventana **Código** del módulo `clsFooter` y seleccionar el evento `Workbook_BeforePrint` en la lista desplegable de la derecha. Cuando se comienzan a escribir los procedimientos de eventos mediante esta técnica (seleccionando las opciones de las listas desplegables **Objeto** y **Procedimiento**), Excel siempre inserta la estructura del procedimiento (el inicio y el final) en la ventana **Código**. De esta forma puedes estar seguro de que siempre estarás trabajando con el nombre de procedimiento correcto. Todo lo que queda por hacer es escribir algún código que especifique las tareas a realizar. El procedimiento anterior simplemente le dice a Excel que coloque la ruta completa y el nombre del archivo en el área derecha del pie de página de la hoja activa.

Después de escribir el procedimiento de eventos en el módulo de clase, necesitamos escribir algún código en el módulo de clase de `ThisWorkbook`.

12. En el **Explorador de proyectos**, haz doble clic en el objeto `ThisWorkbook` del proyecto Personal.xlsb.
13. Escribe el siguiente código de declaración de eventos en la ventana **Código** de `ThisWorkbook`:

```
Dim clsFullPath As New clsFooter
Private Sub Workbook_Open()
    Set clsFullPath.objApp = Application
End Sub
```

La primera línea declara una variable llamada `clsFullPath`, que apunta al objeto (`objApp`) en el módulo de clase `clsFooter`. La palabra `New` indica que se debe crear una nueva instancia del objeto la primera vez que se hace referencia al mismo. Esto se hace en el procedimiento de evento `Workbook_Open` usando la palabra clave `Set`. Esta instrucción conecta el objeto ubicado en el módulo de clase `clsFooter` con la variable de objeto `objApp` que representa el objeto `Application`.

El código colocado en el procedimiento de evento `Workbook_Open` se ejecuta siempre que se abe un libro. Por lo tanto, cuando se abre cualquier libro (existente o nuevo), Excel sabrá que debe atender los eventos de `Application`; en particular debe hacer un seguimiento de los eventos para el objeto `objApp` y ejecutar el código de procedimiento de eventos `WorkbookBeforePrint` cuando se hace una solicitud de impresión o de vista previa de impresión a través de la interfaz de usuario de Excel o mediante el código VBA que se coloca dentro de un procedimiento de impresión personalizado.

Antes de que Excel pueda realizar las tareas programadas, debes guardar los cambios en el archivo Personal.xlsb y salir de Excel.

14. Haz clic en **Depuración – Compilar VBAProject** para asegurarte de que Excel podrá ejecutar el código que acabas de añadir al libro Personal.xlsb. Si Excel encuentra algún error, resaltará la declaración que necesitas examinar.

Haz las correcciones apropiadas y repite la operación. Cuando no haya errores en el proyecto Personal.xlsb, el comando **Compilar VBAProject** estará en gris.

15. Cierra el libro Capítulo 20 – Impresión-Correo.xlsm y cualquier otro libro que tengas abierto.
16. Sal de Microsoft Excel. Cuando Excel te pregunte si quieres guardar los cambios al archivo Personal.xlsb, haz clic en **Sí**.
17. Abre Excel de nuevo creando un libro. A continuación, escribe algún texto en cualquier hoja del libro. Guarda el libro como Prueba pie.xlsx.
18. Haz clic en **Archivo > Imprimir**, selecciona la impresora que desees y haz clic en el botón **Imprimir**. Al hacer clic en el botón, Excel ejecutará el evento `Workbook_BeforePrint` y la impresión resultante debe incluir la ruta del archivo en el pie de página derecho.
19. Cierra el libro.

6 Envío de correos desde Excel

Podemos compartir nuestros libros de Excel con otras personas enviándoles un correo electrónico.

Para enviar un correo desde Excel necesitamos tener instalada una de estas aplicaciones:

- Microsoft Outlook.
- Microsoft Live Mail.
- Cliente de Microsoft Exchange.
- Otro programa compatible con MAPI (Messaging Application Programming Interface).

Un libro de Excel puede enviarse como archivo adjunto PDF/XPS o como fax; o puede enviarse un enlace al archivo almacenado en una ubicación compartida. Cuando se envía un correo con un archivo adjunto, el correo ocupa más espacio pero el destinatario puede abrir y editar el archivo. Para compartir un archivo por correo electrónico hacemos clic en la ficha **Archivo > Compartir**, como se muestra en la Imagen 6.1.

Cuando se selecciona alguna de las opciones de **Adjuntar una copia**, Excel muestra un correo electrónico (ver Imagen 6.2).

Atención

Puede que aparezca un mensaje solicitando crear un perfil de Microsoft Outlook. En este caso seguiremos las instrucciones para crearlo.

Imagen 6.1 Las opciones de Compartir permiten enviar correos electrónicos desde Excel.

Es posible llamar a la ventana del correo electrónico mediante programación, con la siguiente declaración:

```
Application.Dialogs(xlDialogSendMail).Show
```

También podemos incluir algunos argumentos, que se muestran en la siguiente tabla:

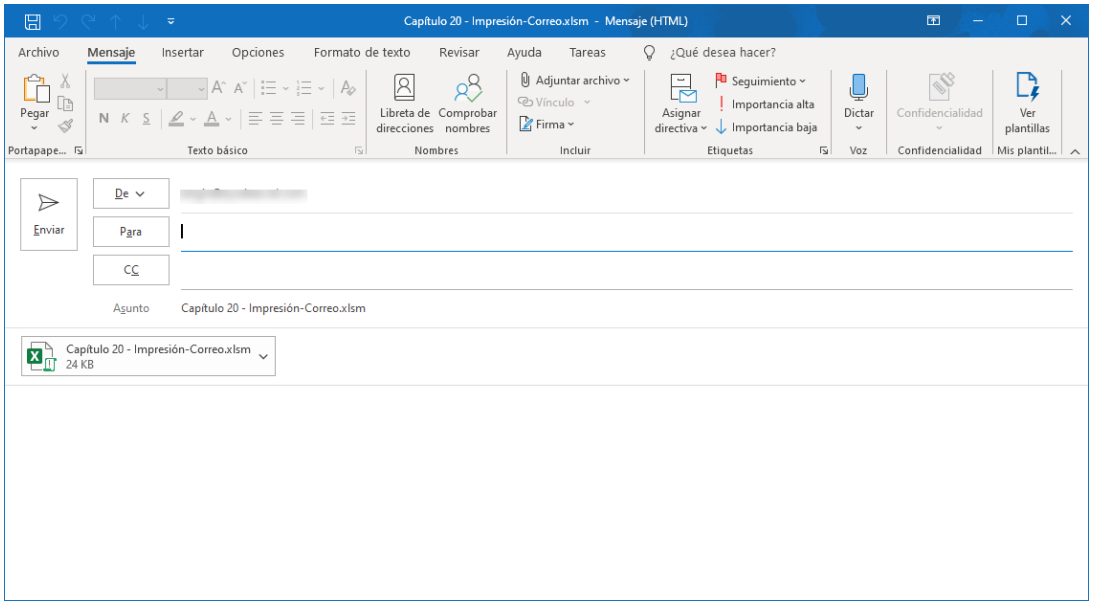


Imagen 6.2 Enviando un archivo adjunto por correo electrónico.

Argumento	Descripción
Arg1	Destinatarios
Arg2	Asunto
Arg3	Acuse de recibo

Por ejemplo, la siguiente declaración muestra la ventana de envío de correo electrónico con la dirección de correo del destinatario y el campo **Asunto** ya relleno:

```
Application.Dialogs(xlDialogSendMail).Show _
Arg1:="SendToName@SendToProvider.com", _
Arg2:="New workbook file"
```

Para comprobar que esta declaración funciona, podemos escribirla (en una sola línea) en la ventana **Inmediato** y presionar **Intro**. También podemos colocarla en un procedimiento.

6.1 Envío de correo usando el método SendMail

Antes de comenzar a enviar correos electrónicos desde procedimientos, es una buena idea conocer qué sistema de correo electrónico tenemos instalado en nuestro ordenador. Podemos hacerlo con la propiedad `MailSystem` del objeto `Application`. Esta es una propiedad de solo lectura que utiliza las constantes `xlMAPI`, `xlPowerTalk` y `xlNoMailSystem` para determinar el sistema de correo instalado. MAPI se utiliza para la interacción con los sistemas de correo. PowerTalk es un sistema de correo de Macintosh.

El siguiente procedimiento muestra cómo utilizar la propiedad `MailSystem`:

```
Sub SistemaCorreo()
    Select Case Application.MailSystem
    Case xlMAPI
        MsgBox "Microsoft Mail instalado."
    Case xlNoMailSystem
        MsgBox "No hay sistemas de correo instalados."
    Case xlPowerTalk
        MsgBox "PowerTalk instalado."
    End Select
End Sub
```

La forma más fácil de enviar un correo electrónico desde Excel es usando el método `SendMail` del objeto `Application`. Este método permite especificar la dirección de correo electrónico del destinatario, el asunto y si deseamos acuse de recibo. Vamos a crear un correo electrónico y a enviárnoslo a nosotros mismos:

1. Abre el libro Capítulo 20 – Impresión-Correo.xlsm e inserta un módulo nuevo.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub EnviarCorreo()
    Dim strEAddress As String

    On Error GoTo GestionErrores
    strEAddress = InputBox("Introduce la dirección de correo", _
    "Correo del destinatario ")
    If IsNull(Application.MailSession) Then
        Application.MailLogon
    End If
    ActiveWorkbook.SendMail Recipients:=strEAddress, _
    Subject:="Correo de prueba"
    Application.MailLogoff
    Exit Sub
GestionErrores:
    MsgBox "Ha ocurrido un error al enviar el correo."
End Sub
```

Si Microsoft Mail no se está ejecutando todavía, debes utilizar la propiedad `MailSession` del objeto `Application` para establecer una sesión de correo en Excel antes de enviar los correos electrónicos. La propiedad `MailSession` devuelve el número de sesión de correo MAPI en forma de cadena hexadecimal, o `Null` si la sesión de correo no se ha establecido todavía. La propiedad `MailSession` no se utiliza en los sistemas de correo `PowerTalk`. Para establecer una sesión de correo, utiliza el método `MailLogon` del objeto `Application`. Y para cerrar la sesión MAPI establecida por Excel, utiliza el método `MailLogoff`.

3. Ejecuta el procedimiento anterior para enviar por correo electrónico el libro activo. Cuando escribas tu dirección de correo y presiones **Aceptar**, se mostrará el mensaje que aparece en la Imagen 6.3. Haz clic en **Permitir** para permitir el envío de correos.

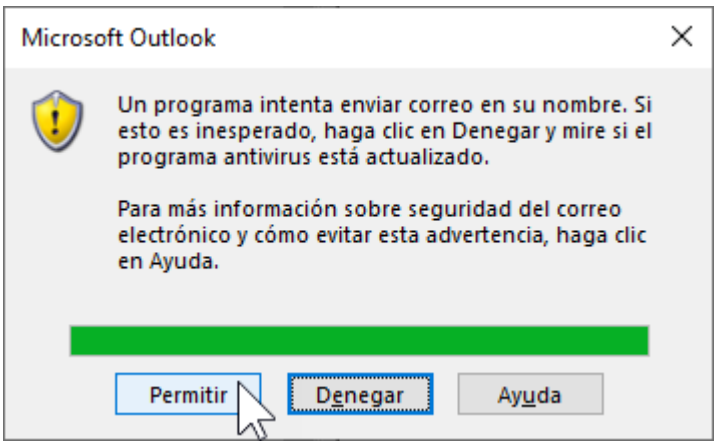


Imagen 6.3 Mensaje de advertencia recibido al intentar enviar un correo electrónico desde Excel.

4. Abre tu programa de correo electrónico y comprueba la bandeja de entrada.

Cuando el destinatario recibe un correo electrónico con un libro de Excel adjunto, necesitará Excel para abrirlo.

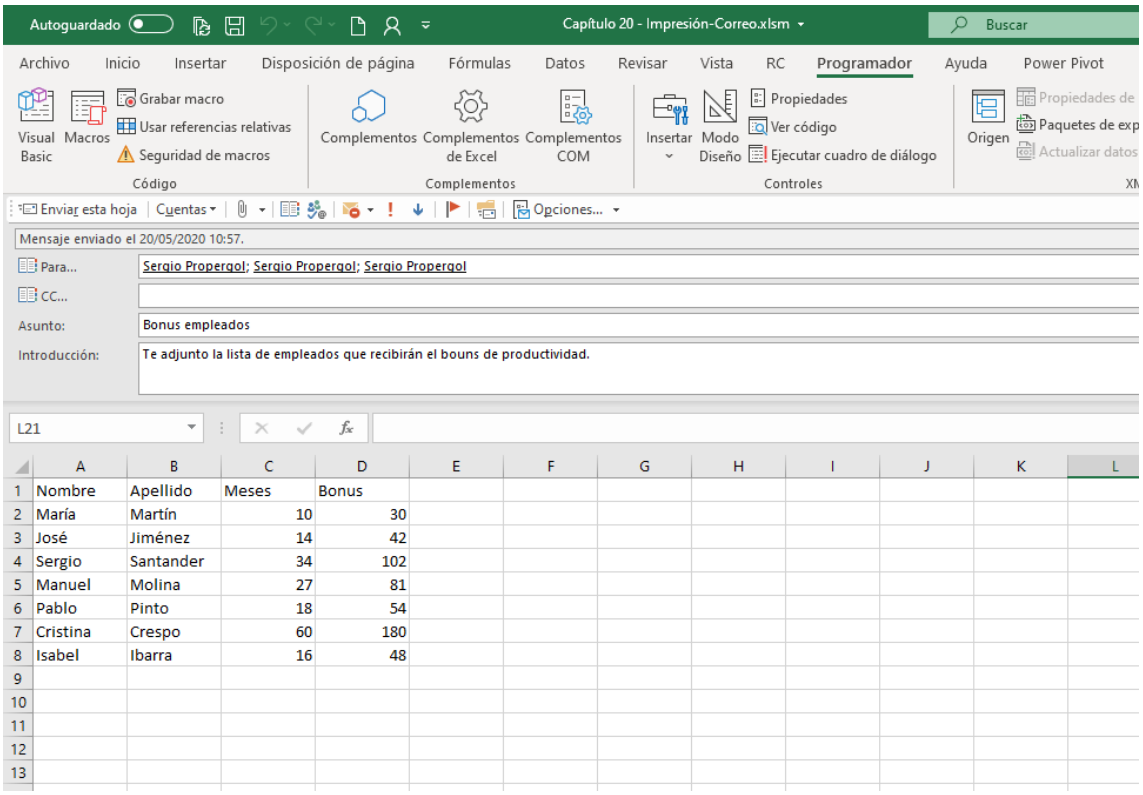
6.2 Envío de correo usando el objeto MsoEnvelope

También podemos enviar correos electrónicos directamente desde Excel u otras aplicaciones de Microsoft Office a través del objeto **MsoEnvelope**, que se incluye en la biblioteca de objetos de Microsoft Office 16.0. Para devolver un objeto **MsoEnvelope**, usamos la propiedad **MailEnvelope** del objeto **Worksheet**. También es necesario configurar una referencia al objeto **MailItem** en la biblioteca de objetos de Microsoft Outlook 16.0 para acceder a sus propiedades y métodos que dan formato al mensaje de correo. El siguiente procedimiento muestra el envío de correo utilizando el objeto **MsoEnvelope**.

En vez de adjuntar todo el libro de Excel, incrustaremos los datos que se muestran en la Hoja1, creados en este capítulo (ver Imagen 6.4).

1. Activa la hoja 1 del libro Capítulo 20 – Impresión-Correo.xlsm.
2. Presiona **Alt + F11** para activar el editor de VBA.
3. Configura una referencia a las bibliotecas de objetos Microsoft Outlook 16.0 y Microsoft Office 16.0 desde el cuadro de diálogo **Herramientas – Referencias**.
4. En el editor de VBA inserta un módulo nuevo.
5. Introduce el siguiente procedimiento en la ventana **Código** del módulo:

```
Sub MetodoMsoMail(ByVal strRecipient As String)
    ' Utiliza la propiedad MailEnvelope de la hoja
    ' para devolver un objeto msoEnvelope
    ActiveWorkbook.EnvelopeVisible = True
    With ActiveSheet.MailEnvelope
        .Introduction = "Te adjunto la lista de empleados " & _
        "que recibirán el bouns de productividad."
        With .Item
            ' Se asegura de que el formato del correo es HTML
            .BodyFormat = olFormatHTML
            ' Agrega el destinatario
            .Recipients.Add strRecipient
            ' Agrega el asunto
            .Subject = "Bonus empleados"
            ' Send Mail
            .Send
        End With
    End With
End Sub
```



	A	B	C	D
1	Nombre	Apellido	Meses	Bonus
2	María	Martín	10	30
3	José	Jiménez	14	42
4	Sergio	Santander	34	102
5	Manuel	Molina	27	81
6	Pablo	Pinto	18	54
7	Cristina	Crespo	60	180
8	Isabel	Ibarra	16	48

Imagen 6.4 Un correo con una hoja incrustada generado por un procedimiento.

6.3 Envío de correos masivos desde Outlook

A veces puede ser necesario enviar mensajes de correo personalizado a las personas cuyas direcciones de correo electrónico y otra información personal se encuentran en la hoja de cálculo. El siguiente procedimiento muestra cómo procesar este tipo de solicitud de Excel desde los objetos, propiedades y métodos que proporciona la biblioteca de objetos de Microsoft Outlook 16.0

1. Elabora una hoja de cálculo como la de la Imagen 6.5.

	A	B	C	D
1	Nombre	Tipo gasto	Importe	Correo
2	Manuel Martín	Transporte	54	manuelmartin@micorreo.com
3	Silvia Segura	Mat. Oficina	36	silviasegura@micorreo.com
4	Iker Iranzo	Transporte	12	ikermartin@micorreo.com
5	Manuel Mendoza	Alojamiento	99	manuelmartin@micorreo.com
6	Ricardo Rojas	Alojamiento	114	ricardomartin@micorreo.com
7				
8				
9				

Imagen 6.5 Hoja con datos de ejemplo para el envío masivo de correos.

2. Activa el editor de VBA y selecciona **Herramientas – Referencias**. Asegúrate de que la biblioteca Microsoft Outlook 16.0 se encuentra activa. En caso de que no sea así, actívala. A continuación, presiona en **Aceptar** para cerrar el cuadro.

3. Inserta un módulo nuevo al libro.
4. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub CorreoMasivo(EmailCol, BeginRow, EndRow, SubjCol, _
NameCol, AmountCol)
    Dim objOut As Outlook.Application
    Dim objMail As Outlook.MailItem
    Dim strEmail As String
    Dim strSubject As String
    Dim strBody As String
    Dim r As Integer

    On Error Resume Next
    Application.DisplayAlerts = False
    Set objOut = New Outlook.Application
    For r = BeginRow To EndRow
        Set objMail = objOut.CreateItem(olMailItem)
        strEmail = Cells(r, EmailCol)
        strSubject = "Reembolso de " & Cells(r, SubjCol)
        strBody = "Estimado " & Cells(r, NameCol).Text & ":" & _
        vbCrLf & vbCrLf
        strBody = strBody & _
        "Se ha aprobado tu nota de gastos de " & _
        LCase(strSubject)
        strBody = strBody & ", por la cantidad de " & Cells(r, _
        AmountCol).Text & "."
        strBody = strBody & vbCrLf & _
        "En aproximadamente 3 días "
        strBody = strBody & " la cantidad aparecerá en tu
cuenta."
        strBody = strBody & vbCrLf & vbCrLf & " Atención al
empleado"
        With objMail
            .To = strEmail
            .Body = strBody
            .Subject = strSubject
            .Send
        End With
    Next
    Set objOut = Nothing
    Application.DisplayAlerts = True
End Sub
```

El procedimiento anterior requiere los siguientes parámetros: **EmailCol**, **BeginRow**, **EndRow**, **SubjCol**, **NameCol** y **AmountCol**. El parámetro **EmailCol** es el número de la columna de la hoja donde se ha introducido la dirección de correo. En este ejemplo, es la cuarta columna. Los parámetros **BeginRow** y **EndRow** especifican la primera y la última fila del rango de datos. En este ejemplo, la primera fila que queremos procesar es la 2 y la última la 6. **SubjCol** es el número de columna donde se introduce el asunto del correo. En este ejemplo, es la segunda columna. **NameCol** contiene el nombre del empleado (primera columna). **AmountCol** es el número de columna donde se ha introducido la cantidad del gasto. En este ejemplo es la tercera columna.

La declaración **Application.DisplayAlerts = False** hará que Excel deje de mostrar mensajes de alerta; sin embargo, esto no impedirá que aparezcan los mensajes en Outlook. Antes de especificar los detalles del correo debemos establecer una referencia a la aplicación de Outlook con la siguiente declaración:

```
Set objOut = New Outlook.Application
```

A continuación, necesitamos obtener datos de cada persona a la que hay que enviar el correo. El procedimiento utiliza el bucle **For … Next** para recorrer los datos de la hoja comenzando en la fila 5 y finalizando en la 6. Cada vuelta del bucle se establece una referencia a un **MailItem** de Outlook y se colocan los datos necesarios para el mensaje de correo en las variables. Una vez que el procedimiento sabe dónde están los datos en la hoja de cálculo, es posible seguir adelante y establecer las propiedades requeridas de Microsoft Outlook.

```
With objMail
            .To = strEmail
            .Body = strBody
            .Subject = strSubject
            .Send
End With
```

La propiedad **To** devuelve o establece una lista de cadenas delimitadas por punto y coma con los nombres de los destinatarios. En este ejemplo se usa un destinatario por cada correo enviado. La propiedad **Body** devuelve o establece una cadena que representa el mensaje de texto a enviar en el correo. La propiedad **Subject** se utiliza para especificar el asunto del correo electrónico. Finalmente, el método **Send** envía el mensaje de correo. Si prefieres no enviar el correo, puedes verlo sustituyendo el método **Send** por **Display**:

```
With objMail
            .To = strEmail
            .Body = strBody
            .Subject = strSubject
            .Display
End With
```

5. Introduce el siguiente procedimiento en el mismo módulo donde introdujiste **CorreoMasivo**:

```
Sub LlamadaCorreoMasivo()
    CorreoMasivo EmailCol:=4, _
    BeginRow:=2, _
    EndRow:=6, _
    SubjCol:=2, _
    NameCol:=1, _
    AmountCol:=3
End Sub
```

El procedimiento anterior llama al procedimiento **CorreoMasivo** y le pasa los parámetros que indican el número de columna de la dirección del destinatario, las filas inicial y final de los datos, la columna donde se encuentra el asunto, la columna que contiene el nombre del empleado y el número de columna con el importe a devolver.

6. Ejecuta el procedimiento **LlamadaCorreoMasivo**.
   Excel comienza a ejecutar el procedimiento especificado. El primer destinatario en la hoja que debe recibir el correo se muestra en la Imagen 6.6.

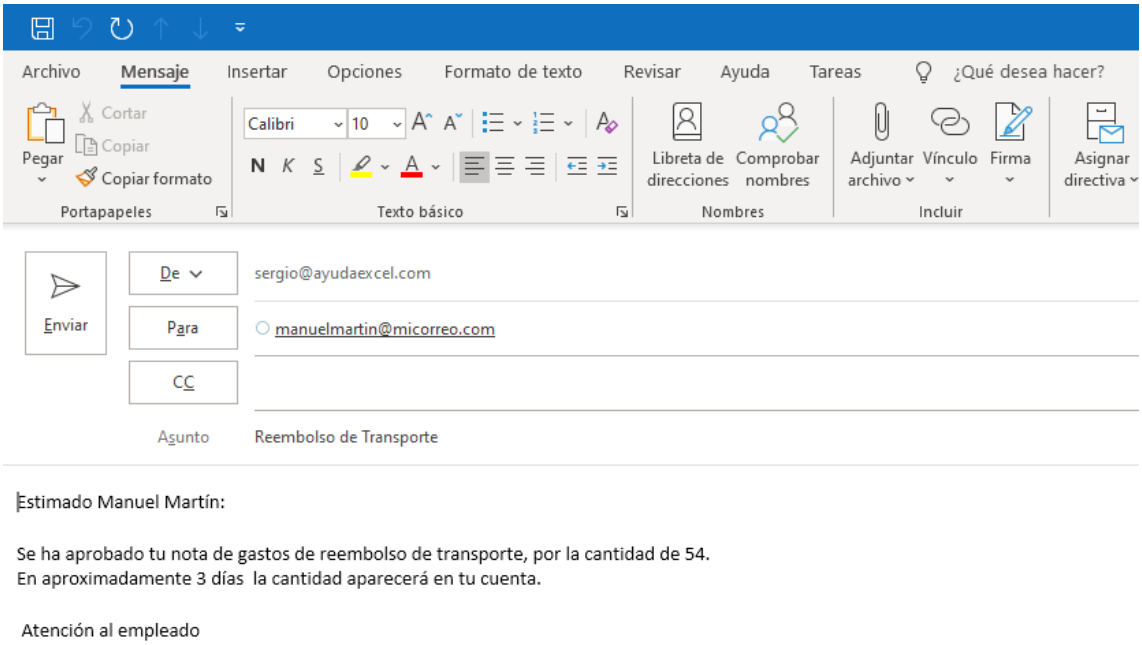


Imagen 6.6 Correo de ejemplo enviado desde Outlook a través de un procedimiento VBA.

7 Resumen

En este capítulo has aprendido a imprimir y utilizar varias técnicas de envío de correo electrónico para la presentación y distribución de libros de Excel. También aprendiste a programar las opciones de página e impresión, a configurar las impresoras y a usar los eventos relacionados con la impresión para realizar tareas de formato y cálculo de datos antes de la impresión.

A continuación, practicaste varios métodos para enviar libros de Excel a través del correo electrónico como archivos adjuntos o incrustados en el cuerpo del mensaje.

El siguiente capítulo se centra en el uso y la programación de tablas de Excel.

Capítulo 21
Las tablas

A lo largo de los años la gente ha utilizado hojas de cálculo para almacenar y extraer datos de bases de datos. Actualmente una hoja de cálculo de Excel permite a los usuarios almacenar datos en 1.048.576 filas y 16.384 columnas. Además, los datos pueden ser fácilmente ordenados, filtrados resumidos y validados. Si necesitas crear cualquier tipo de tabla y almacenarla en una hoja de cálculo, este capítulo te resultará útil. Se mostrará la interfaz de usuario de las tablas y aprenderás a acceder y a trabajar con las tablas de Excel desde procedimientos VBA.

1 Introducción a las tablas de Excel

Las tablas son grupos de celdas que sirven para almacenar datos relacionados entre sí y se gestionan de forma independiente a los datos de otras celdas de la hoja. Las tablas no son nuevas. Se conocían con el nombre de listas cuando se introdujeron por primera vez en Excel 2003. Es posible tener muchas tablas en la misma hoja pero no se pueden superponer. Cada tabla se trata como una entidad única y puede ser ordenada, filtrada y compartida de forma individual. Las tablas pueden reconocerse fácilmente en una hoja, pues Excel habilita automáticamente el filtrado en la fila de encabezados de cada columna (ver Imagen 2.1). Podemos utilizar esta función para ordenar los datos de forma ascendente o descendente o para crear una vista personalizada de los datos. Cuando se crea una tabla a partir de un rango de celdas y no se especifica si la tabla contiene encabezados de columna, Excel los añade automáticamente (Columna1, Columna2, etc.) al rango.

2 Crear una tabla desde la interfaz de usuario

Para crear una tabla en Excel, seleccionamos previamente el rango de celdas que contiene los datos que queremos incluir en la tabla y luego seleccionamos **Insertar > Tabla**. Excel muestra el cuadro de diálogo **Crear tabla** que se muestra en la Imagen 2.2, donde podemos aceptar la selección actual de celdas o cambiar el rango de datos. Para ver cómo se hace comenzaremos escribiendo un procedimiento VBA que obtenga datos de la base de datos Northwind.mdb.

	A	B	C	D	E
1	OrderID	CustomerID	Freight		
2	10248	WILMK	32,38		
3	10249	TRADH	11,61		
4	10250	HANAR	65,83		
5	10251	VICTE	41,34		
6	10252	SUPRD	51,3		
7	10253	HANAR	58,17		
8	10254	CHOPS	22,98		
9	10255	RICSU	148,33		
10	10256	WELLI	13,97		
11	10257	HILAA	81,91		
12	10258	ERNSH	140,51		
13	10259	CENTC	3,25		
14	10260	OLDWO	55,09		
15	10261	QUEDE	3,05		

Imagen 2.1 Una tabla en una hoja de cálculo.

1. Crea un nuevo libro de trabajo y llámalo Capítulo 21 – Tablas.xlsm. Guárdalo en la carpeta habitual: C:\Archivos Manual VBA\.
2. Presiona **Alt + F11** para acceder al editor de VBA. A continuación, haz clic en el nombre del proyecto en el **Explorador de proyectos** y haz clic en **Insertar – Módulo**.
3. En la ventana **Propiedades**, modifica la propiedad `Name` para ponerle `Tablas`.
4. A continuación, haz clic en **Herramientas – Referencias** y activa la casilla de verificación correspondiente a **Microsoft ActiveX Data Objects Library** (6.1 o una versión anterior). A continuación, haz clic en **Aceptar** para salir del cuadro de diálogo.
5. En la ventana **Código** del módulo `Tablas`, introduce el siguiente procedimiento:

```
Sub ObtenerPedidos()
    Dim conn As New ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim strPath As String
    Dim wks As Worksheet
    Dim j As Integer

    strPath = "C:\archivos manual vba\Northwind.mdb"
    Worksheets.Add
    Set wks = ThisWorkbook.ActiveSheet
    conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
    & "Data Source=" & strPath & ";"
    ' Crea el conjunto de registros en la tabla Categories
    Set rst = conn.Execute(CommandText:="Select OrderID," & _
    "CustomerID, Freight from Orders", _
```

```
        Options:=adCmdText)
        rst.MoveFirst
        ' Transfiere los datos a Excel
        ' y obtiene los nombres de los campos primero
        With wks.Range("A1")
            .CurrentRegion.Clear
            For j = 0 To rst.Fields.Count - 1
                .Offset(0, j) = rst.Fields(j).Name
            Next j
            .Offset(1, 0).CopyFromRecordset rst
            .CurrentRegion.Columns.AutoFit
            .Cells(1, 1).Select
        End With
        rst.Close
        conn.Close
        Set rst = Nothing
        Set conn = Nothing
    End Sub
```

Activa la ventana de Excel y selecciona la Hoja1.

6. Presiona **Alt + F8** para mostrar el cuadro de diálogo **Macro**. Selecciona `ObtenerPedido` de la lista de macros y haz clic en **Ejecutar**.
   Los datos se obtendrán de la tabla Pedidos y serán colocados en una hoja nueva.
7. En la ventana de Excel haz clic en la ficha **Insertar > Tabla**.
   Excel mostrará el cuadro de diálogo **Crear tabla** y seleccionará el rango de celdas que identifique como tabla (ver Imagen 2.2). Podemos cambiar este rango predefinido por otro rango que definamos.

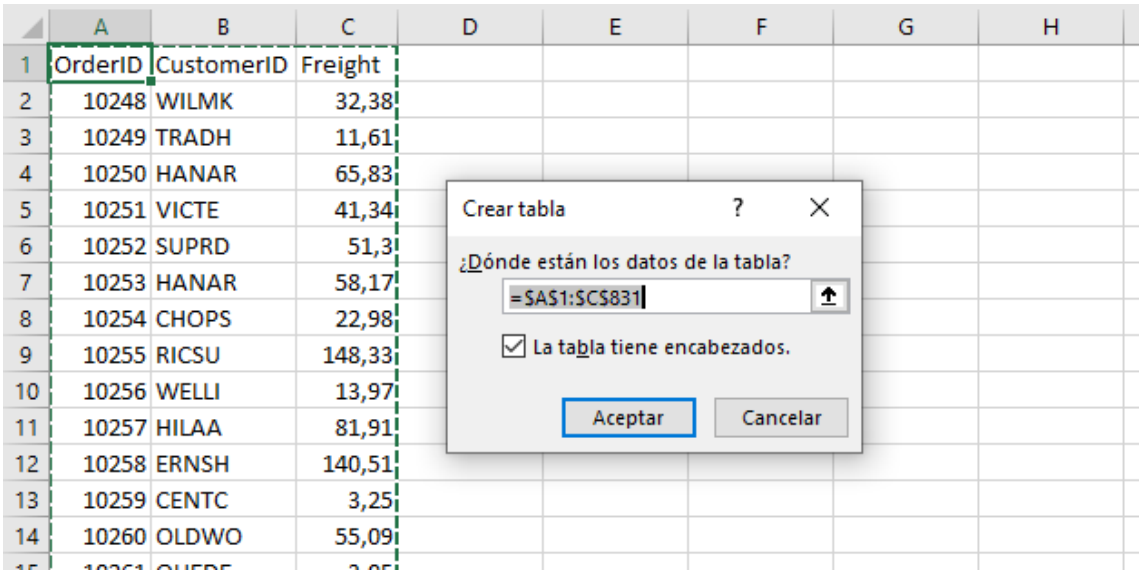


Imagen 2.2 Convirtiendo un rango de celdas en una tabla de Excel.

8. Haz clic en **Aceptar** para salir del cuadro de diálogo.
   La tabla estará lista para usar o compartir con otros.

3 Crear una tabla utilizando VBA

Para crear una tabla de Excel mediante programación, usamos el objeto `ListObject`, que representa una lista de objetos en la hoja. El objeto `ListObject` es miembro de la colección `ListObjects`. Esta colección contiene todos los objetos de lista de la hoja de cálculo.

En la sección anterior hemos aprendido a usar VBA para recibir datos desde Access y convertirlos manualmente en una tabla de Excel utilizando la interfaz de usuario. En esta sección modificaremos el procedimiento `ObtenerPedidos` de la sección anterior para hacer que automáticamente se cree una tabla a partir de los datos:

1. En la ventana **Código** del módulo `Tablas`, modifica el procedimiento `ObtenerPedidos` de la siguiente forma:
   a. Agrega la siguiente declaración a la sección de declaraciones del proyecto:

```
Dim rng As Range
```

   b. Escribe las siguientes declaraciones justo antes de la declaración `End Sub`:

```
' crea una tabla a partir de los datos
Set rng = wks.Range(Range("A1").CurrentRegion.Address)
wks.ListObjects.Add xlSrcRange, rng
```

La primera declaración detrás del comentario establecerá la variable de objeto `rng` para que apunte al rango que celdas que queremos convertir en tabla. La segunda declaración utiliza el método `Add` de la colección `ListObjects` para crear una tabla sobre ese rango de celdas específico. La constante `xlSrcRange` especifica que la fuente de datos de la tabla es un rango de Excel mientras que la variable de objeto `rng` indica un objeto `Range` que representa a la fuente de datos.

El procedimiento resultante debe ser el siguiente:

```
Sub ObtenerPedidos2()
    Dim conn As New ADODB.Connection
    Dim rst As ADODB.Recordset
    Dim strPath As String
    Dim wks As Worksheet
    Dim j As Integer
    Dim rng As Range

    strPath = "C:\Archivos Manual VBA\Northwind.mdb"
    Worksheets.Add
    Set wks = ThisWorkbook.ActiveSheet
```

```
        conn.Open "Provider=Microsoft.Jet.OLEDB.4.0;" _
        & "Data Source=" & strPath & ";"
        ' Crea el conjunto de registros en la tabla Categories
        Set rst = conn.Execute(CommandText:="Select OrderID," & _
        "CustomerID, Freight from Orders", _
        Options:=adCmdText)
        rst.MoveFirst
        ' Transfiere los datos a Excel
        ' y obtiene los nombres de los campos primero
        With wks.Range("A1")
            .CurrentRegion.Clear
            For j = 0 To rst.Fields.Count - 1
                .Offset(0, j) = rst.Fields(j).Name
            Next j
            .Offset(1, 0).CopyFromRecordset rst
            .CurrentRegion.Columns.AutoFit
            .Cells(1, 1).Select
        End With
        rst.Close
        conn.Close
        Set rst = Nothing
        Set conn = Nothing
        ' crea una tabla a partir de los datos
        Set rng = wks.Range(Range("A1").CurrentRegion.Address)
        wks.ListObjects.Add xlSrcRange, rng
    End Sub
```

2. Activa la ventana de Excel.
3. Presiona **Alt + F8** para mostrar el cuadro de diálogo **Macro**. Selecciona el procedimiento `ObtenerPedidos2` y presiona en **Ejecutar**.
   Los datos recibidos de la tabla Orders de Access se sitúan en una tabla en una hoja nueva. Cuando se usa el método `Add` de la colección `ListObject` para crear la tabla de Excel, puedes especificar algunos argumentos, como se muestra en la siguiente tabla:

Nombre de argumento	Descripción
`SourceType` (opcional)	Indica el tipo de dato para la lista. Se puede usar uno de los siguientes tipos: Datos externos (`xlSrcExternal`) Rango de Excel (`xlSrcRange`) Datos XML (`xlSrcXML`)

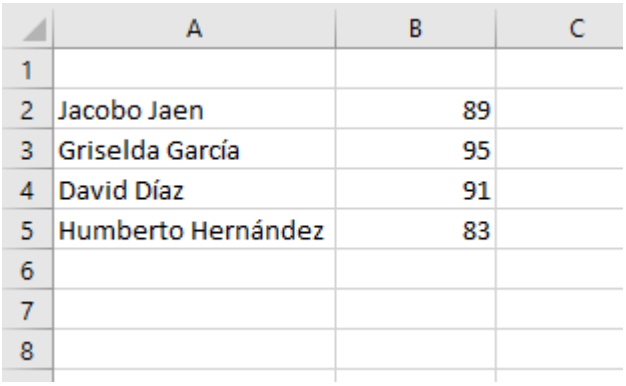
Nombre de argumento	Descripción
	Si se omite el argumento, el tipo por defecto será **xlSrcRange**.
Source (opcional cuando **SourceType** sea **xlSrcRange**) (obligatorio cuando **SourceType** sea **xlSrcExternal**)	Este argumento puede ser una de estas opciones: Una matriz de valores String especificando una conexión a la fuente: **0**: para indicar la URL de SharePoint. **1**: Para indicar el nombre de la lista. **2**: para indicar la ViewGUID (identifica la vista de la lista de SharePoint). Un objeto **Range** que represente la fuente de datos. Si se omite el argumento, **Source** será el rango devuelto por defecto al crear la tabla.
LinkSource (opcional)	Indica si una fuente de datos externa se debe enlazar al objeto **ListObject**. El argumento **SourceType** debe ser **xlSrcExternal**.
HasHeaders (opcional)	Indica si los datos de la lista contienen etiquetas de columnas. Puedes usar una de las siguientes constantes: **xlGuess**, **xlNo** o **xlYes**. Si **Source** no tiene encabezados de columna, Excel genera automáticamente las etiquetas Columna1, Columna2, etc.
Destination (obligatorio cuando **SourceType** sea **xlSrcExternal**) (ignorado cuando **SourceType** sea **xlSrcRange**)	Indica la esquina superior izquierda del nuevo objeto de lista. Se usa el objeto **Range** haciendo referencia a una sola celda. Si el rango de destino no está vacío, los datos existentes se sobrescribirán.

Observa que los argumentos del método **Add** del objeto **ListObject** son opcionales. Si se omiten estos argumentos, Excel utilizará su propia lógica para identificar el rango de celdas de la tabla y determinará si contiene encabezados de columna. Excel supone que las celdas contiguas con datos forman parte de la tabla. Si la primera fila del rango de datos identificado contiene texto, Excel lo trabará como fila de encabezados.

4 Los encabezados de columna de la tabla

Cuando creamos una tabla, Excel añade encabezados de columna automáticamente. En función del tipo de dato encontrado en la primera fila del rango de datos, éstos pueden establecerse como encabezados de columna o desplazar el rango de datos hacia abajo para insertar una nueva fila. Como no sabemos qué hará Excel en cada situación particular, es una buena idea suministrar el valor para el argumento `HasHeaders` en el código VBA, como se muestra en la siguiente tabla. Echemos un vistazo a cómo controlar los encabezados de columna en la tabla de Excel:

1. Crea una hoja nueva en el libro Capítulo 21 – Tablas.xlsm y escribe algunos datos de ejemplo como los de la Imagen 4.1.



	A	B	C
1			
2	Jacobo Jaen	89	
3	Griselda García	95	
4	David Díaz	91	
5	Humberto Hernández	83	
6			
7			
8			

Imagen 4.1 Datos antes de convertirlos en una tabla.

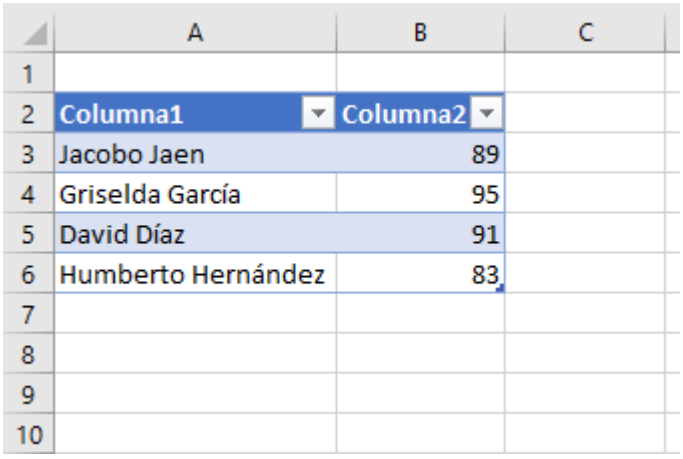
2. Activa el editor de VBA e introduce el siguiente procedimiento en la ventana **Código** del módulo `Tablas`:

```
Sub Encabezados()
    Dim rng As Range
    Dim wks As Worksheet

    Set wks = ActiveSheet
    Set rng = wks.Range("A2:B5")
    wks.ListObjects.Add SourceType:=xlSrcRange, Source:=rng, _
    XlListObjectHasHeaders:=xlNo
End Sub
```

Como los datos en la Imagen 4.1 no tienen encabezados de columna, especificamos `xlNo` como argumento de `HasHeaders`. Cuando Excel ejecuta este procedimiento, agregará los encabezados predeterminados (Columna1, Columna2) y el resto de los datos se deslizará una fila hacia abajo, como se muestra en la Imagen 4.2.

3. Sitúa el cursor del ratón en cualquier punto del procedimiento `Encabezados` y presiona **F5** para ejecutarlo.
4. Regresa a la ventana de Excel para ver el resultado de la ejecución del procedimiento. A veces no querrás que Excel desplace los datos hacia abajo cuando la tabla no incluya encabezado. Para evitarlo, se recomienda que especifiques para la tabla un rango de

datos en el que la primera fila esté en blanco. Por ejemplo, para evitar que Excel desplace los datos hacia abajo una fila, especifica el rango A1:B5 y utiliza `xlYes` en el argumento `HasHeaders`. Antes de probar esto, convierte la tabla de Excel que acabas de crear en un rango normal.



	A	B	C
1			
2	Columna1	Columna2	
3	Jacobo Jaen	89	
4	Griselda García	95	
5	David Díaz	91	
6	Humberto Hernández	83	
7			
8			
9			
10			

Imagen 4.2 Un rango de datos tras la conversión a tabla. Observa que Excel ha añadido en la fila 2 los encabezados de las columnas.

5. Selecciona cualquier celda de la tabla y haz clic en la ficha contextual **Diseño de tabla > Convertir en rango**. Haz clic en **Sí** cuando Excel solicite confirmación.
   Observa que Excel crea un rango normal, pero los encabezados de columnas se mantienen.
6. Elimina la primera fila (encabezados) y guarda los cambios.
7. Regresa al editor de VBA e introduce el siguiente procedimiento en el módulo `Tablas`:

```
Sub Encabezados2()
    Dim rng As Range
    Dim wks As Worksheet

    Set wks = ActiveSheet
    Set rng = wks.Range("A1:B5")
    wks.ListObjects.Add SourceType:=xlSrcRange, _
    Source:=rng, XlListObjectHasHeaders:=xlYes
End Sub
```

8. Ejecuta el procedimiento y observa el resultado Imagen 5.1).

5 Varias tablas en una hoja

Hemos visto en la sección anterior, cómo los datos de la tabla se deslizan una fila hacia abajo cuando el rango de datos no contiene encabezados de columnas. En caso de tener varias tablas en la misma hoja, este hecho puede provocar problemas cuando situamos otra tabla justo a la derecha de la primera.

	A	B	C
1	Columna1	Columna2	
2	Jacobo Jaen	89	
3	Griselda García	95	
4	David Díaz	91	
5	Humberto Hernández	83	
6			
7			
8			

Imagen 5.1 Un rango de celdas tras la conversión a tabla. Observa que Excel ha añadido la fila de encabezados en la fila1, que se encontraba vacía.

Cuando tenemos más de una tabla en una hoja y las queremos manipular con VBA, es una buena práctica asignarles nombres para poder hacer referencia a ellas fácilmente en el código. Aunque podemos hacer referencia a las tablas usando sus nombres por defecto, se trabaja de una forma más clara si cada tabla tiene un nombre significativo. Por defecto Excel asigna los nombres Tabla1, Tabla2, Tabla2, etc. a las tablas de la hoja. Para nombrar una tabla u obtener el nombre de una tabla existente, se utiliza la propiedad `Name` del objeto `ListObject`. Por ejemplo, la siguiente declaración introducida en la ventana **Inmediato** devuelve el nombre de la tabla en la hoja activa:

```
?ActiveSheet.ListObjects(1).Name
```

Para cambiar el nombre a la Tabla1, podemos escribir la siguiente instrucción en la ventana **Inmediato** y presionar **Intro**:

```
ActiveSheet.ListObjects(1).Name = "Notas Alumnos"
```

Desde ahora ya podemos referirnos a la primera tabla de la hoja activa como **Notas_Alumnos**.

El siguiente procedimiento utiliza la propiedad `ListsObject` para obtener una referencia de la primera tabla en la Hoja3. A continuación, la propiedad `Name` se usa para asignar un nombre a la tabla referenciada.

```
Sub DefinirNombreTabla()
    Dim wks As Worksheet
    Dim lst As ListObject

    Set wks = ActiveWorkbook.Worksheets("Hoja3")
    Set lst = wks.ListObjects(1)
    lst.Name = "Notas alumnos 1er trimestre"
End Sub
```

6 El objeto ListObject

El objeto `ListObject` representa una tabla en una hoja. Podemos manipular la tabla con las propiedades y métodos de las colecciones `ListColumns` y `ListRows`.

- La colección **ListColumns** contiene todos los objetos **ListColumn** en el objeto **ListObject** especificado. Cada objeto **ListColumn** es una columna de la tabla.
- Por su parte, la colección **ListRows** contiene todos los objetos **ListRow** del objeto **ListObject** especificado. Cada objeto **ListRow** es una fila de la tabla.

Podemos realizar varias operaciones en tablas de Excel utilizando propiedades y métodos del objeto **ListObject** como se muestra en las siguientes tablas:

Propiedad	Descripción

Active	Indica si una lista en la hoja de cálculo se encuentra actualmente activa. Devuelve **True** o **False**. Por ejemplo:
	`TablaActiva = ActiveSheet.ListObjects(1).active`
	`Debug.Print IsTblActive`

DataBodyRange	Devuelve un objeto **Range** que representa el rango de celdas sin la fila de encabezados. Por ejemplo: `ActiveSheet.ListObjects(1).DataBodyRange.Select`
HeaderRowRange	Devuelve un objeto **Range** que representa el rango de la fila de encabezados de la tabla. Por ejemplo, usa la siguiente declaración para seleccionar la fila de encabezados: `ActiveSheet.ListObjects(1).HeaderRowRange.Select`
InsertRowRange	En Excel 2003 esta propiedad devuelve un objeto **Range** que representa la fila insertada. Esta propiedad no se utiliza en otras versiones de Excel.

ListColumns	Devuelve una colección **ListColumns** que representa todas las columnas en el objeto **ListObject**. Por ejemplo, el siguiente procedimiento elimina la última columna de la tabla:
	`Sub EliminarUltimaColumna()`
	`  Dim myList As ListObject`
	`  Dim lastCol As Integer`
	
	`  Set myList = ActiveSheet.ListObjects(1)`
	`  lastCol = myList.ListColumns.Count`
	`  myList.ListColumns(lastCol).Delete`
	`End Sub`

Propiedad	Descripción

`ListRows`	Devuelve un objeto `ListRows` que representa a todas las filas de datos del objeto `ListObject`. Por ejemplo, el siguiente procedimiento muestra en la ventana **Inmediato** el número total de filas de la tabla:
	`Sub ContarFilasTabla()`
	`Dim objRows As ListRows`
	`Set objRows = ActiveSheet.ListObjects(1).ListRows`
	`Debug.Print objRows.Count`
	`End Sub`

`Name`	Devuelve o establece el nombre del objeto `ListObject`. Por ejemplo, usa la siguiente declaración para asignar un nombre a la primera tabla de la hoja activa: `ActiveSheet.ListObjects(1).Name = "Notas alumnos"`
`QueryTable`	Devuelve el objeto `QueryTable` que proporciona un vínculo al objeto `ListObject` de un servidor SharePoint.

`Range`	Devuelve un objeto `Range` que representa el rango de celdas en el que se encuentra la tabla. Por ejemplo, la siguiente declaración muestra en la ventana **Inmediato** la dirección del rango de la tabla completa:
	`Debug.Print _`
	`ActiveSheet.ListObjects(1).Range.Address`

`SharePointURL`	Devuelve la cadena de texto que representa la URL de la tabla de SharePoint. Úsala para encontrar la dirección de una tabla compartida después de que haya sido publicada.

`ShowAutoFilter`	Indica si el Autofiltro se mostrará en la fila de encabezados (`True` o `False`). Usa la siguiente declaración para ocultar el modo AutoFiltro en una tabla:
	`ActiveSheet.ListObjects(1).ShowAutoFilter = _`
	`False`

`ShowTotals`	Indica si la fila Total es visible (`True`) o está oculta (`False`). La siguiente declaración muestra la fila Total: `ActiveSheet.ListObjects(1).ShowTotals = True`

Propiedad	Descripción
`SourceType`	Devuelve una de las constantes `XlListObjectSourceType` indicando la fuente de datos de la tabla (`xlSrcRange`, `xlSrcExternal` o `xlSrcXML`).

`TotalsRowRange`	Devuelve el rango que representa la fila de totales del objeto `ListObject`:
	`Debug.Print ActiveSheet.ListObjects(1). _`
	`TotalsRowRange.Address`

`XmlMap`	Devuelve un objeto `XmlMap` que representa la estructura XML de la tabla.

Método	Descripción
`Delete`	Elimina el objeto `ListObject` y limpia la celda de la hoja. Si la tabla está vinculada con un sitio SharePoint, eliminarlo no borra los datos del servidor.
`Publish`	Publica el objeto `ListObject` en un servidor que esté ejecutando Microsoft Windows SharePoint Services. Devuelve una cadena de texto que indica la URL de publicación en el sitio SharePoint.
`Refresh`	Este método puede ser usado únicamente con tablas vinculadas a un sitio SharePoint. Obtiene los datos actuales y la estructuras de datos desde SharePoint Server.

`Resize`	Permite redimensionar un objeto `ListObject` en un nuevo rango. Debes proporcionar el rango como argumento del método `Resize`. Suponiendo que la tabla actual se encuentra en el rango A1:B6, puedes especificar un rango nuevo así:
	`ActiveSheet.ListObjects(1).Resize _`
	`Range("A1:B3")`

`Unlink`	Elimina el vínculo con SharePoint.
`Unlist`	Convierte una tabla de Excel en un rango normal. Por ejemplo, la tabla de la hoja activa se puede convertir en un rango normal de la siguiente forma: `ActiveSheet.ListObjects(1).Unlist`
`UpdateChanges`	Actualiza la tabla en un servicio de SharePoint con los cambios hechos en la hoja.

1. Introduce el siguiente procedimiento en el módulo **Tablas**:

```
Sub DefinirNombreTabla2()
    Dim wks As Worksheet
    Dim lst As ListObject
    Dim col As ListColumn
    Dim c As Variant

    Set wks = ActiveWorkbook.Worksheets("Hoja3")
    Set lst = wks.ListObjects(1)
    With lst
        .Name = "Notas alumnos 1er trimestres"
        .ListColumns(1).Name = "Nombre Alumno"
        .ListColumns(2).Name = "Nota"
        Set col = .ListColumns.Add
        col.Name = "Nota anterior"
        Debug.Print "Rango de encabezados = " &
.HeaderRowRange.Address
        Debug.Print "Rango Total de datos = " & .Range.Address
        Debug.Print "Rango de datos = " & .DataBodyRange.Address
        For Each c In wks.Range(.HeaderRowRange.Address)
            Debug.Print c
        Next
    End With
End Sub
```

2. Activa la Hoja3 del libro Capítulo 21 – Tablas.xlsm.
3. Presiona **Alt + F8** para mostrar el cuadro de diálogo **Macro**. Selecciona el nombre de la nueva macro y haz clic en **Ejecutar**.
4. Comprueba la ventana **Inmediato** para ver el resultado:

```
Rango de encabezados = $A$1:$C$1
Rango Total de datos = $A$1:$C$5
Rango de datos = $A$2:$C$5
Nombre Alumno
Nota
Nota anterior
```

7 Filtrar datos de tablas con AutoFiltro

El desplegable Autofiltro (el botón situado en cada encabezado de columna) tiene una gran capacidad de búsqueda, haciendo tremendamente fácil encontrar datos en tablas grandes.

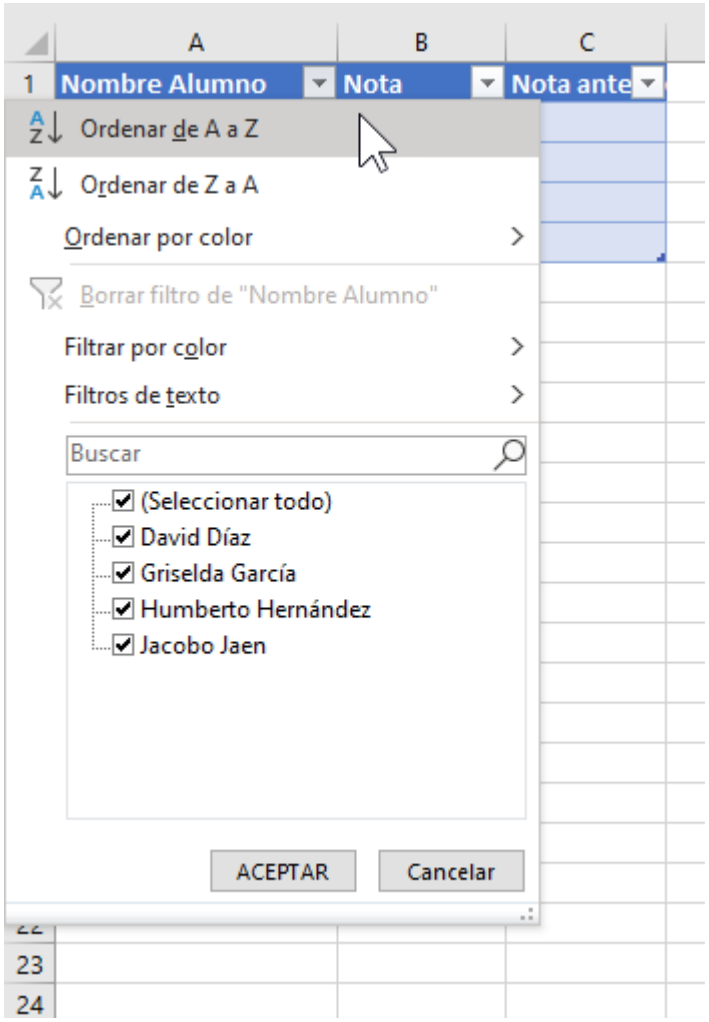


Imagen 7.1 Podemos llegar rápidamente a datos específicos introduciendo los parámetros en el cuadro de búsqueda del AutiFiltro.

Para comenzar a usar la herramienta autofiltro desde VBA, todo lo que tenemos que hacer es utilizar la grabadora de macros y realizar algunas acciones. La siguiente macro corresponde a la búsqueda de Griselda García en la tabla de la Hoja 3 del libro. Para eliminar un filtro, simplemente debemos comenzar la línea correspondiente con un apóstrofe.

```
Sub Macro4()
'
' Macro4 Macro
'

'
    ActiveSheet.ListObjects("Notas alumnos 1er
trimestres").Range.AutoFilter Field _
        :=1, Criteria1:="Griselda García"
End Sub
```

8 Filtrar datos con Segmentación de datos

Los segmentadores de datos se pueden utilizar en cualquier tabla de Excel (desde Excel 2013). Hacen que filtrar una tabla de Excel sea muy sencillo. Los segmentadores son controles flotantes que podemos ubicar en cualquier parte de la hoja. La apariencia de nuestras hojas de cálculo puede ser mucho más profesional pues se eliminan los botones de AutoFiltro de los encabezados de las tablas y usan grandes botones de color para filtrar los datos.

Para usar los segmentadores de datos, simplemente seleccionamos cualquier celda de la tabla y hacemos clic en el botón **Insertar segmentación de datos** en la ficha contextual **Diseño de tabla** (ver Imagen 8.1).

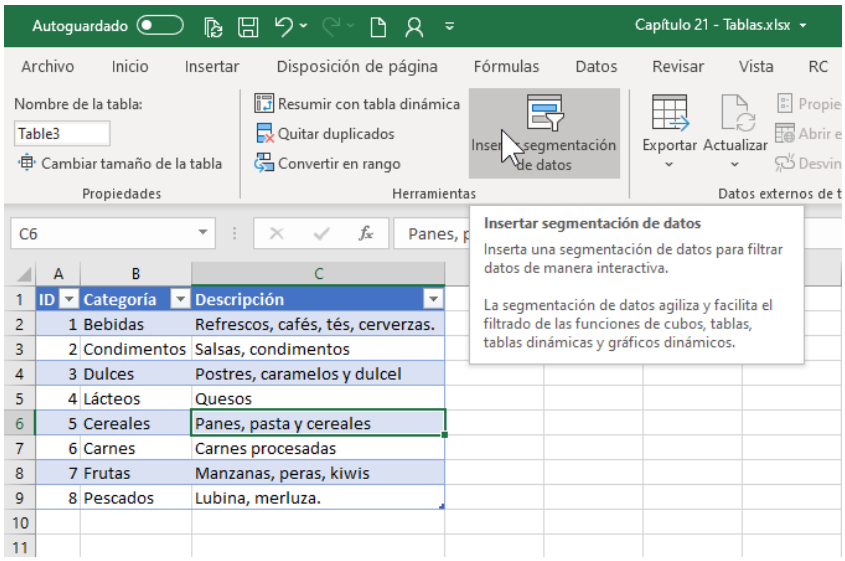


Imagen 8.1 Insertando un segmentador de datos en una hoja que contiene una tabla.

Excel muestra el cuadro de diálogo **Insertar segmentación de datos**, donde podemos seleccionar las columnas que deseamos usar para crear los filtros. Cada segmentador de datos corresponde a una columna.

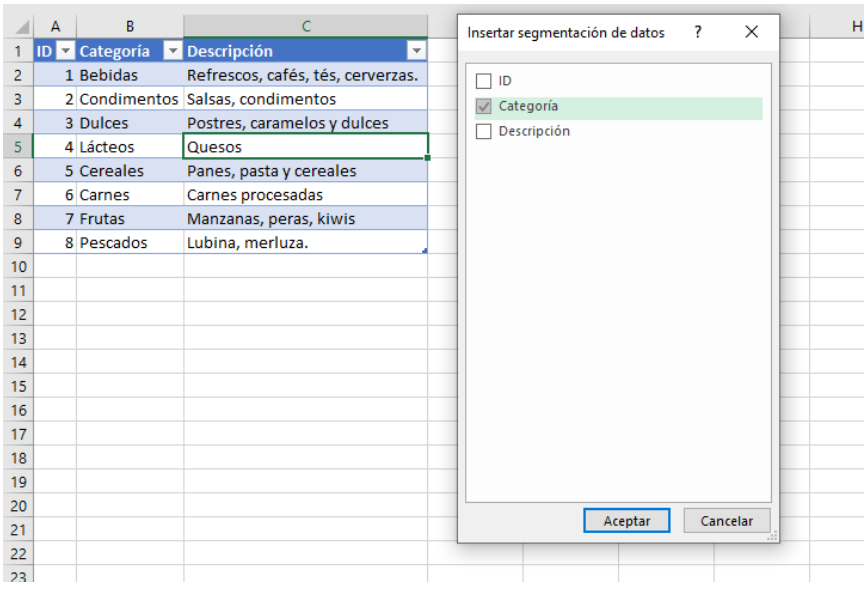


Imagen 8.2 Seleccionando las columnas para los segmentadores.

Cuando seleccionamos las columnas en el cuadro de diálogo y presionamos **Aceptar**, se crean los filtros y se muestran en la hoja:

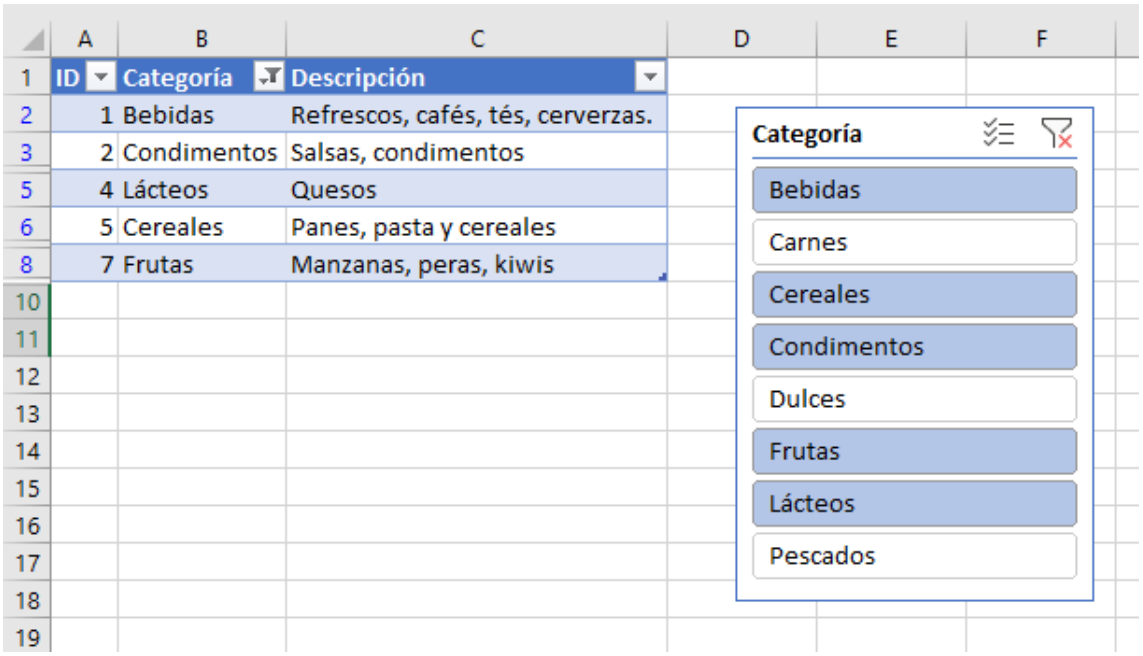


Imagen 8.3 Se ha usado la columna Categoría para filtrar la tabla. Podemos seleccionar varios botones del segmentador.

Para eliminar el filtro de la tabla, hacemos clic en el botón **Borrar filtro** en la esquina superior derecha del segmentador. Esto devolverá la tabla a su vista completa.

Con VBA también podemos manejar los segmentadores de datos. Este es un ejemplo de macro creada con la grabadora de macros:

```
Sub Macro5()
'
' Macro5 Macro
'

'
    ActiveWorkbook.SlicerCaches.Add2(ActiveSheet. _
ListObjects("Table3"), "Categoría" _
        ).Slicers.Add ActiveSheet, , "Categoría", "Categoría", _
        189.75, 639.75, 144, 198.75
    ActiveSheet.Shapes.Range(Array("Categoría")).Select
    ActiveSheet.Shapes("Categoría").IncrementLeft -355.5
    ActiveSheet.Shapes("Categoría").IncrementTop -165
    ActiveSheet.Shapes("Categoría").ScaleWidth 1.3020833333, _
    msoFalse, msoScaleFromTopLeft
    ActiveSheet.Shapes("Categoría").ScaleHeight 1.0377358491, _
    msoFalse, msoScaleFromTopLeft
```

```
    With _
ActiveWorkbook.SlicerCaches("SegmentaciónDeDatos_Categoría")
        .SlicerItems("Bebidas").Selected = True
        .SlicerItems("Carnes").Selected = False
        .SlicerItems("Cereales").Selected = False
        .SlicerItems("Condimentos").Selected = False
        .SlicerItems("Dulces").Selected = False
        .SlicerItems("Frutas").Selected = False
        .SlicerItems("Lácteos").Selected = False
        .SlicerItems("Pescados").Selected = False
    End With
    With
ActiveWorkbook.SlicerCaches("SegmentaciónDeDatos_Categoría")
        .SlicerItems("Bebidas").Selected = True
        .SlicerItems("Cereales").Selected = True
        .SlicerItems("Condimentos").Selected = True
        .SlicerItems("Dulces").Selected = True
        .SlicerItems("Frutas").Selected = True
        .SlicerItems("Pescados").Selected = True
        .SlicerItems("Carnes").Selected = False
        .SlicerItems("Lácteos").Selected = False
    End With

ActiveWorkbook.SlicerCaches("SegmentaciónDeDatos_Categoría"). _
Slicers("Categoría").Style = "SlicerStyleLight6"
End Sub
```

9 Eliminar tablas

Podemos eliminar una tabla usando uno de los siguientes métodos:

- Interfaz de usuario:
  - Seleccionamos la tabla y hacemos clic en **Inicio > Celdas > Eliminar > Eliminar celdas**.
  - Si no necesitamos la hoja que contiene la tabla, podemos eliminar la hoja entera.
- Con VBA:
  - Utilizamos el método `Delete` para eliminar la tabla y sus datos.
  - Utilizamos el método `Unlist` para convertir la tabla en un rango de celdas.
  - Utilizamos el método `Unlink` para eliminar la relación entre la tabla de la hoja de cálculo y la lista en el sitio SharePoint. Una tabla desvinculada no se puede volver a vincular.

10 Resumen

En este capítulo has conocido las tablas de Excel. Has aprendido a obtener información de una base de datos de Microsoft Access, a convertirla en una tabla y a disfrutar de la funcionalidad de una base de datos en la hoja de cálculo. También has aprendido el funcionamiento de las tablas a través del modelo de objetos de Excel y cómo se manipulan mediante VBA.

En el próximo capítulo, aprenderás a programar dos objetos de Excel que se utilizan para el análisis de datos: **`PivotTable`** y **`PivotChart`**.

Capítulo 22
Las tablas dinámicas

Las tablas dinámicas sirven a millones de usuarios de las aplicaciones de Office como herramientas para organizar y presentar información de diversas fuentes. Si no estás familiarizado con esta herramienta, ahora es el momento de comenzar. Mediante las tablas dinámicas y los gráficos dinámicos, puedes analizar datos desde múltiples perspectivas. Las tablas dinámicas permiten arrastrar los encabezados de una tabla para organizarlos de forma que los datos se muestren dinámicamente de la forma que deseas. Al igual que las tablas dinámicas, los gráficos dinámicos son interactivos y te permiten ver los datos de diferentes maneras al cambiar la posición o el detalle de los campos. Tanto las tablas dinámicas como los gráficos dinámicos te permiten concentrarte en la comprensión de los datos en lugar de organizarlos.

1 Crear un informe de tabla dinámica

Antes de crear una tabla dinámica necesitamos preparar los datos. Podemos obtener estos datos de una de las siguientes fuentes:

- Un rango de una hoja de Excel (podemos traer los datos desde otras fuentes pegándolos en Excel).
- Una fuente de datos externa (podemos conectarnos a un archivo de Access o a una base de datos SQL directamente).

La Imagen 1.1 muestra los datos volcados en Excel desde una base de datos SQL Server. El archivo descargado se llama ListaEquipos.xlsx. Este archivo contiene más de 1.400 filas de datos, lo que podría dificultar su resumen y análisis si no fuera gracias a las tablas dinámicas.

Comencemos nuestra aproximación a las tablas dinámicas usando los comandos predefinidos de la cinta de opciones:

1. Copia el libro ListaEquipos.xlsx a la carpeta C:\Archivos Manual VBA y ábrelo con Excel. Selecciona cualquier celda del rango.
2. Haz clic en la ficha **Insertar > Tabla dinámica**.

Aparece el cuadro de diálogo **Crear tabla dinámica** que se muestra en la Imagen 1.2.

	A	B	C	D	E	F	G
1	**Cliente**	**Tipo de equipo**	**ID**	**Comienzo Garantía**	**Tiempo garantía**	**Partes Garantia**	**Tipo Garantía**
2	Titan Expres, S.L.	Escáner	11483	30/09/2016	1	NULL	Garantía total
3	Titan Expres, S.L.	Escáner	11487	27/09/2015	1	NULL	Garantía total
4	Titan Expres, S.L.	Escáner	11486	25/09/2018	1	NULL	Garantía total
5	J3 Equipamiento	Escáner	11481	24/09/2018	1	NULL	Garantía total
6	J3 Equipamiento	Escáner	11480	20/09/2018	1	NULL	Garantía total
7	J3 Equipamiento	Escáner	11485	17/09/2018	1	NULL	Garantía total
8	J3 Equipamiento	Escáner	11484	15/09/2018	1	NULL	Garantía total
9	J3 Equipamiento	Escáner	11479	11/09/2018	1	NULL	Garantía total
10	J3 Equipamiento	Escáner	11478	05/09/2018	1	NULL	Garantía total
11	J3 Equipamiento	Escáner	11477	31/08/2018	1	NULL	Garantía total
12	J3 Equipamiento	Escáner	11482	17/08/2015	1	NULL	Garantía total
13	J3 Equipamiento	Escáner	11476	07/08/2015	1	NULL	Garantía total
14	Maine Citi	Escáner	11475	03/08/2017	1	NULL	Garantía total
15	Maine Citi	Escáner	11474	31/07/2017	1	NULL	Garantía total
16	Maine Citi	Escáner	11473	27/07/2017	1	NULL	Garantía total
17	Maine Citi	Escáner	11471	18/07/2017	1	NULL	Garantía total
18	Maine Citi	Escáner	11472	18/07/2017	1	NULL	Garantía total

Imagen 1.1 Fuente de datos para la tabla dinámica.

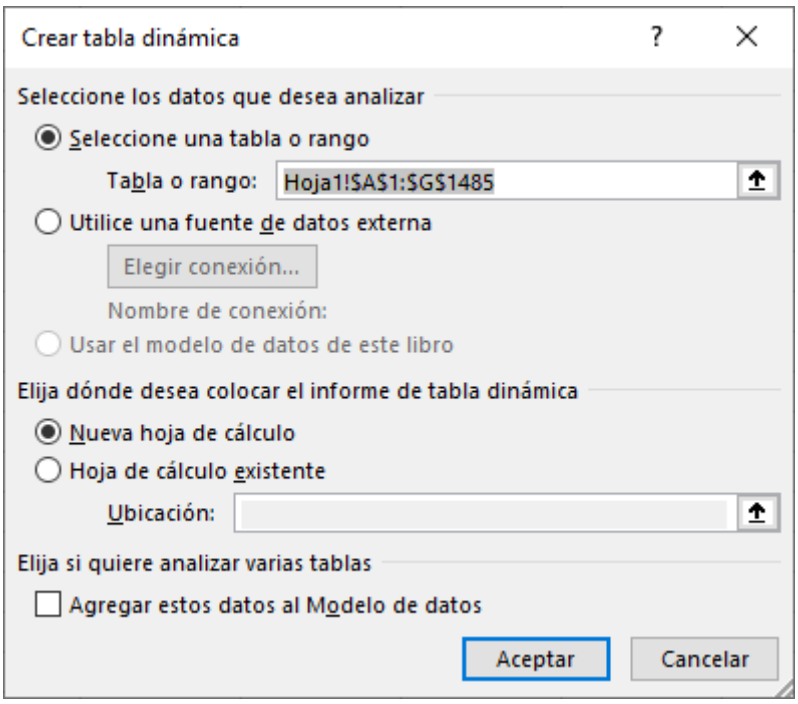


Imagen 1.2Cuadro de diálogo Crear tabla dinámica.

Observa que hay tres partes en el cuadro. En la parte de arriba debes elegir la fuente de datos para la tabla. Puede ser una tabla o rango de una hoja de Excel o datos externos. La parte central del cuadro de diálogo te da a elegir entre situar la tabla dinámica en una nueva hoja o en una ya existente. Finalmente, la casilla de verificación de la parte de abajo te permite agregar los datos al modelo de datos de Excel. Esto se verá más adelante en el capítulo.

3. Asegúrate de que las opciones **Seleccione una tabla o rango** y **Nueva hoja de cálculo** estén seleccionadas.
   También fíjate en que el rango mostrado en el cuadro **Tabla o rango** contiene todos los datos con los que deseas hacer el informe. El rango aparecerá automáticamente si la celda activa se encuentra dentro del rango. Si la celda seleccionada está fuera del rango, tendrás que seleccionar los datos manualmente.
4. Haz clic en **Aceptar**.
   Se creará una nueva hoja y dentro de ella se insertará la estructura de la tabla dinámica. En la parte derecha se mostrará el panel **Campos de tabla dinámica** (ver Imagen 1.3).
   Observa cómo se muestran todos los campos en la lista de campos del panel. Cada campo tiene una casilla de verificación para indicar fácilmente qué campos debe incluir el informe. La tabla se crea seleccionando campos. Por ejemplo, cuando activamos la casilla de verificación Cliente, observarás que el campo se agrega automáticamente al área de filas del panel de la tabla dinámica y, además la tabla dinámica muestra los datos en el lugar correspondiente.

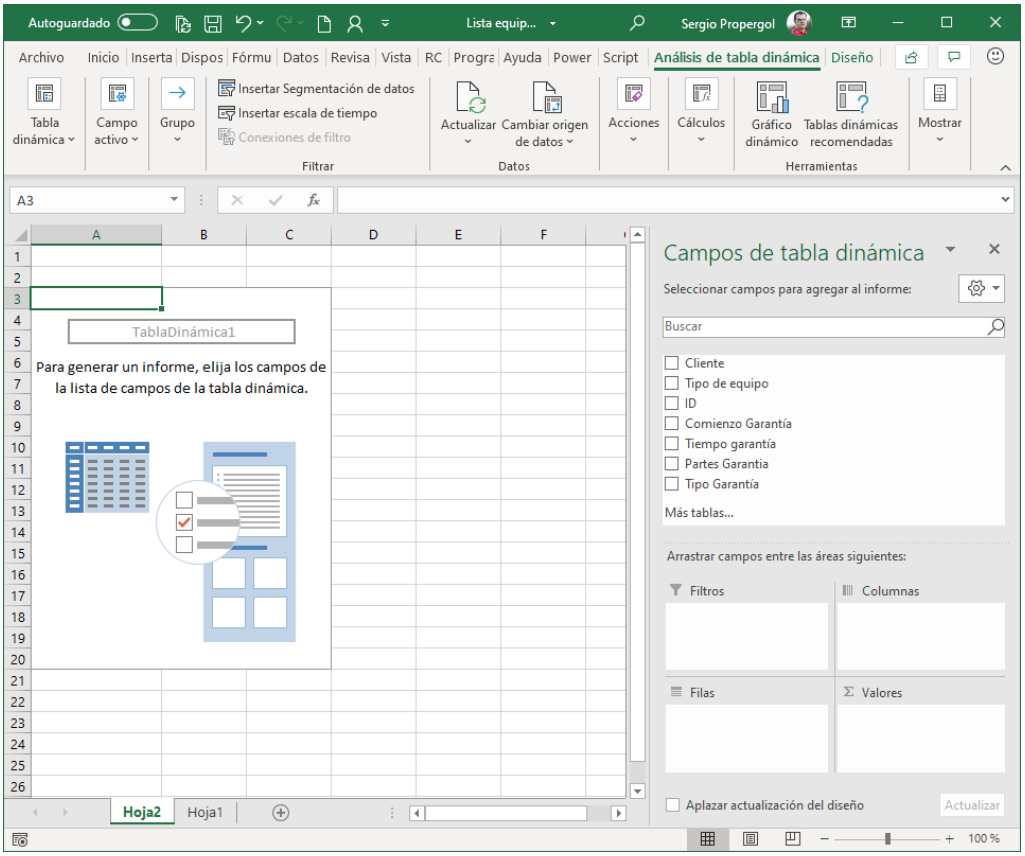


Imagen 1.3 La estructura de la tabla está lista para seleccionar los campos.

De forma predeterminada, los campos que contienen texto son situados en el área de filas y los campos numéricos aparecen en el área de valores, como se muestra en la Imagen 1.4. Puedes especificar el tipo de cálculo que necesitas para resumir los datos haciendo clic en la flecha del nombre del campo situado en el área de valores y seleccionando **Configuración de campo de valor**. La función predeterminada es la suma. La Imagen 1.4 muestra la tabla dinámica usando la función Recuento. Puedes

fácilmente ajustar la posición de los campos arrastrándolos de un área a otra en el panel **Campos de tabla dinámica**.

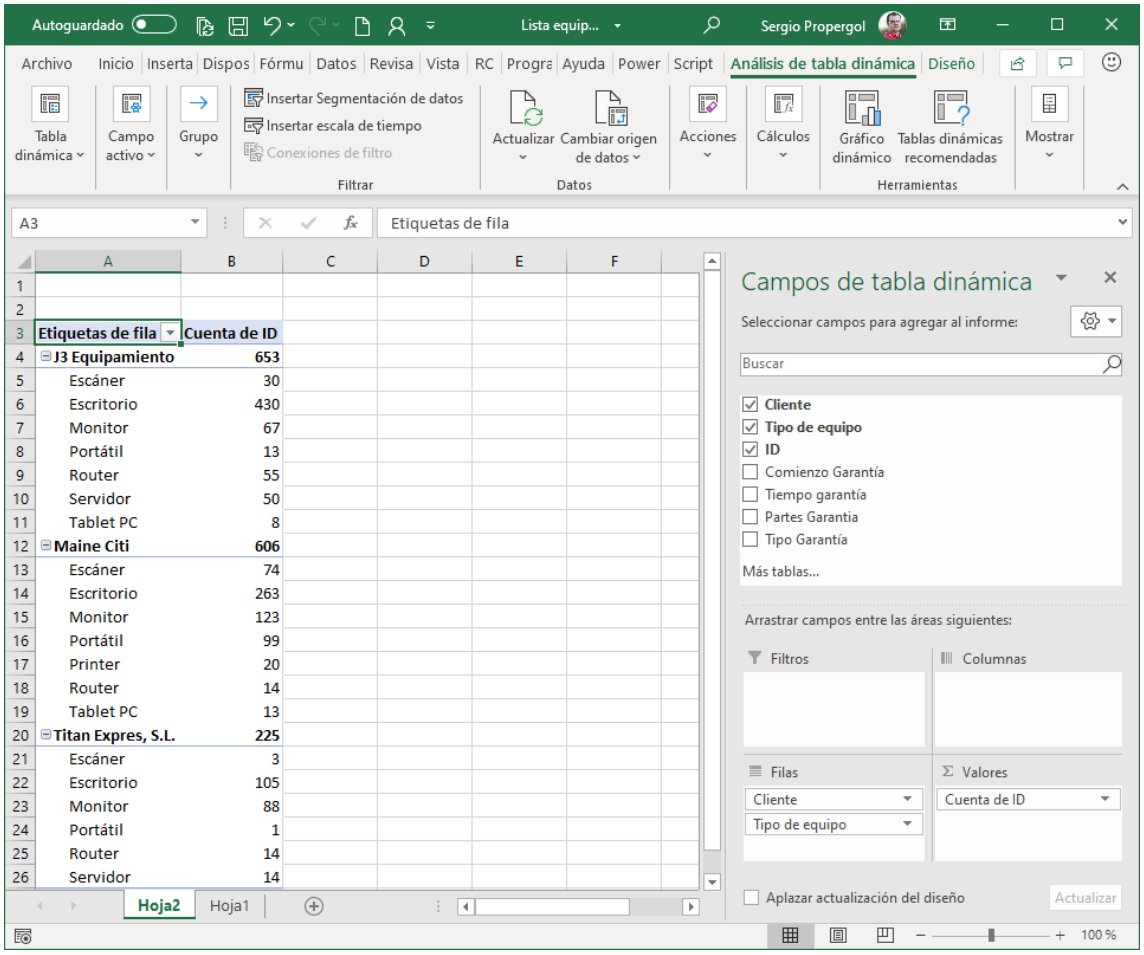


Imagen 1.4 Agregando campos a la tabla dinámica.

La Imagen 1.5 muestra las diferentes vistas disponibles en el panel **Campos de tabla dinámica**.

La parte de abajo del panel contiene cuatro áreas donde poder situar los campos de datos:

- El área **Filas** contiene los campos que deseas "*mostrar por*". Por ejemplo, si deseas crear un informe "*por*" Cliente, deberías situar ese campo en el área de filas. El área de filas puede contener más de un campo. En el informe que estás creando también quieres que se muestre "por" Tipo de equipo, así que se debe situar el campo en el área de filas. Si sitúas Tipo de equipo debajo de Cliente, los datos se agruparán primero por Cliente y luego por Tipo de equipo. En caso de que no se ajuste a las necesidades, es posible cambiar el orden arrastrando los campos.
- El área **Columnas** contiene los campos que responden a la pregunta "¿qué?". Por ejemplo "¿qué tipo de información quieres mostrar en cada uno de los campos del área de filas? En este ejemplo, la tabla dinámica mostrará el Tipo de garantía. Como queremos ver todos los tipos de garantías de cada cliente y tipo de equipo, sitúa el campo Tipo de garantía en el área de columnas.
- El área **Valores** muestra los datos que quieres analizar. En este ejemplo, queremos encontrar el número total de unidades (Tipo de equipo) correspondientes a cada tipo

de garantía. El área **Valores** puede contener un campo numérico. Una vez situado el campo numérico en el área es posible escoger un tipo de cálculo (suma, recuento, promedio, etc.) que realizar con los datos.

- En el área **Filtros** no es obligatorio situar campos. Los campos de filtro añaden una tercera dimensión al análisis de datos. Más adelante en el capítulo, cuando generemos una tabla dinámica con VBA, añadiremos un campo de filtro para experimentar con ese área.

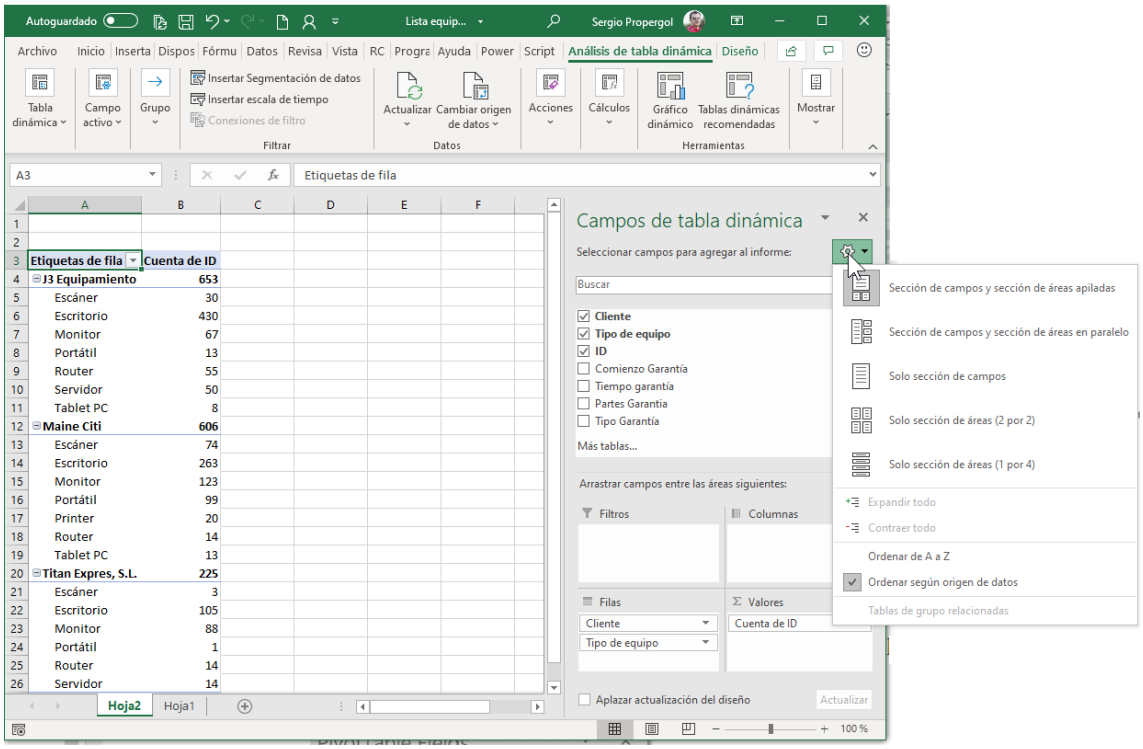


Imagen 1.5 Es posible cambiar fácilmente la disposición de las secciones del panel Campos de tabla dinámica.

Observa que no hemos ubicado todos los campos de la fuente de datos en la tabla dinámica. Únicamente se trabaja con aquellos datos que necesites. Más tarde se pueden agregar otros campos.

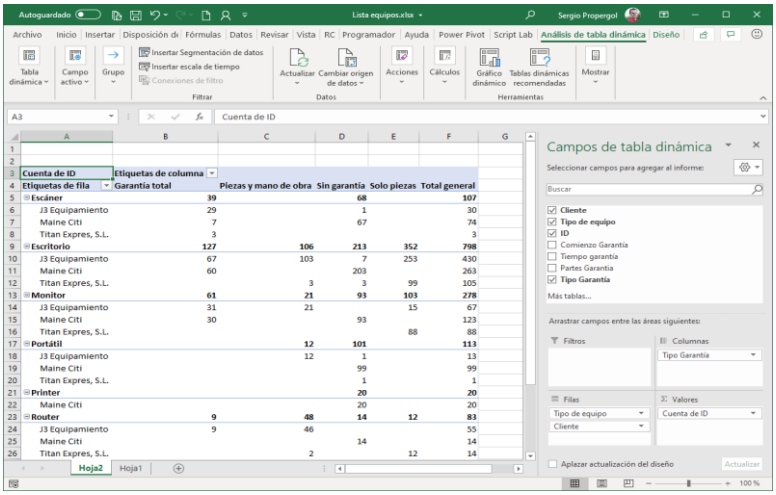


Imagen 1.6 La tabla finalizada.

5. Sitúa los campos en las áreas que se muestran en la Imagen 1.6.
   Cuando la tabla dinámica se encuentra seleccionada, el panel **Campos de tabla dinámica** permanece visible en la parte derecha de la pantalla para que puedas realizar modificaciones de forma fácil. Para ocultar el panel solo tienes que hacer clic fuera de la tabla.

Las tabla dinámicas se usan para analizar y presentar datos. Esto significa que no está permitido introducir datos directamente en una tabla dinámica. Podemos hacer cualquier cambio en la fuente de datos y hacer clic en el botón **Actualizar** de la ficha contextual **Análisis de tabla dinámica** para actualizarla.

6. Para ver los datos desde un punto de vista diferente, modifica la disposición de los campos de la tabla dinámica como se muestra en la Imagen 1.7.

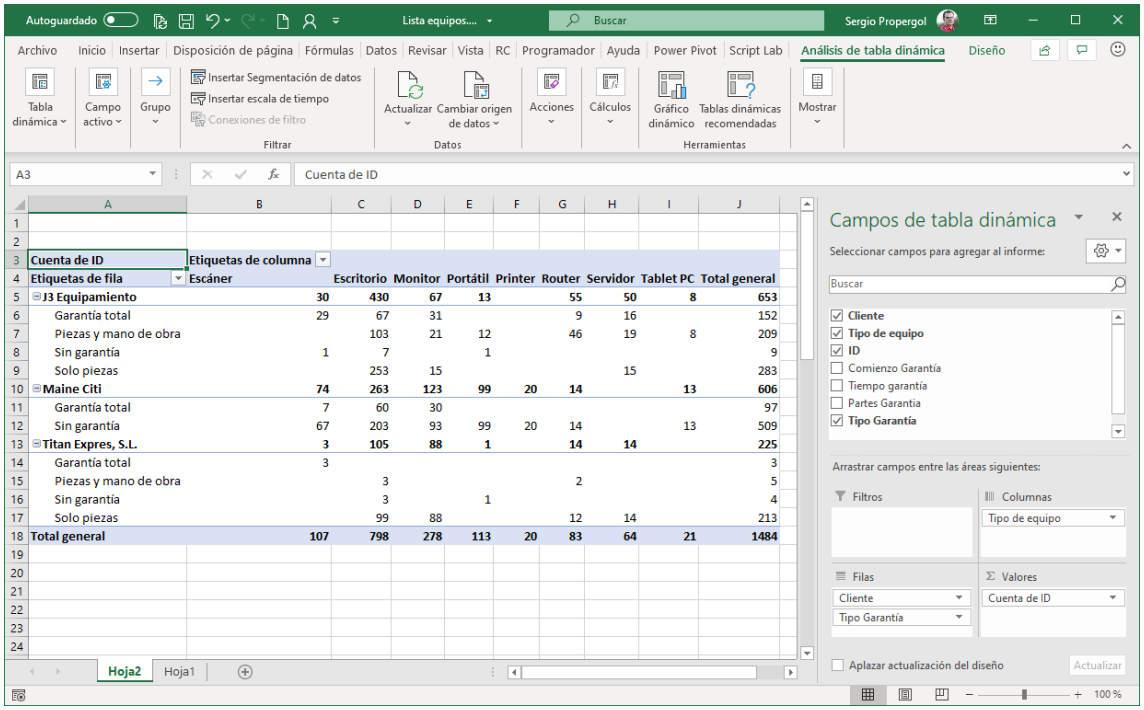
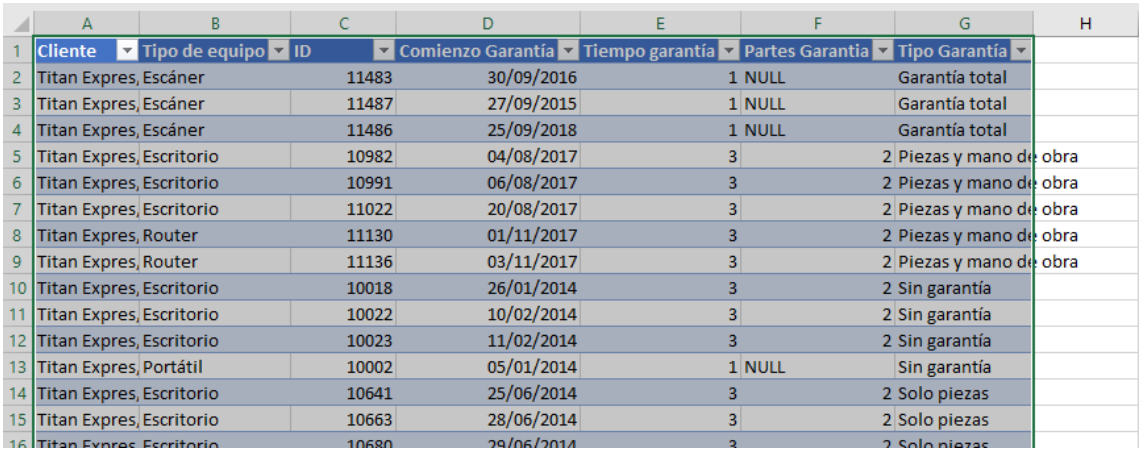


Imagen 1.7 Otra vista del informe de tabla dinámica.

7. Haz doble clic en la celda J13.
   Excel creará una hoja nueva en el libro activo. En ella se mostrarán todos los registros que forman parte del valor en el que has hecho doble clic (ver Imagen 1.8).
8. Guarda los cambios en el archivo ListaEquipos.xlsx. No lo cierres, pues será usado en el siguiente ejercicio.

Profundizar en los datos del paso 7 es una buena opción para conocer el origen del valor sobre el que se hace doble clic. Si hacemos muchas veces doble clic para ver estos detalles nos encontraremos con un libro lleno de hojas que probablemente no deseemos mantener. Para borrar estas hojas podemos hacerlo manualmente o desde un procedimiento de eventos de VBA como se describe en la siguiente sección.



	A	B	C	D	E	F	G	H
1	Cliente	Tipo de equipo	ID	Comienzo Garantía	Tiempo garantía	Partes Garantia	Tipo Garantía	
2	Titan Expres	Escáner	11483	30/09/2016	1	NULL	Garantía total	
3	Titan Expres	Escáner	11487	27/09/2015	1	NULL	Garantía total	
4	Titan Expres	Escáner	11486	25/09/2018	1	NULL	Garantía total	
5	Titan Expres	Escritorio	10982	04/08/2017	3	2	Piezas y mano de obra	
6	Titan Expres	Escritorio	10991	06/08/2017	3	2	Piezas y mano de obra	
7	Titan Expres	Escritorio	11022	20/08/2017	3	2	Piezas y mano de obra	
8	Titan Expres	Router	11130	01/11/2017	3	2	Piezas y mano de obra	
9	Titan Expres	Router	11136	03/11/2017	3	2	Piezas y mano de obra	
10	Titan Expres	Escritorio	10018	26/01/2014	3	2	Sin garantía	
11	Titan Expres	Escritorio	10022	10/02/2014	3	2	Sin garantía	
12	Titan Expres	Escritorio	10023	11/02/2014	3	2	Sin garantía	
13	Titan Expres	Portátil	10002	05/01/2014	1	NULL	Sin garantía	
14	Titan Expres	Escritorio	10641	25/06/2014	3	2	Solo piezas	
15	Titan Expres	Escritorio	10663	28/06/2014	3	2	Solo piezas	
16	Titan Expres	Escritorio	10680	29/06/2014	3	2	Solo piezas	

Imagen 1.8 Al hacer doble clic en una celda de la tabla dinámica se puede ver el detalle de los datos que componen.

2 Eliminar hojas de detalle de la tabla dinámica

En el ejercicio anterior (paso 7) hemos visto como crear una hoja con los detalles de un total de la tabla dinámica. El siguiente ejemplo muestra dos procedimientos de eventos que permiten especificar qué hojas deseas mantener o borrar automáticamente una vez que has examinado los datos. Este ejercicio requiere haber completado los pasos del ejercicio anterior.

1. En el libro ListaEquipos.xlsx renombra la hoja que contiene la tabla dinámica **Informe**.
2. Guarda el libro en el formato compatible para macros en la carpeta C:\Archivos Manual VBA.
3. Presiona **Alt + F11** para acceder al editor de VBA. En el **Explorador de proyectos** haz doble clic en el objeto **`ThisWorkbook`**.
4. En la ventana **Código** del objeto **`ThisWorkbook`**, introduce la siguiente declaración y los dos procedimientos:

```
' Variables globales
Dim flag As Boolean ' Variable booleana que indica
' si se debe borrar la hoja
Dim strPivSheet As String ' Mantiene la hoja que contiene
' la tabla dinámica
Dim strDrillSheet As String ' Mantiene la hoja que contiene
' el detalle
Dim strPivSource As String ' Mantiene la hoja que contiene
' la fuente de datos de la tabla dinámica
Private Sub Workbook_SheetActivate(ByVal Sh As Object)
    If strPivSheet = "" Then Exit Sub
    If Sh.Name <> strPivSheet Then
        If InStr(1, strPivSource, Sh.Name) = 0 Then
            If MsgBox("¿Deseas eliminar la hoja " & Sh.Name & _
                " del libro " & vbCrLf _
                & "al regresar a la tabla dinámica?", _
```

```
                vbYesNo + vbQuestion, _
                "Hoja: ¿Deseas mantenerla?") = vbYes Then
                flag = True
                strDrillSheet = Sh.Name
            Else
                flag = False
                Exit Sub
            End If
        End If
    End If
    If ActiveSheet.Name = strPivSheet And flag = True Then
        Application.DisplayAlerts = False
        Worksheets(strDrillSheet).Delete
        Application.DisplayAlerts = True
        flag = False
    End If
End Sub
Private Sub Workbook_SheetBeforeDoubleClick(ByVal Sh As Object, _
ByVal Target As Range, Cancel As Boolean)
    With ActiveSheet
        If .PivotTables.Count > 0 Then
            strPivSource = ActiveSheet.PivotTables(1).SourceData
            If ActiveCell.PivotField.Name <> "" And _
                IsEmpty(Target) Then
                MsgBox "La celda seleccionada no contiene datos " & _
                "- no contiene detalles."
                Cancel = True
                Exit Sub
            End If
            strPivSheet = ActiveSheet.Name
        End If
    End With
End Sub
```

El procedimiento de evento `Workbook_SheetActivate` preguntará al usuario si la hoja de detalle debe eliminarse cuando se active la hoja que contiene la tabla dinámica. Si el usuario responde **Si** en el cuadro de mensaje, la variable `flag` se establecerá en `True`. Como por defecto Excel muestra un mensaje de confirmación si la hoja está a punto de eliminarse, el código del procedimiento deshabilita este mensaje para que la eliminación de la hoja se haga sin intervención del usuario.

El procedimiento `Workbook_SheetBeforeDoubleClick` deshabilita la creación de las hojas de detalle si el usuario hace clic en una celda de la tabla dinámica que se encuentra vacía. Si la celda no está vacía, el nombre de la hoja de la tabla dinámica se copiará en la variable `strPivSheet`. Como no queremos eliminar la hoja que contiene

la fuente de datos de la tabla usaremos la propiedad `SourceData` de la colección `PivotTables` para almacenar el nombre de la hoja de la fuente de datos y el rango de datos en la variable global `strPivSource`.

5. Guarda los cambios hechos en el editor de VBA.
6. Vuelve a la ventana de Excel, selecciona la hoja **Informe** y haz doble clic en la celda J9.
   Excel ejecutará el código que se encuentra dentro del procedimiento `Workbook_SheetBeforeDoubleClick` y procederá a ejecutar el código de `Workbook_SheetActivate`. Como J9 no está vacía, Excel preguntará si quieres eliminar la hoja de detalla cuando se regrese a la hoja **Informe**.
7. Haz clic en el botón **Sí** del cuadro de mensaje.
   En este momento no sucede nada. Excel simplemente ha guardado la información que necesita.
8. Haz clic en la hoja **Informe**.
   En ese momento, Excel elimina la hoja de detalle y activa la hoja **Informe**.
9. Haz doble clic en la celda B15 en la hoja de la tabla dinámica.
   Excel muestra el mensaje que indica que esa celda no contiene datos.
10. Guarda el libro ListaEquipos.xlsm y ciérralo.

3 Crear una tabla dinámica con VBA

Aunque el proceso de creación de las tablas dinámicas ha experimentado muchas mejoras, algunos usuarios podrían seguir encontrando confusa la creación de informes de este tipo. Para esos usuarios puede ser conveniente generar las tablas dinámicas mediante código VBA. Con VBA también se pueden hacer muchos cambios de formato. Esta sección muestra la forma de trabajar con tablas dinámicas desde VBA. Comenzaremos creando el informe anterior usando la misma fuente de datos.

1. Abre el libro ListaEquipos.xlsx.
   Haz clic con el botón derecho del ratón en el nombre de la hoja Hoja1 y selecciona la opción **Mover o copiar** del menú contextual.
2. En el cuadro de diálogo **Mover o copiar**, selecciona la opción **(nuevo libro)** que encontrarás en el cuadro desplegable **Al libro**. Haz clic en la casilla de verificación **Crear una copia**. Haz clic en **Aceptar** para continuar.
   Excel crea un nuevo libro con una hoja llamada Hoja1. Esta hoja ha sido copiada desde el archivo ListaEquipos.xlsx.
3. Guarda el nuevo libro en el formato adecuado para macros con el nombre Capítulo 22 – Tablas dinámicas.xlsm.
4. Inserta tres hojas nuevas y guarda el libro de nuevo.
5. Cierra el libro ListaEquipos.xlsx dejando abierto el archivo Capítulo 22 – Tablas dinámicas.xlsm.
6. Presiona **Alt + F11** para acceder al editor de VBA.
7. En el **Explorador de proyectos** selecciona el objeto `VBAProject` (Capítulo 22 – Tablas dinámicas.xlsm) y haz clic en el menú **Insertar – Módulo**.
8. En la ventana **Código** del módulo introduce el siguiente procedimiento:

```
Sub CrearTD()
    Dim wksData As Worksheet
    Dim rngData As Range
    Dim wksDest As Worksheet
    Dim pvtTable As PivotTable

    ' Establece variables de objeto
    Set wksData = ThisWorkbook.Worksheets("Hoja1")
    Set rngData = wksData.UsedRange
    Set wksDest = ThisWorkbook.Worksheets("Hoja2")
    ' Crea el esqueleto de la tabla dinámica
    Set pvtTable = _
    wksData.PivotTableWizard(SourceType:=xlDatabase, _
    SourceData:=rngData, TableDestination:=wksDest.Range("B5"))
    ' Cierra el panel de Campos de tabla dinámica
    ' que aparece automáticamente
    ActiveWorkbook.ShowPivotTableFieldList = False
    ' Agrega los campos a la tabla dinámica
    With pvtTable
        .PivotFields("Cliente").Orientation = xlRowField
        .PivotFields("Tipo de equipo").Orientation = xlRowField
        .PivotFields("Tipo garantía").Orientation = _
        xlColumnField
        With .PivotFields("ID")
            .Orientation = xlDataField
            .Function = xlCount
        End With
        .PivotFields("ID").Orientation = xlPageField
    End With
    ' Ajusta el ancho de columnas para que los encabezados
    ' se muestren completos
    wksDest.UsedRange.Columns.AutoFit
End Sub
```

El procedimiento anterior crea una tabla dinámica usando el método `PivotTableWizard` del objeto `Worksheet`. Este método tiene unos pocos argumentos que especifican el tipo de la fuente de datos, su localización y el lugar donde debe ubicarse la tabla. Todos estos argumentos son opcionales; no obstante, es una buena idea utilizarlos en el código. Como puedes crear una tabla dinámica desde varias fuentes de datos usando la constante `xlDatabase` en el argumento `SourceType`, el código dice específicamente que los datos llegan desde un rango de Excel. Para crear una tabla dinámica desde otra tabla dinámica, usa `xlPivotTable` en

este argumento. Si tus datos deben extraerse de una base de datos (como en un ejemplo posterior), se especificará `xlExternal` como `SourceType`. El argumento `SourceData` en el ejemplo anterior es una referencia al rango usado en la hoja que contiene la fuente de datos. El argumento `TableDestination` hace referencia a la celda B2 de la Hoja2 del libro actual. Esta será la esquina superior izquierda de la tabla dinámica.

Es importante entender que cuando llamas al método `PivotTableWizard`, estás creando una tabla dinámica en blanco. Todos los campos de la fuente de datos están ocultos. Para hacerlos visibles, necesitas añadirlos a las áreas apropiadas del panel **Campos de tabla dinámica** (Filas, Columnas, Valores y Filtro). Mientras se está creando el informe de tabla dinámica, la lista de campos aparece en la parte derecha de la hoja. Como la tabla dinámica se crea mediante macros, no es necesario que se muestre, así que estableciendo en False la propiedad `ShowPivotTableFieldList`, se ocultará.

Por cada campo que desees mostrar en la tabla dinámica, debes establecer la propiedad `Orientation` del objeto `PivotField`. Utiliza las siguientes constantes: `xlRowField`, `xlColumnField`, `xlDataField` y `xlPageField`. Observa que en el campo ID, situado en el área **Valores**, se establece la propiedad `Function` del objeto `PivotField` en `xlCount`.

9. Cuando estás creando la tabla dinámica con VBA, puede que necesites comprobar si ya existe en la hoja de destino. Puedes situar el siguiente código justo debajo del código que establece las variables de objeto:

```
If wksDest.PivotTables.Count > 0 Then
    MsgBox "La hoja " & wksDest.Name & _
    " ya contiene una tabla dinámica."
    Exit Sub
End If
```

10. Ejecuta el procedimiento `CrearTD`.
    Cuando actives la ventana de Excel, la Hoja2 debería contener una tabla dinámica como la que se muestra en la Imagen 3.1.

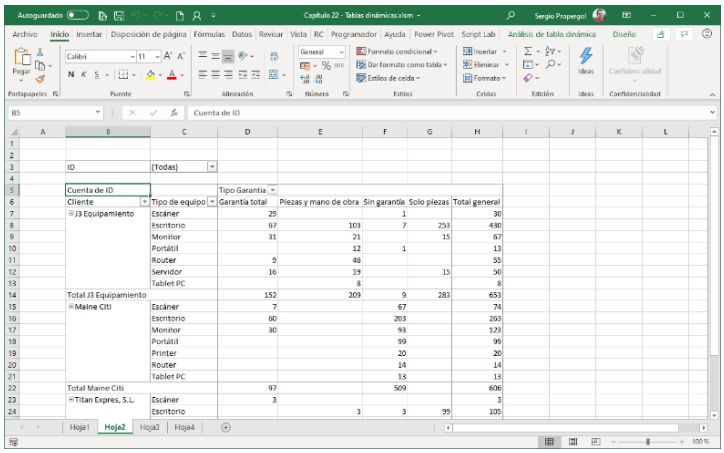


Imagen 3.1 Una tabla dinámica creada a partir de código VBA.

4 Crear una tabla dinámica desde una base de datos Access

Podemos utilizar el mismo método **PivotTableWizard** del objeto **Worksheet** para crear un informe de tabla dinámica a partir de una fuente de datos externa. Comencemos por crear una tabla dinámica a partir de una base de datos Access. Utilizaremos un controlador de Microsoft Access para conectarnos a la base de datos Northwind.mdb y luego llamaremos al método **PivotTableWizard** para crear una tabla dinámica vacía. Llenaremos la tabla con los datos estableciendo la propiedad **Orientation** de los objetos **PivotField**.

1. Inserta un módulo nuevo en el libro Capítulo 22 – Tablas dinámicas.xlsm e introduce el siguiente procedimiento:

```
Sub TDExterna()
    Dim strConn As String
    Dim strQuery_1 As String
    Dim strQuery_2 As String
    Dim myArray As Variant
    Dim destRange As Range
    Dim strPivot As String

    strConn = "Driver={Microsoft Access Driver (*.mdb)};" & _
    "DBQ=" & "C:\Archivos Manual VBA\Northwind.mdb;"
    strQuery_1 = "SELECT Customers.CustomerID, " & _
    "Customers.CompanyName," & _
    "Orders.OrderDate, Products.ProductName, Sum([Order " & _
    "Details].[UnitPrice]*[Quantity]*(1-[Discount])) " & _
    "AS Total " & _
    "FROM Products INNER JOIN ((Customers INNER JOIN Orders " _
    & "ON Customers.CustomerID = "
    strQuery_2 = "Orders.CustomerID) INNER JOIN [Order Details] " & _
    "ON Orders.OrderID = [Order Details].OrderID) ON " & _
    "Products.ProductID = [Order Details].ProductID " & _
    "GROUP BY Customers.CustomerID, Customers.CompanyName, " & _
    "Orders.OrderDate, Products.ProductName;"
    myArray = Array(strConn, strQuery_1, strQuery_2)
    Worksheets.Add
    Set destRange = ActiveSheet.Range("B5")
    strPivot = "PivotFromAccess"
    ActiveSheet.PivotTableWizard _
    SourceType:=xlExternal, _
    SourceData:=myArray, _
    TableDestination:=destRange, _
    TableName:=strPivot, _
    SaveData:=False, _
    BackgroundQuery:=False
```

```
    ' Oculta el panel Campos de tabla dinámica
    ActiveWorkbook.ShowPivotTableFieldList = False
    ' Agrega los campos a la tabla dinámica
    With ActiveSheet.PivotTables(strPivot)
        .PivotFields("ProductName").Orientation = xlRowField
        .PivotFields("CompanyName").Orientation = xlRowField
        With .PivotFields("Total")
            .Orientation = xlDataField
            .Function = xlSum
            .NumberFormat = "#,##0.00 €"
        End With
        .PivotFields("CustomerID").Orientation = xlPageField
        .PivotFields("OrderDate").Orientation = xlPageField
    End With
    ' Ajusta el ancho de las columnas
    ActiveSheet.UsedRange.Columns.AutoFit
End Sub
```

Cuando utilizas el método **PivotTablwWizard** del objeto **Worksheet** para crear la tabla dinámica desde una fuente externa, necesitas especificar como mínimo los siguientes argumentos:

Argumento	Descripción
SourceType	Usa la constante **xlExternal** para indicar que los datos de la tabla dinámica vienen de una fuente externa.
SourceData	Especifica una matriz que contiene dos o más elementos. El primero debe ser la cadena de conexión con la base de datos. El segundo argumento es la declaración SQL que hará la consulta a la base de datos. Si la declaración SQL tiene más de 255 caracteres, divídela en varias cadenas y pasa cada una de ellas a la matriz por separado. En el procedimiento del ejemplo anterior, la declaración SQL necesaria para obtener los datos requeridos tiene más de 255 caracteres. Por eso se divide en dos cadenas: **strQuery_1** y **strQuery_2**. A continuación se conectan ambas cadenas en una sola. **myArray = Array(strConn, strQuery_1, strQuery_2)**
TableDestination	Especifica el rango de la hoja donde se ubicará la tabla dinámica.
TableName	Especifica el nombre de la tabla dinámica que quieres crear.

Además de los argumentos anteriores, el procedimiento **TDExterna** usa los argumentos opcionales **SaveData** y **BackgroundQuery**:

Savedata	Le dice a VBA si guardar la tabla dinámica si el archivo es grabado. Estableciéndolo en **False**, la tabla dinámica no se guardará. Esto permite ahorrar espacio en disco.
BackgroundQuery	Cuando es **False**, este argumento le dice a VBA que no ejecute otras operaciones en Excel en segundo plano hasta que la consulta se complete.

Después de crear la tabla dinámica, el procedimiento especifica dónde se deben ubicar los campos devueltos por la declaración SQL.

2. Ejecuta el procedimiento **TDExterna** para generar la tabla dinámica. El resultado se muestra en la Imagen 4.1.

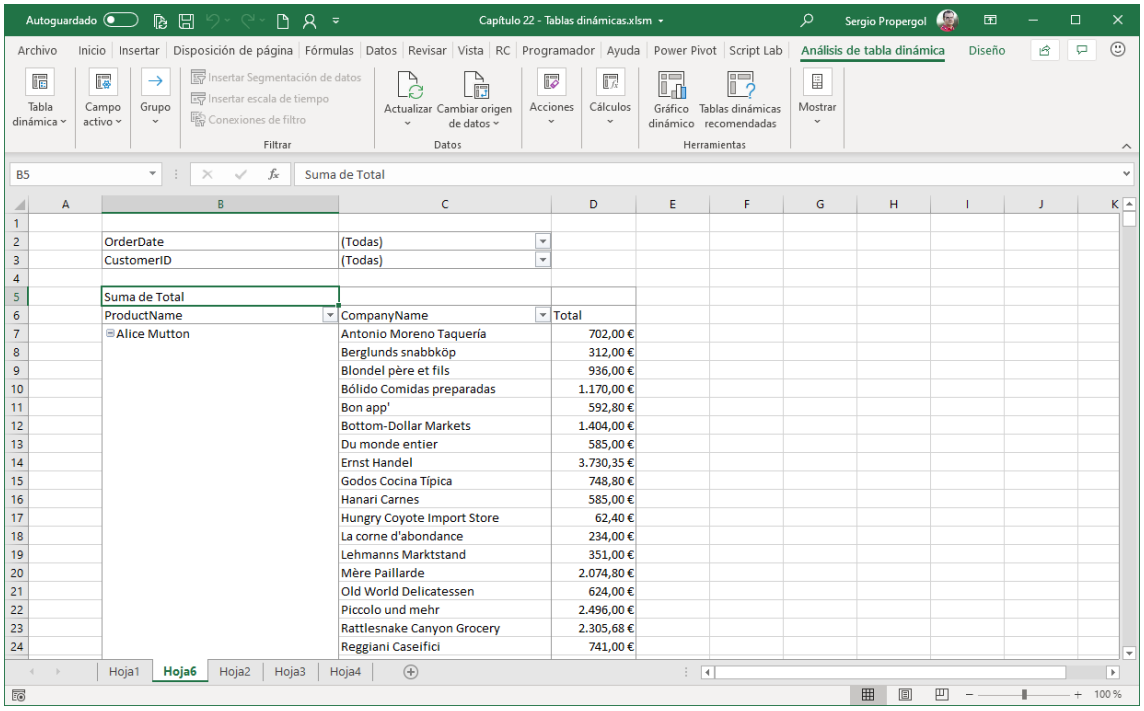


Imagen 4.1 Una tabla dinámica creada con VBA a partir de una base de datos Microsoft Access.

5 El objeto CreatePivotTable del objeto PivotCache

Cuando utilizamos la grabadora de macros para generar el código de la tabla dinámica, Excel usa el método **Add** de la colección **PivotCaches** para crear un nuevo objeto **PivotCache**. **PivotCache** representa los datos de la tabla dinámica. Se trata de un área en la memoria donde se almacenan los datos y se accede a ellos desde la fuente de datos cuando el procedimiento lo requiere. Utilizamos **PivotCache** cuando necesitamos generar múltiples

tablas dinámicas a partir de la misma fuente de datos. Usar el objeto `PivotCache` nos proporciona más control sobre las fuentes de datos externas. Puede también usarse para cambiar y refrescar los datos almacenados en la caché.

El siguiente ejemplo se conecta a la base de datos Northwind 2007.accdb utilizando el proveedor Microsoft.ACE.OLEDB.12.0. Para usar este tipo de conexión debemos habilitar la referencia a Microsoft ActiveX Data Objects (ADO) en el cuadro de diálogo **Referencias**, del menú **Herramientas** en el editor de VBA.

1. En el editor de VBA haz clic en el menú **Herramientas – Referencias**. Selecciona Microsoft ActiveX Data Objects 6.1 Library en la lista de referencias y haz clic en **Aceptar**.
2. Agrega un módulo nuevo al proyecto. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub TDExterna2()
    Dim objPivotCache As PivotCache
    Dim conn As New ADODB.Connection
    Dim rst As New ADODB.Recordset
    Dim dbPath As String
    Dim strSQL As String

    dbPath = "C:\Archivos Manual VBA\Northwind 2007.accdb"
    conn.Open "Provider=Microsoft.ACE.OLEDB.12.0;" _
    & "Data Source=" & dbPath & _
    "; Persist Security Info=False;"
    strSQL = "SELECT Products.[Product Name], " & _
    "Orders.[Order Date], " & _
    "Sum([Unit Price]*[Quantity]) AS Amount " & _
    "FROM Orders INNER JOIN (Products INNER JOIN " & _
    "[Order Details] ON Products.ID = " & _
    "[Order Details].[Product ID]) ON " & _
    "Orders.[Order ID] = [Order Details].[Order ID] " & _
    "GROUP BY Products.[Product Name], " & _
    "Orders.[Order Date], Products.[Product Name]" & _
    "ORDER BY Sum([Unit Price]*[Quantity]) DESC , " & _
    "Products.[Product Name];"
    Set rst = conn.Execute(strSQL)
    ' Crea la caché de la tabla dinámica y el informe
    Set objPivotCache = ActiveWorkbook.PivotCaches.Add( _
    SourceType:=xlExternal)
    Set objPivotCache.Recordset = rst
    Worksheets.Add
```

```
    With objPivotCache
        .CreatePivotTable TableDestination:=Range("B6"), _
        TableName:="Facturacion"
    End With
    ' Agrega los campos a la tabla dinámica
    With ActiveSheet.PivotTables("Facturacion")
        .SmallGrid = False
        With .PivotFields("Product Name")
            .Orientation = xlRowField
            .Position = 1
        End With
        With .PivotFields("Order Date")
            .Orientation = xlRowField
            .Position = 2
            .Name = "Date"
        End With
        With .PivotFields("Amount")
            .Orientation = xlDataField
            .Position = 1
            .NumberFormat = "#,##0.00 €"
        End With
    End With
    ' Ajusta el ancho de las columnas
    ActiveSheet.UsedRange.Columns.AutoFit
    ' Cierra conexiones y vacía variables
    rst.Close
    conn.Close
    Set rst = Nothing
    Set conn = Nothing
    ' Obtiene información sobre la caché
    With ActiveSheet.PivotTables("Facturacion").PivotCache
        Debug.Print "Información sobre la caché de la TD"
        Debug.Print "Número de registros: " & .RecordCount
        Debug.Print "Los datos se actualizaron el: " & _
        .RefreshDate
        Debug.Print "Datos actualizados por: " & .RefreshName
        Debug.Print "Memoria usada por la caché: " & _
        .MemoryUsed & " (bytes)"
    End With
End Sub
```

Después de establecer la conexión con la base de datos y ejecutar la declaración SQL para obtener los datos, el procedimiento crea una caché usando la siguiente línea de código:

```
Set objPivotCache = _
ActiveWorkbook.PivotCaches.Add(SourceType:=xlExternal)
```

En ese momento, el código copia los datos de la fuente de datos en la caché asignando un objeto **Recordset** al objeto **PivotCache**, de esta forma:

```
Set objPivotCache.Recordset = rst
```

A continuación, el código usa el método **CreatePivotTable** del objeto **PivotCache** para crear la tabla dinámica vacía:

```
With objPivotCache
    .CreatePivotTable TableDestination:=Range("B6"), _
    TableName:="Facturacion"
End With
```

Una vez se ha creado el esqueleto de la tabla dinámica, el código añade los campos adecuados. Las últimas líneas del procedimiento muestran información sobre la caché.

Para forzar a refrescar automáticamente la caché de un libro que contiene una tabla dinámica al abrirse, establece la propiedad **RefreshOnFileOpen** en **True**. Para hacer esto debes añadir al procedimiento anterior la siguiente declaración al final del código:

```
ActiveSheet.PivotTables("Invoices").PivotCache. _
RefreshOnFileOpen = True
```

3. Ejecuta el procedimiento TDExterna2 para generar la tabla dinámica.
   El resultado se muestra en la Imagen 5.1.

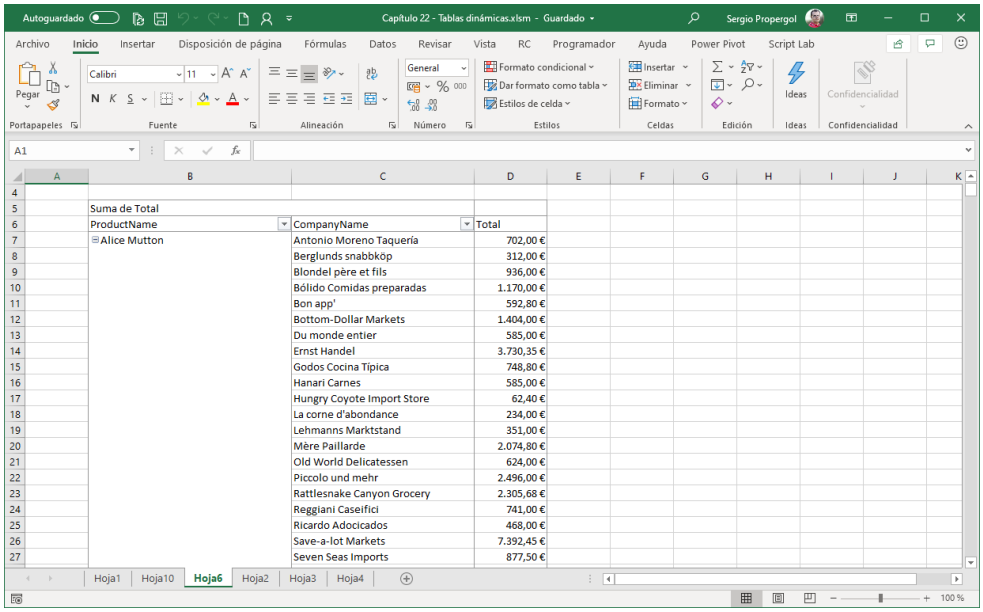


Imagen 5.1 Una tabla dinámica creada usando el método CreatePivotTabla del objeto PivotCache.

6 Dar formato, agrupar y ordenar una tabla dinámica

Podemos modificar la visualización y el formato de una tabla dinámica mediante programación utilizando un gran número de propiedades del objeto **PivotTable**. Por ejemplo, puede que queramos reubicar los campos en otro área, ordenar los datos por un campo específico, o agrupar los datos por años, trimestres, meses etc. En el siguiente ejemplo, se da formato a la tabla dinámica de la Imagen 4.1 para que se asemeje a la tabla de la Imagen 6.1.

1. Inserta un módulo nuevo en el proyecto e introduce el siguiente código:

```
Sub FormatoTablaDinamica()
    Dim pvtTable As PivotTable
    Dim strPiv As String

    If ActiveSheet.PivotTables.Count > 0 Then
        strPiv = ActiveSheet.PivotTables(1).Name
        Set pvtTable = ActiveSheet.PivotTables(strPiv)
    Else
        Exit Sub
    End If
    With pvtTable
        .PivotFields("OrderDate").Orientation = xlRows
        .PivotFields("CompanyName").Orientation = xlHidden
        ' Usa esta declaración para agrupar por año
        .PivotFields("OrderDate").DataRange.Cells(1).Group _
        Start:=True, End:=True, _
        periods:=Array(False, False, False, False, False, _
        False, True)
        ' Usa esta declaración para agrupar por trimestre y año
        ' .PivotFields("OrderDate").DataRange.Cells(1).Group _
        Start:=True, End:=True, _
        periods:=Array(False, False, False, False, _
        False, True, True)
        .PivotFields("OrderDate").Orientation = xlColumns
        .TableRange1.AutoFormat Format:=xlRangeAutoFormatColor2
        .PivotFields("ProductName").DataRange.Select
        ' Ordena el campo Product Name de forma descendente
        ' en función de la Suma Total
        .PivotFields("ProductName").AutoSort xlDescending, _
        "Sum of Total """
        Selection.IndentLevel = 2
        With Selection.Font
            .Name = "Times New Roman"
```

```
            .FontStyle = "Bold"
            .Size = 10
        End With
        With Selection.Borders(xlInsideHorizontal)
            .LineStyle = xlContinuous
            .Weight = xlThin
            .ColorIndex = xlAutomatic
        End With
    End With
End Sub
```

Mirando con detenimiento el código del procedimiento, puedes concluir fácilmente que:

- Para cambiar la disposición de la tabla dinámica debes utilizar la propiedad `Orientation` del campo requerido para establecer una constante diferente. En el ejemplo anterior se movió el campo **OrderDate** del área de filtros al área de filas.
- Para mostrar una tabla dinámica sin que aparezca un determinado campo, debes establecer su propiedad `Orientation` a `xlHidden`.
- Para agrupar por año el campo **OrderDate**, debes usar el método `Group` del objeto `Range`. Por ejemplo, el código usa la siguiente declaración para agrupar los datos por año:

```
PivotFields("OrderDate").DataRange. _
Cells(1).Group Start:=True, _
End:=True, periods:=Array(False, False, _
False, False, False, False, True)
```

  - Los argumentos `Start` y `End` especifican la fecha de comienzo y final, que son incluidos en el agrupamiento. Si estableces estos argumentos como `True`, se incluirán todas las fechas. El argumento `periods` es una matriz de valores booleanos que especifican los periodos de agrupamiento, como se muestra en la siguiente tabla:

Elemento	Periodo que representa
1	Segundos
2	Minutos
3	Horas
4	Días
5	Meses
6	Trimestres
7	Años

- Puedes aplicar automáticamente un formato a la tabla dinámica usando la propiedad **AutoFormat** del objeto **Range**. La propiedad **TableRange1** devuelve un objeto **Range** que representa el rango donde se encuentra la tabla dinámica sin los campos de filtro:

```
.TableRange1.AutoFormat Format:=xlRangeAutoFormatColor2
```

- Puedes seleccionar los elementos de un campo particular usando la propiedad **DataRange** y el método **Select**:

```
.PivotFields("ProductName").DataRange.Select
```

- Puedes ordenar un campo en particular de forma ascendente o descendente. El procedimiento anterior usa la siguiente declaración para ordenar el campo **ProductName** descendentemente en función de la suma total.

```
.PivotFields("ProductName"). _
AutoSort xlDescending, "Sum of Total"
```

- Puedes cambiar la sangría del texto, el tipo de fuente, el tamaño y el estilo del rango seleccionado, como se muestra en las últimas instrucciones del procedimiento anterior.

2. Activa la ventana de Excel y activa la hoja que contiene la tabla dinámica mostrada en la Imagen 4.1.
3. Presiona **Alt + F8** para abrir el cuadro de diálogo **Macros**. Selecciona **FormateoTablaDinamica** y haz clic en **Ejecutar**.
   El resultado se muestra en la Imagen 6.1.

	A	B	C	D	E	F	G	H
1								
2								
3		CustomerID	(Todas)					
4								
5		Suma de Total	OrderDate					
6		ProductName	1996	1997	1998	Total general		
7		Alice Mutton	6.962,28 €	17.604,60 €	8.131,50 €	32.698,38 €		
8		Aniseed Syrup	240,00 €	1.724,00 €	1.080,00 €	3.044,00 €		
9		Boston Crab Meat	2.778,30 €	9.814,73 €	5.317,60 €	17.910,63 €		
10		Camembert Pierrot	9.024,96 €	20.505,40 €	17.295,12 €	46.825,48 €		
11		Carnarvon Tigers	4.725,00 €	15.950,00 €	8.496,87 €	29.171,87 €		
12		Chai	1.605,60 €	4.887,00 €	6.295,50 €	12.788,10 €		
13		Chang	3.017,96 €	7.038,55 €	6.299,45 €	16.355,96 €		
14		Chartreuse verte	3.558,24 €	4.475,70 €	4.260,60 €	12.294,54 €		
15		Chef Anton's Cajun Seasoning	1.851,52 €	5.214,88 €	1.501,50 €	8.567,90 €		
16		Chef Anton's Gumbo Mix	1.931,20 €	373,62 €	3.042,38 €	5.347,20 €		
17		Chocolade		1.282,01 €	86,70 €	1.368,71 €		
18		Côte de Blaye	24.874,40 €	49.198,08 €	67.324,25 €	141.396,73 €		
19		Escargots de Bourgogne	1.378,00 €	2.076,27 €	2.427,40 €	5.881,67 €		
20		Filo Mix	246,40 €	2.124,15 €	862,40 €	3.232,95 €		
21		Flotemysost	4.248,40 €	8.438,75 €	6.863,87 €	19.551,02 €		
22		Geitost	385,50 €	786,00 €	476,62 €	1.648,12 €		
23		Genen Shouyu	310,00 €	1.474,82 €		1.784,82 €		
24		Gnocchi di nonna Alice	2.763,36 €	32.604,00 €	7.225,70 €	42.593,06 €		

Hoja1 | Hoja10 | Hoja6 | Hoja2 | Hoja3 | Hoja4 ...

Imagen 6.1 Otra vista de los datos tras el formateo efectuado mediante una macro.

7 Ocultar elementos en una tabla dinámica

En el ejemplo anterior agrupamos los datos de la tabla dinámica por año en función del campo OrderDate. Para ocultar datos agrupados, podemos establecer la propiedad **Visible** del objeto **PivotItem** como **False**. El siguiente procedimiento muestra cómo ocultar la columna del año 1996 de la tabla dinámica anterior:

```
Sub Ocultar1996()
    Dim myPivot As PivotTable
    Dim myItem As PivotItem
    Dim strFieldLabel As String

    strFieldLabel = "1996"
    Set myPivot = ActiveSheet.PivotTables(1)
    For Each myItem In myPivot.PivotFields _
    ("OrderDate").PivotItems
        If myItem.Name <> strFieldLabel Then
            myItem.Visible = True
        Else
            myItem.Visible = False
        End If
    Next
End Sub
```

8 Agregar campos y elementos calculados a una tabla dinámica

Podemos personalizar una tabla dinámica definiendo campos y elementos calculados. Usando el contenido de otros campos numéricos de la tabla dinámica, podemos crear un campo calculado que realice los cálculos requeridos. Por ejemplo, vamos a crear un procedimiento con dos campos calculados (Change:2010/2009 y Change: 2009/2008) para calcular la diferencia en número de productos vendidos de un año a otro.

1. Guarda el libro anterior y ciérralo.
2. Crea un nuevo libro de trabajo y llámalo Capítulo 22b – Tablas dinámicas xlsm. Guárdalo en la carpeta C:\Archivos Manual VBA.
3. En el nuevo libro introduce los datos que se muestran en la Imagen 8.1.

	A	B	C	D
1	Producto	2016	2017	2018
2	Prod1	904	614	694
3	Prod2	456	139	755
4	Prod3	1522	1009	1002
5				
6				

Imagen 8.1 Datos de ejemplo para crear una tabla dinámica.

4. Activa el editor de VBA y haz clic en el nombre del proyecto.
5. Haz clic en el menú **Insertar – Módulo** para insertar un módulo nuevo. A continuación, introduce el siguiente procedimiento:

```
Sub CamposCalculados()
    ActiveWorkbook.PivotCaches.Add( _
    SourceType:=xlDatabase, _
    SourceData:="Hoja1!R1C1:R4C4").CreatePivotTable _
    TableDestination:= _
    "'[Capítulo 22b - Tablas dinámicas.xlsm]Hoja1'!R4C7", _
    TableName:="Piv1", _
    DefaultVersion:=xlPivotTableVersion10
    With ActiveSheet.PivotTables("Piv1").PivotFields("Producto")
    .Orientation = xlRowField
    .Position = 1
    End With
    ActiveSheet.PivotTables("Piv1").AddDataField _
    ActiveSheet.PivotTables("Piv1").PivotFields("2018"), _
    "Sum of 2018", xlSum
    ActiveSheet.PivotTables("Piv1").AddDataField _
    ActiveSheet.PivotTables("Piv1").PivotFields("2017"), _
    "Sum of 2017", xlSum
    ActiveSheet.PivotTables("Piv1").AddDataField _
    ActiveSheet.PivotTables("Piv1").PivotFields("2016"), _
    "Sum of 2016", xlSum
    ActiveSheet.PivotTables("Piv1").CalculatedFields.Add _
    "Change: 2018/2017", "='2018' - '2017'", True
    ActiveSheet.PivotTables("Piv1").CalculatedFields.Add _
    "Change: 2017/2016", "='2017' - '2016'", True
    ActiveSheet.PivotTables("Piv1"). _
    PivotFields("Change: 2018/2017"). _
    Orientation = xlDataField
    ActiveSheet.PivotTables("Piv1"). _
    PivotFields("Change: 2017/2016"). _
    Orientation = xlDataField
End Sub
```

Los campos calculados se definen utilizando el método `Add` del objeto `CalculatedFields`, proporcionando el nombre del campo y su fórmula:

```
    ActiveSheet.PivotTables("Piv1").CalculatedFields.Add _
    "Change: 2018/2017", "='2018' - '2017'", True
    ActiveSheet.PivotTables("Piv1").CalculatedFields.Add _
```

```
"Change: 2017/2016", "='2017' - '2016'", True
```

Un campo calculado utiliza una fórmula que se refiere a otros campos de la tabla que contienen datos numéricos. Puede ser desde una simple fórmula, como sumas o restas, hasta fórmulas más complejas y funciones. En el procedimiento hemos creado dos campos calculados que se muestran a continuación:

Nombre del campo calculado	Fórmula utilizada
Change: 2018/2017	=’2018’ - ‘2017’
Change: 2017/2016	=’2017’ - ‘2016’

6. Ejecuta el procedimiento **CamposCalculados**.
   La tabla dinámica resultante se muestra en la Imagen 8.2.

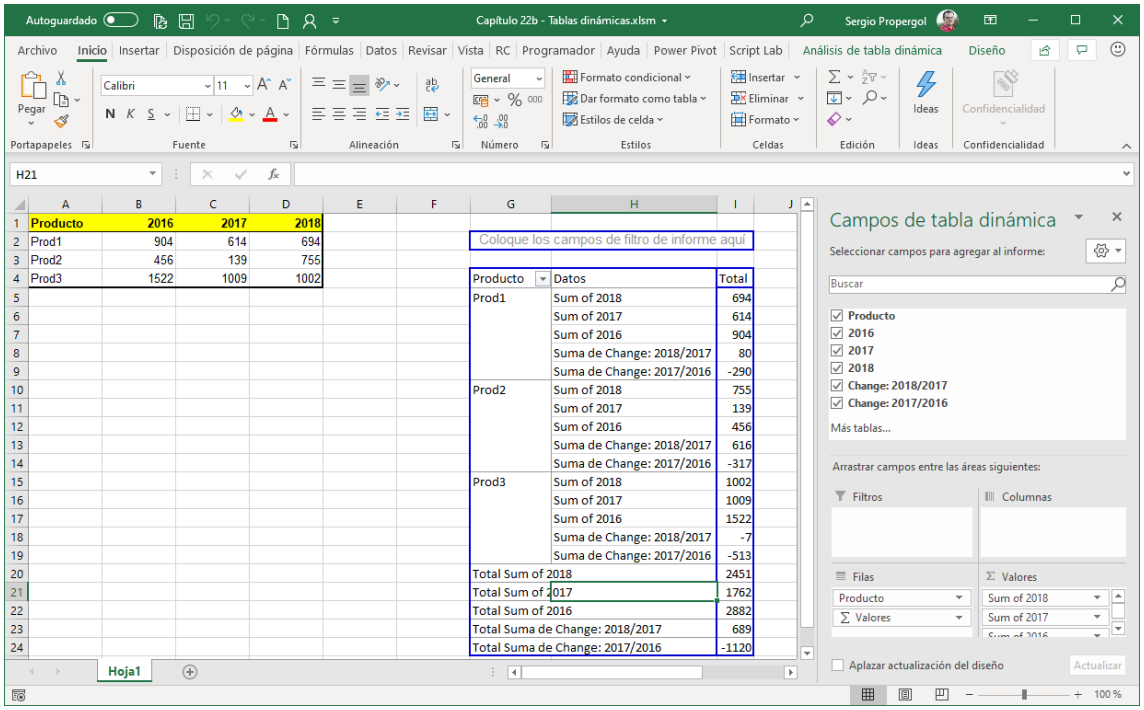


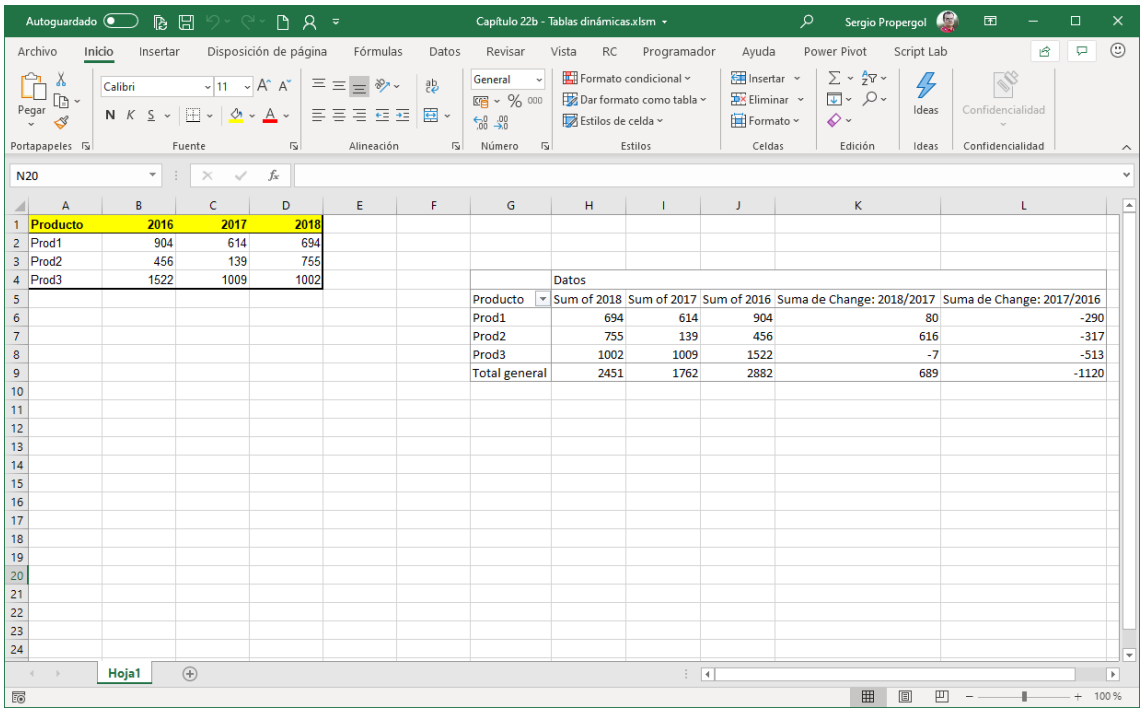
Imagen 8.2 Podemos crear cálculos adicionales en la tabla dinámica definiendo campos calculados.

Si añadimos la siguiente declaración al final del procedimiento, eliminamos las columnas G:I en la hoja y volvemos a ejecutar el procedimiento, la tabla dinámica cambia la disposición de los datos:

```
ActiveSheet.PivotTables("Piv1"). _
PivotFields("Data").Orientation = xlColumnField
```

7. Guarda el libro y ciérralo.

No debemos confundir un campo calculado con un elemento calculado. Un elemento calculado es un elemento personalizado que sirve para realizar cálculos usando los contenidos de otros campos y elementos de la tabla dinámica.



Producto	2016	2017	2018
Prod1	904	614	694
Prod2	456	139	755
Prod3	1522	1009	1002

	Datos				
Producto	Sum of 2018	Sum of 2017	Sum of 2016	Suma de Change: 2018/2017	Suma de Change: 2017/2016
Prod1	694	614	904	80	-290
Prod2	755	139	456	616	-317
Prod3	1002	1009	1522	-7	-513
Total general	2451	1762	2882	689	-1120

Imagen 8.3 Modificando la orientación de la tabla dinámica.

Por ejemplo, digamos que nos piden crear un informe que muestre las ventas totales de productos por cada uno de los vendedores y por cada país. Queremos hacerlo de forma diferente y vamos a mostrar las ventas realizadas por cada vendedor en tres continentes. Necesitaremos tres nuevos elementos (calculados) en el campo País. Estos elementos se llamarán América del Norte, América del Sur y Europa. Después de crear estos elementos podemos cambiar el nombre del campo País a Continente, como en la Imagen 8.4, para que los datos sean más fáciles de leer. El siguiente procedimiento obtiene los datos para este ejemplo de la base de datos Northwind. El código de este procedimiento fue generado por la grabadora de macros:

1. Crea un nuevo libro y llámalo Capítulo 22c – Tablas dinámicas.xlsm.
   Activa el editor de VBA e inserta un módulo nuevo.
2. Introduce el siguiente procedimiento:

```
Sub ElementosCalculados()
    Dim strConn As String
    Dim strSQL As String
    Dim myArray As Variant
    Dim destRng As Range
    Dim strPivot As String

    strConn = "Driver={Microsoft Access Driver (*.mdb)};" & _
    "DBQ=" & "C:\Archivos Manual VBA\" & _
    "Northwind.mdb;"
```

```
    strSQL = "SELECT Invoices.Customers.CompanyName, " & _
    "Invoices.Country, Invoices.Salesperson, " & _
    "Invoices.ProductName, Invoices.ExtendedPrice " & _
    "FROM Invoices ORDER BY Invoices.Country"
    myArray = Array(strConn, strSQL)
    Worksheets.Add
    Set destRng = ActiveSheet.Range("B5")
    strPivot = "PivotTable1"
    ActiveSheet.PivotTableWizard _
    SourceType:=xlExternal, _
    SourceData:=myArray, _
    TableDestination:=destRng, _
    TableName:=strPivot, _
    SaveData:=False, _
    BackgroundQuery:=False
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("CompanyName")
        .Orientation = xlPageField
        .Position = 1
    End With
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("Country")
        .Orientation = xlRowField
        .Position = 1
    End With
    ActiveSheet.PivotTables(strPivot).AddDataField _
ActiveSheet.PivotTables(strPivot).PivotFields("ExtendedPrice"),
_
    "Sum of ExtendedPrice", xlSum
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("Salesperson")
        .Orientation = xlRowField
        .Position = 1
    End With
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("Salesperson")
        .Orientation = xlPageField
        .Position = 1
    End With
    With _
```

```
ActiveSheet.PivotTables(strPivot).PivotFields("Salesperson")
    .Orientation = xlColumnField
    .Position = 1
End With
ActiveSheet.PivotTables(strPivot).PivotFields("Country"). _
CalculatedItems.Add "América del Norte", "=USA+Canada", True
ActiveSheet.PivotTables(strPivot).PivotFields("Country"). _
CalculatedItems.Add "América del Sur", _
"=Argentina+Brazil+Venezuela ", True
ActiveSheet.PivotTables(strPivot).PivotFields("Country"). _
CalculatedItems("América del Norte").StandardFormula = _
"=USA+Canada+Mexico"
ActiveSheet.PivotTables(strPivot).PivotFields("Country"). _
CalculatedItems.Add "Europa", _
"=Austria+Belgium+Denmark+Finland+" & _
"France+Germany+Ireland+Italy+Norway+Poland+" & _
"Portugal+Spain+Sweden+Switzerland+UK", True
With _
ActiveSheet.PivotTables(strPivot).PivotFields("Country")
    .PivotItems("Argentina").Visible = False
    .PivotItems("Austria").Visible = False
    .PivotItems("Belgium").Visible = False
    .PivotItems("Brazil").Visible = False
    .PivotItems("Canada").Visible = False
    .PivotItems("Denmark").Visible = False
    .PivotItems("Finland").Visible = False
    .PivotItems("France").Visible = False
    .PivotItems("Germany").Visible = False
    .PivotItems("Ireland").Visible = False
    .PivotItems("Italy").Visible = False
    .PivotItems("Mexico").Visible = False
    .PivotItems("Norway").Visible = False
    .PivotItems("Poland").Visible = False
    .PivotItems("Portugal").Visible = False
    .PivotItems("Spain").Visible = False
    .PivotItems("Sweden").Visible = False
    .PivotItems("Switzerland").Visible = False
    .PivotItems("UK").Visible = False
    .PivotItems("USA").Visible = False
    .PivotItems("Venezuela").Visible = False
```

```
        End With
        ActiveSheet.PivotTables(strPivot). _
        PivotFields("Country").Caption = "Continente"
        With ActiveSheet.PivotTables(strPivot). _
            PivotFields("Sum of ExtendedPrice"). _
            NumberFormat = "#,##0.00 €"
        End With
        With _
        ActiveSheet.PivotTables(strPivot).PivotFields("ProductName")
            .Orientation = xlRowField
            .Position = 2
        End With
        ActiveSheet.PivotTables(strPivot). _
        PivotFields("ProductName").Orientation = xlHidden
    End Sub
```

Un elemento calculado usa una fórmula que se refiere a otros elementos de la tabla dinámica. Por ejemplo, una tabla dinámica que contiene un campo País, que muestra diferentes elementos de países (Austria Argentina, etc.) podría tener un elemento calculado que sume todos sus valores:

Elemento Calculado	Fórmula
América del sur	=Argentina+Brazil+Venezuela

Todos los elementos calculados de una tabla dinámica son miembros de la colección `CalculatedItems`. Los elementos calculados se definen con el método `Add` del objeto `CalculatedItems` y proporcionando dos argumentos: el nombre del elemento y su fórmula:

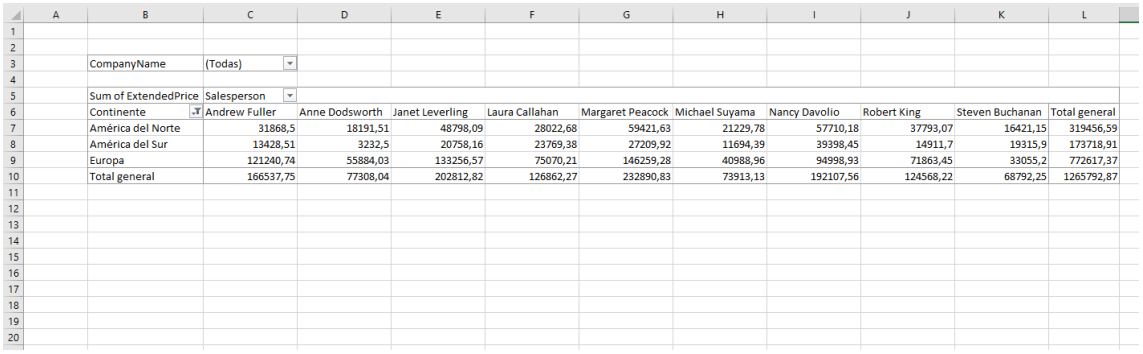
```
ActiveSheet.PivotTables(strPivot).PivotFields("Country"). _
CalculatedItems.Add "América del sur", _
"=Argentina+Brazil+Venezuela", True
```

3. Ejecuta el procedimiento `ElementosCalculados`.
   El resultado se muestra en la Imagen 8.4

Podemos modificar el procedimiento definiendo nuevos elementos calculados en el campo **SalesPerson** para generar otro informe. Por ejemplo, si añadimos el siguiente código después de las líneas:

```
ActiveSheet.PivotTables(strPivot). _
PivotFields("Country").Caption = "Continent"
```

En el procedimiento anterior, obtendremos el resultado de la Imagen 9.1.



CompanyName	(Todas)										
Sum of ExtendedPrice	Salesperson										
Continente	Andrew Fuller	Anne Dodsworth	Janet Leverling	Laura Callahan	Margaret Peacock	Michael Suyama	Nancy Davolio	Robert King	Steven Buchanan	Total general
América del Norte	31868,5	18191,51	48798,09	28022,68	59421,63	21229,78	57710,18	37793,07	16421,15	319456,59
América del Sur	13428,51	3232,5	20758,16	23769,38	27209,92	11694,39	39398,45	14911,7	19315,9	173718,91
Europa	121240,74	55884,03	133256,57	75070,21	146259,28	40988,96	94998,93	71863,45	33055,2	772617,37
Total general	166537,75	77308,04	202812,82	126862,27	232890,83	73913,13	192107,56	124568,22	68792,25	1265792,87

Imagen 8.4 Mediante elementos calculados podemos presentar la información resumida de acuerdo a nuestras necesidades.

```
ActiveSheet.PivotTables(strPivot).PivotFields("Salesperson"). _
    CalculatedItems.Add "Hombres", _
    "=Michael Suyama+Andrew Fuller+Robert King+" & _
    "Steven Buchanan", True

ActiveSheet.PivotTables(strPivot).PivotFields("Salesperson"). _
    CalculatedItems.Add "Mujeres", _
    "=Anne Dodsworth+Laura Callahan+Janet Leverling+" & _
    "Margaret Peacock+Nancy Davolio", True
    With ActiveSheet.PivotTables("PivotTable1"). _
    PivotFields("Salesperson")
        .PivotItems("Andrew Fuller").Visible = False
        .PivotItems("Anne Dodsworth").Visible = False
        .PivotItems("Janet Leverling").Visible = False
        .PivotItems("Laura Callahan").Visible = False
        .PivotItems("Margaret Peacock").Visible = False
        .PivotItems("Michael Suyama").Visible = False
        .PivotItems("Nancy Davolio").Visible = False
        .PivotItems("Robert King").Visible = False
        .PivotItems("Steven Buchanan").Visible = False
    End With
```

9 Crear un gráfico dinámico

Un gráfico dinámico es la representación visual de una tabla dinámica. Con VBA podemos crear un gráfico dinámico en función de una tabla dinámica existente, y cambiar la disposición de los datos de la misma forma que con las tablas dinámicas.

Un gráfico dinámico siempre está enlazado a una tabla dinámica. Esto significa que cuando reestructuras los datos en la tabla dinámica, el gráfico dinámico muestra la misma visualización

de los datos y viceversa. El tipo de gráfico predeterminado es el de barras apiladas. Este tipo de gráficos es muy útil para comparar la contribución de cada valor al total de las categorías. Podemos generar cualquier tipo de gráfico excepto gráficos XY (dispersión), de cotizaciones o de burbujas.

	A	B	C	D	E	F
1						
2						
3		CompanyName	(Todas)			
4						
5		Sum of ExtendedPrice	Salesperson			
6		Continente	Hombres	Mujeres	Total general	
7		América del Norte	107312,5	212144,09	319456,59	
8		América del Sur	59350,5	114368,41	173718,91	
9		Europa	267148,35	505469,02	772617,37	
10		Total general	433811,35	831981,52	1265792,87	
11						
12						
13						
14						
15						
16						
17						

Imagen 9.1 Podemos resumir todavía más la tabla, introduciendo nuevos elementos calculados.

De forma manual podemos crear un gráfico dinámico desde la ficha **Insertar > Gráfico dinámico**. Desde VBA, la creación requiere usar el método `SetDataSource` del objeto `PivotChart` y especificar una referencia al rango de la tabla dinámica. El objeto `PivotTable` tiene las siguientes dos propiedades que representan parte o toda la tabla dinámica.

- `TableRange1`: Devuelve el rango que representa la tabla dinámica exceptuando los campos de Filtro.
- `TableRange2`. Devuelve el rango que representa la totalidad de la tabla dinámica.

El procedimiento del siguiente ejercicio genera una tabla dinámica desde la base de datos Northwind.mdb. Otro procedimiento del ejercicio configura un gráfico dinámico basado en la tabla dinámica anterior.

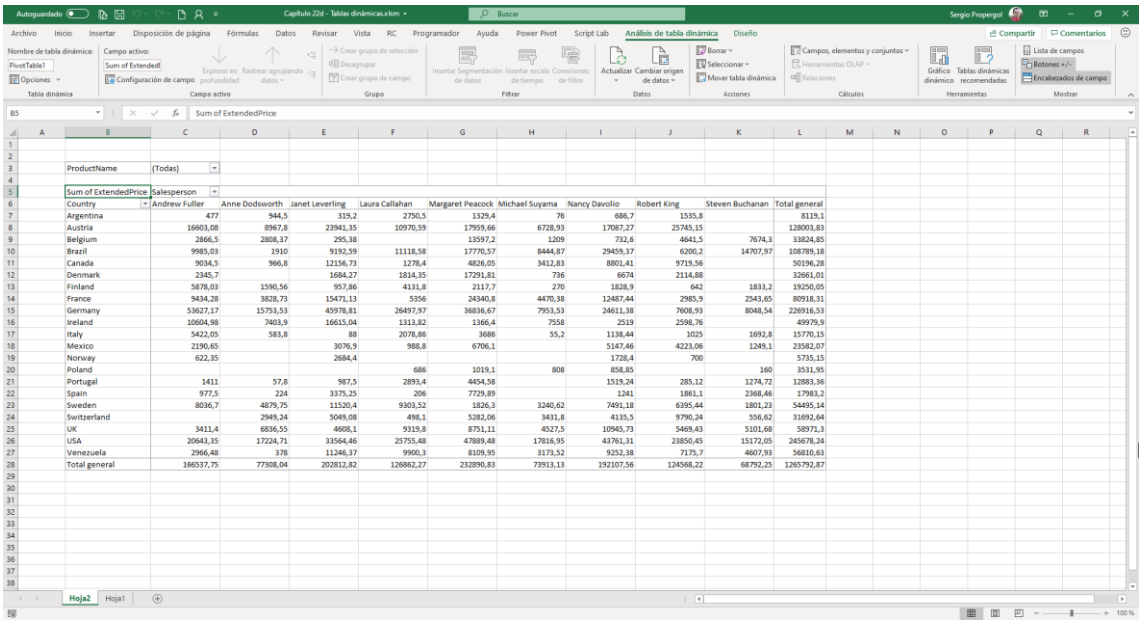
1. Guarda el libro Capítulo 22c – Tablas dinámicas xlsm.
2. Crea un nuevo libro y guárdalo como Capítulo 22d – Tablas dinámicas.xlsm.
3. Activa el editor de VBA y selecciona el proyecto en el **Explorador de proyectos**.
4. Inserta un módulo nuevo y agrega el siguiente procedimiento:

```
Sub GeneraGraficoDinamico()
    Dim strConn As String
    Dim strSQL As String
    Dim myArray As Variant
    Dim destRng As Range
```

```
    Dim strPivot As String

    strConn = "Driver={Microsoft Access Driver (*.mdb)};" & _
    "DBQ=" & "C:\Archivos Manual VBA\Northwind.mdb;"
    strSQL = "SELECT Invoices.Customers.CompanyName, " & _
    "Invoices.Country, Invoices.Salesperson, " & _
    "Invoices.ProductName, Invoices.ExtendedPrice " & _
    "FROM Invoices ORDER BY Invoices.Country"
    myArray = Array(strConn, strSQL)
    Worksheets.Add
    Set destRng = ActiveSheet.Range("B5")
    strPivot = "PivotTable1"
    ActiveSheet.PivotTableWizard _
    SourceType:=xlExternal, _
    SourceData:=myArray, _
    TableDestination:=destRng, _
    TableName:=strPivot, _
    SaveData:=False, _
    BackgroundQuery:=False
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("ProductName")
        .Orientation = xlPageField
        .Position = 1
    End With
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("Country")
        .Orientation = xlRowField
        .Position = 1
    End With
    With _
    ActiveSheet.PivotTables(strPivot).PivotFields("Salesperson")
        .Orientation = xlColumnField
        .Position = 1
    End With
    ActiveSheet.PivotTables(strPivot).AddDataField _
ActiveSheet.PivotTables(strPivot).PivotFields("ExtendedPrice"),
_
    "Sum of ExtendedPrice", xlSum
    With ActiveSheet.PivotTables(strPivot). _
```

```
        PivotFields("Sum of ExtendedPrice").NumberFormat =
"#,##0.00 €"
    End With
End Sub
```

5. Ejecuta el procedimiento.
   Excel agrega una nueva hoja con la tabla dinámica que se muestra en la Imagen 9.2.



ProductName	(Todas)										
Sum of ExtendedPrice	Salesperson										
Country	Andrew Fuller	Anne Dodsworth	Janet Leverling	Laura Callahan	Margaret Peacock	Michael Suyama	Nancy Davolio	Robert King	Steven Buchanan	Total general	
Argentina	477	944,5	319,2	2750,5	1329,4	76	686,7	1535,8		8119,1	
Austria	16603,08	8967,8	23941,35	10970,59	17959,66	6728,93	17087,27	25745,15		128003,83	
Belgium	2866,5	2808,37	295,38		13597,2	1209	732,6	4641,5	7674,3	33824,85	
Brazil	9985,03	1910	9192,59	11118,58	17770,57	8444,87	29459,37	6200,2	14707,97	108789,18	
Canada	9034,5	966,8	12156,73	1278,4	4826,05	3412,83	8801,41	9719,56		50196,28	
Denmark	2345,7		1684,27	1814,35	17291,81	736	6674	2114,88		32661,01	
Finland	5878,03	1590,56	957,86	4131,8	2117,7	270	1828,9	642	1833,2	19250,05	
France	9434,28	3828,73	15471,13	5356	24340,8	4470,38	12487,44	2985,9	2543,65	80918,31	
Germany	53627,17	15753,53	45978,81	26497,97	36836,67	7953,53	24611,38	7608,93	8048,54	226916,53	
Ireland	10604,98	7403,9	16615,04	1313,82	1366,4	7558	2519	2598,76		49979,9	
Italy	5422,05	383,8	88	2078,86	3686	55,2	1138,44	1025	1692,8	15770,15	
Mexico	2190,65		3076,9	988,8	6706,1		5147,46	4223,06	1249,1	23582,07	
Norway	622,35		2684,4				1728,4	700		5735,15	
Poland				686	1019,1	808	858,85		160	3531,95	
Portugal	1411	57,8	987,5	2893,4	4454,58		1519,24	285,12	1274,72	12883,36	
Spain	977,5	224	3375,25	206	7729,89		1241	1861,1	2368,46	17983,2	
Sweden	8036,7	4879,75	11520,4	9303,52	1826,3	3240,62	7491,18	6395,44	1801,23	54495,14	
Switzerland		2949,24	5049,08	498,1	5282,06	3431,8	4135,5	9790,24	556,62	31692,64	
UK	3411,4	6836,55	4608,1	9319,8	8751,11	4527,5	10945,73	5469,43	5101,68	58971,3	
USA	20643,35	17224,71	33564,46	25755,48	47889,48	17816,95	43761,31	23850,45	15172,05	245678,24	
Venezuela	2966,48	378	11246,37	9900,3	8109,95	3173,52	9252,38	7175,7	4607,93	56810,63	
Total general	166537,75	77308,04	202812,82	126862,27	232890,83	73913,13	192107,56	124568,22	68792,25	1265792,87	

Imagen 9.2 Esta tabla dinámica se usará en el gráfico dinámico.

6. En el mismo módulo donde introdujiste el procedimiento anterior, introduce este otro procedimiento:

```
Sub CrearGraficoDinamico()
Dim shp As Shape
Dim rngSource As Range
Dim pvtTable As PivotTable
Dim r As Integer

Set pvtTable = Worksheets("Sheet2").PivotTables(1)
' Cambia el nombre de la hoja donde se encuentra la
' tabla dinámica, llamándola "Tofu"
pvtTable.PivotFields("ProductName").CurrentPage = "Tofu"
Set rngSource = pvtTable.TableRange2
Set shp = ActiveSheet.Shapes.AddChart
shp.Chart.SetSourceData Source:=rngSource
shp.Chart.SetElement (msoElementChartTitleAboveChart)
shp.Chart.ChartTitle.Caption = _
pvtTable.PivotFields("ProductName").CurrentPage
r = ActiveSheet.UsedRange.Rows.Count + 3
```

```
        With Range("B" & r & ":E" & r + 15)
            shp.Width = .Width
            shp.Height = .Height
            shp.Left = .Left
            shp.Top = .Top
        End With
    End Sub
```

El procedimiento cambia la página actual de la tabla dinámica para mostrar información sobre el producto llamado Tofu. El método `AddChart` de la colección `Shapes` se usa para crear un objeto `Chart`. El método `SetSourceData` del objeto `Chart` se usa entonces para especificar el rango de la tabla dinámica como fuente de datos para el gráfico. Siempre es una buena idea añadir un título al gráfico, así que las siguientes dos líneas de código aseguran que el título se ubicará encima del área del gráfico y que en el texto aparecerá el nombre del producto.

```
shp.Chart.SetElement (msoElementChartTitleAboveChart)
shp.Chart.ChartTitle.Caption = _
pvtTable.PivotFields("ProductName").CurrentPage
```

Para asegurarte de que el gráfico aparece justo debajo de la tabla dinámica, calculamos el rango usado en la hoja activa y le añadimos tres filas. La propiedades `Top`, `Left`, `Width` y `Height` se usan para ubicar el gráfico sobre el rango especificado.

```
    r = ActiveSheet.UsedRange.Rows.Count + 3
        With Range("B" & r & ":E" & r + 15)
            shp.Width = .Width
            shp.Height = .Height
            shp.Left = .Left
            shp.Top = .Top
        End With
```

7. Ejecuta el procedimiento.
   El resultado aparece en la Imagen 9.3.

Para asegurarte de que el título del gráfico cambia cuando seleccionas un producto diferente en el filtro de la tabla dinámica, debes crear el procedimiento de evento `Worksheet_PivotTableUpdate` en el módulo de la Hoja2:

1. Haz doble clic en el objeto de la Hoja2 en el **Explorador de proyectos** e introduce el siguiente código:

```
Private Sub Worksheet_PivotTableUpdate(ByVal Target As
PivotTable)
    Dim strPivotPage As String
    Dim r As Integer
```

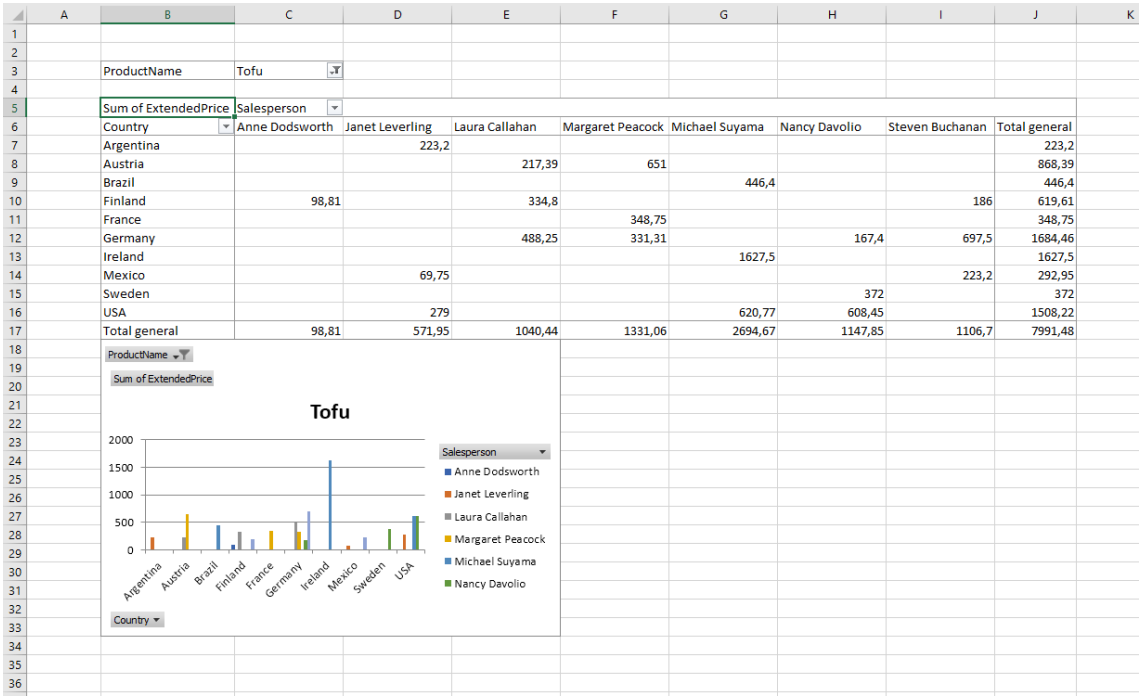


Imagen 9.3 El gráfico dinámico se genera a partir de una tabla dinámica incrustada en la misma hoja.

```
        strPivotPage = Target.PivotFields("ProductName"). _
        CurrentPage.Value
        If ActiveSheet.ChartObjects.Count > 0 Then
            ActiveSheet.ChartObjects(1).Activate
            ActiveChart.ChartTitle.Text = strPivotPage
            r = ActiveSheet.UsedRange.Rows.Count + 3
            With Range("B" & r)
                ActiveSheet.ChartObjects(1).Top = .Top
            End With
        End If
    End Sub
```

Este procedimiento se ejecutará automáticamente cuando se actualice la tabla dinámica.

2. En la Hoja2, selecciona otro producto desde el campo filtro.
   Observa cómo cambia el título del gráfico para ajustarse a la selección.
3. Guarda y cierra el libro Capítulo 22d – Tablas dinámicas.xlsm.

10 Modelo de datos y tablas dinámicas

La herramienta de Excel llamada **Modelo de datos** permite trabajar con fuentes de datos muy distintas de forma simultánea en tablas y gráficos dinámicos. Con el modelo de datos de Excel se pueden gestionar varias conexiones de datos, importar millones de filas de múltiples fuentes de datos y crear relaciones entre tablas. Cuando se utiliza el modelo de datos, los

datos no solo se procesan más rápidamente, sino que están muy comprimidos, por lo que no hay que preocuparse de manejar archivos de gran tamaño.

El siguiente ejercicio nos guiará por todos los pasos necesarios para crear un modelo de datos. En este ejemplo, relacionaremos tres tablas de la base de datos Northwind 2007 (Products, Ordes y Order Details) para analizar las ventas de productos por ciudad.

1. Crea un libro nuevo y llámalo C:\Capítulo 22 – Modelo de datos.xlsm.
2. Haz clic en la ficha **Insertar > Tabla dinámica**.
   Aparece el cuadro de diálogo **Crear tabla dinámica**.
3. En la sección superior del cuadro de diálogo haz clic en **Utilice una fuente de datos externa** y, a continuación, en el botón **Elegir conexión**.
4. En el cuadro de diálogo **Conexiones existentes**, haz clic en el botón **Examinar en busca de más…**
5. En el nuevo cuadro de diálogo que se abre, selecciona la base de datos ubicada en C:\Archivos Manual VBA\Northwind 2007accdb (ver Imagen 10.1).

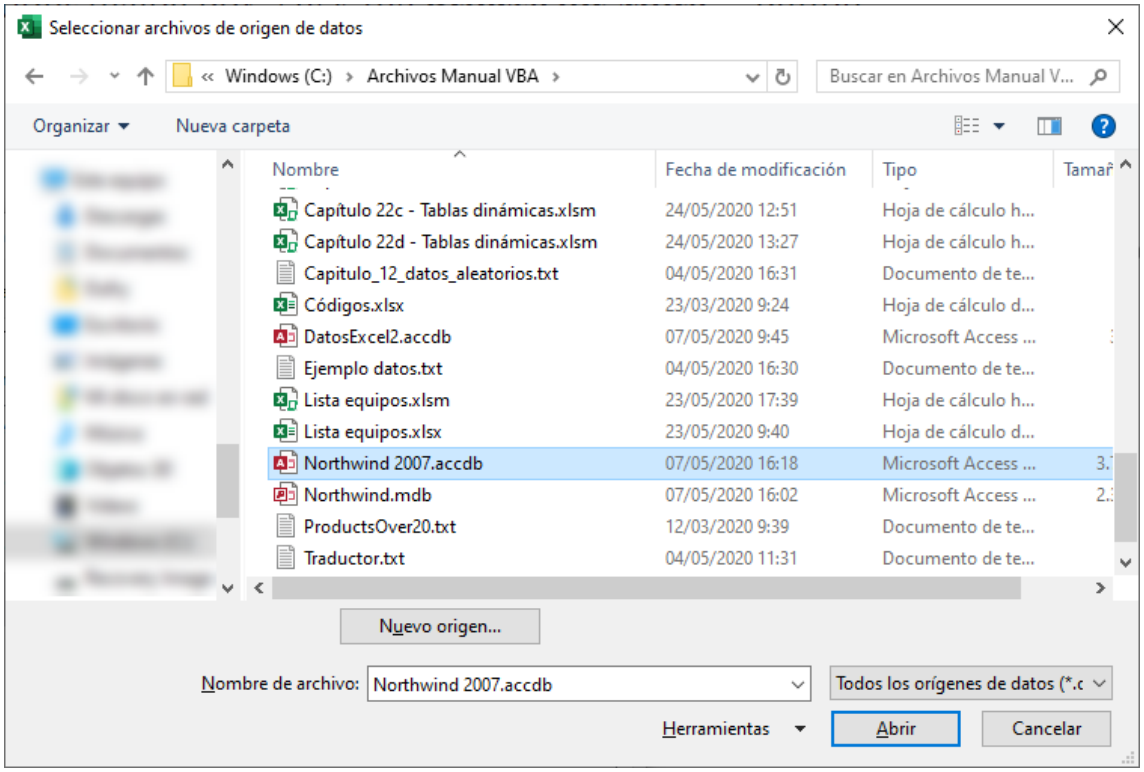


Imagen 10.1 Seleccionando una base de datos Access.

Excel mostrará el cuadro de diálogo **Seleccionar tabla**, que contiene todas las tablas de la base de datos.

6. Haz clic en la casilla de verificación **Activar selección de varias tablas** y selecciona Order Details, Orders y Products. A continuación haz clic en **Aceptar**.
   Ahora vemos que Excel ha establecido la conexión con la base de datos Northwind 2007 justo al lado del botón **Elegir conexión** y ha activado la casilla **Agregar estos datos al modelo de datos** (ver Imagen 10.2).

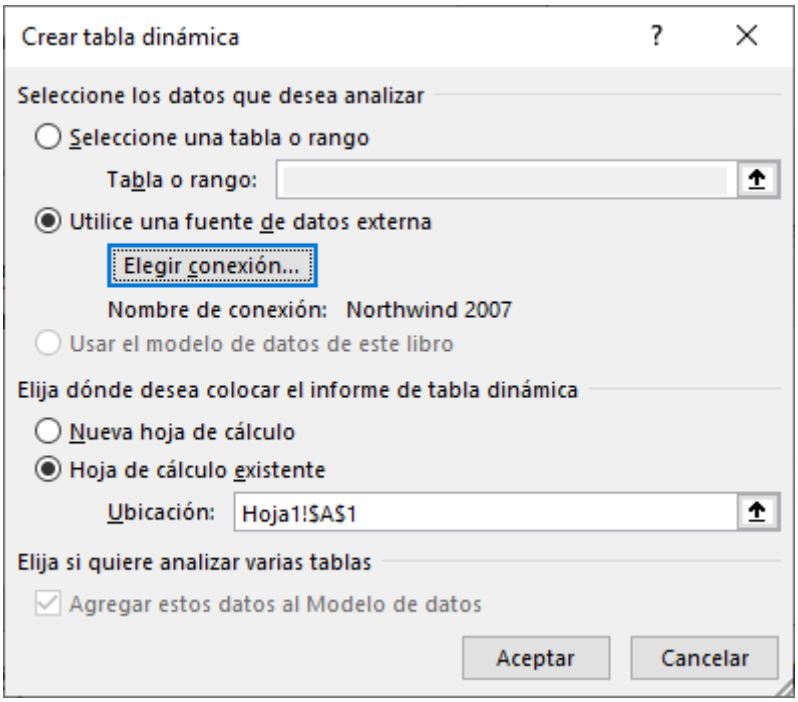


Imagen 10.2 Cuando se seleccionan varias tablas o consultas Excel activa automáticamente la casilla Agregar estos datos al modelo de datos.

7. Haz clic en **Aceptar** para crear la tabla dinámica.
   Excel muestra el mensaje "Cargando modelo de datos y obteniendo datos" en la barra de estado del libro. Al finalizar, debería aparecer una tabla dinámica en blanco. Ya puedes comenzar a construir la tabla dinámica basada en el modelo de datos que acabas de crear.
   Si Excel detecta relaciones entre las tablas seleccionadas, automáticamente las recrea en el modelo de datos cuando importas de una sola vez todas las tablas.
   Si Excel no logra determinar las relaciones entre tablas, necesitarás definirlas explícitamente antes de que Excel pueda manejar los datos del modelo de datos. Esto se hace desde el cuadro de diálogo **Administrar relaciones**, cuyo botón de acceso se encuentra en la ficha **Datos**. Vamos a crear las relaciones apropiadas entre las tablas.
8. Haz clic en el botón **Relaciones** de la ficha **Datos**.
9. En el cuadro de diálogo **Administrar relaciones**, haz clic en el botón **Nuevo**.
10. En el cuadro **Crear relación** introduce los datos como se muestra en la Imagen 10.3 para definir la relación entre las tablas OrderDetails y Products.

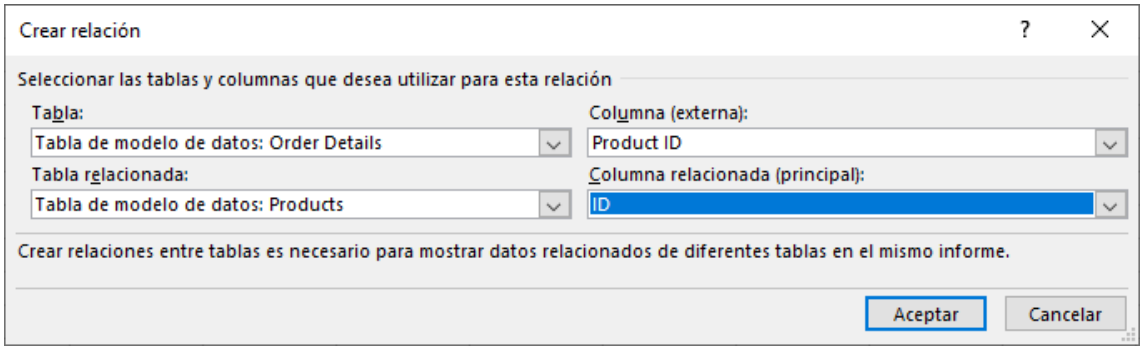


Imagen 10.3 Definiendo una relación entre dos tablas.

11. Haz clic en **Aceptar**.
    Excel muestra ahora la relación en el cuadro de diálogo.
12. Presiona el botón de nuevo para crear otra relación entre las tablas Orders y Order Details.
13. En el cuadro **Crear relación**, introduce los datos como se muestra en la Imagen 10.4.

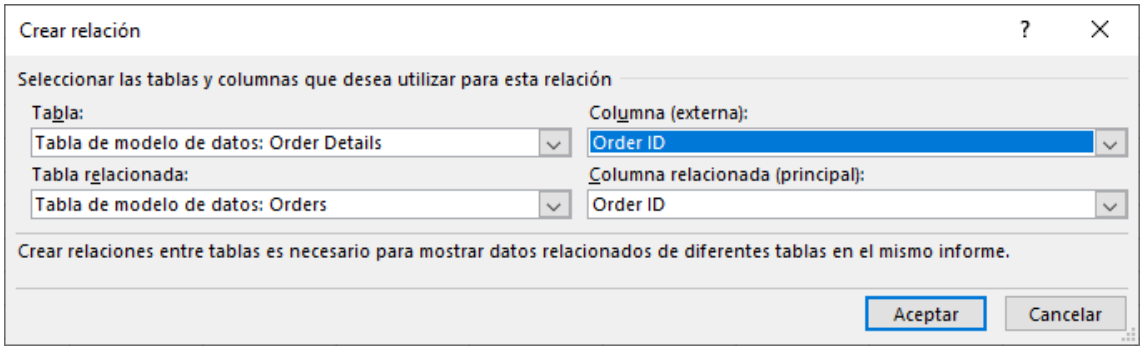


Imagen 10.4 Creando otra relación entre tablas.

14. Haz clic en **Aceptar**.
    Excel agrega la segunda relación, como se muestra en la Imagen 10.5.

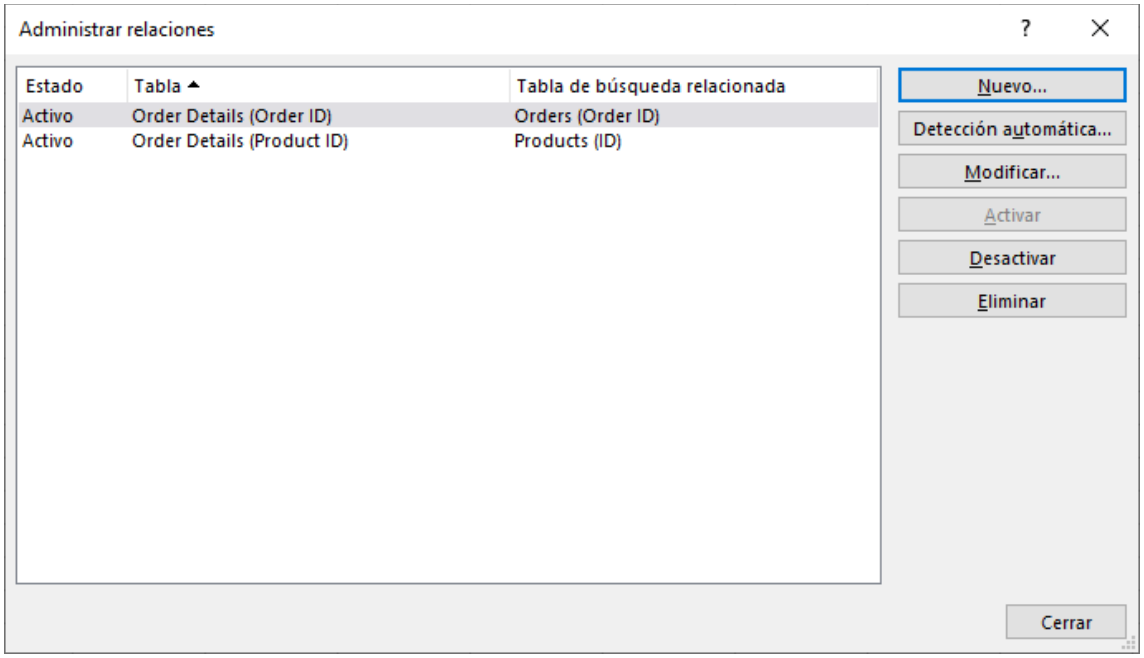


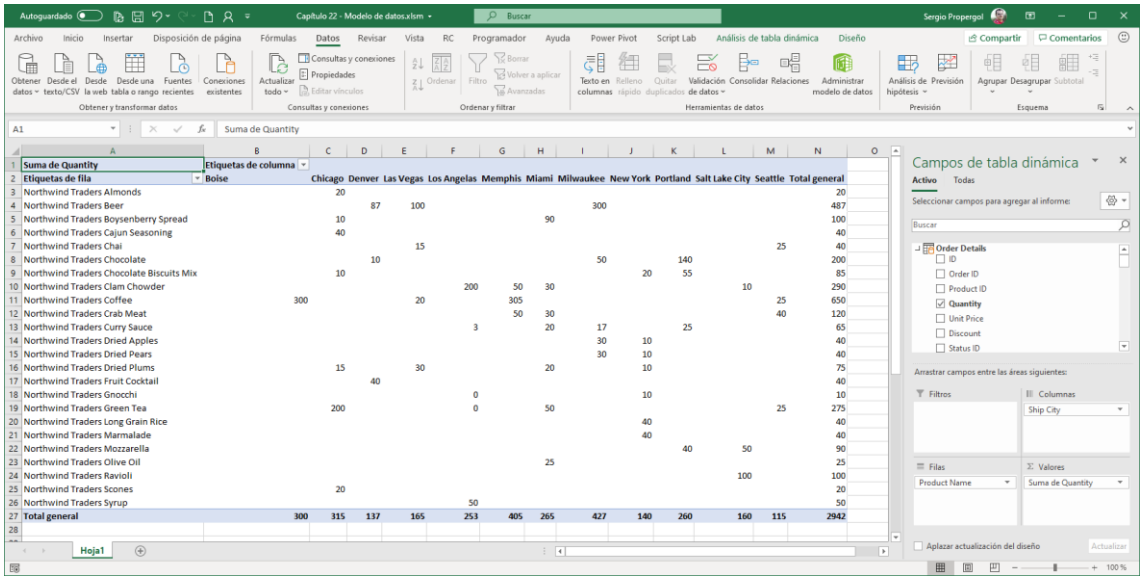
Imagen 10.5 Las relaciones entre las tablas del modelo de datos se muestran en el cuadro Administrar relaciones.

15. Haz clic en **Cerrar** para salir del cuadro **Administrar relaciones**.
16. Una vez definidas las relaciones entre tablas, podemos proceder a crear la tabla dinámica. Vamos a elegir los campos del panel **Campos de tabla dinámica**.
17. En el panel **Campos de tabla dinámica**, expande la tabla Products y arrastra el campo Product Name al área de filas.

Atención

En cualquier momento, podemos agregar otras tablas de la base de datos. También es posible almacenar datos de libros de Excel, aunque primero debes convertirlos en tablas de Excel.

18. Ahora expande la tabla Orders y arrastra Ship City al área de columnas.
19. Expande la tabla Orders Details y arrastra el campo Quantity al área de Valores.
20. Realiza los cambios de formato que desees utilizando los botones de la ficha **Diseño de la tabla dinámica**. La tabla dinámica completa se muestra en la Imagen 10.6.



Suma de Quantity	Etiquetas de columna												
Etiquetas de fila	Boise	Chicago	Denver	Las Vegas	Los Angeles	Memphis	Miami	Milwaukee	New York	Portland	Salt Lake City	Seattle	Total general
Northwind Traders Almonds		20											20
Northwind Traders Beer			87	100				300					487
Northwind Traders Boysenberry Spread		10					90						100
Northwind Traders Cajun Seasoning		40											40
Northwind Traders Chai				15								25	40
Northwind Traders Chocolate			10					50		140			200
Northwind Traders Chocolate Biscuits Mix		10							20	55			85
Northwind Traders Clam Chowder					200	50	30				10		290
Northwind Traders Coffee		300		20		305						25	650
Northwind Traders Crab Meat						50	30					40	120
Northwind Traders Curry Sauce					3		20	17		25			65
Northwind Traders Dried Apples								30	10				40
Northwind Traders Dried Pears								30	10				40
Northwind Traders Dried Plums		15		30			20		10				75
Northwind Traders Fruit Cocktail			40										40
Northwind Traders Gnocchi					0				10				10
Northwind Traders Green Tea		200			0		50					25	275
Northwind Traders Long Grain Rice									40				40
Northwind Traders Marmalade									40				40
Northwind Traders Mozzarella										40	50		90
Northwind Traders Olive Oil							25						25
Northwind Traders Ravioli											100		100
Northwind Traders Scones		20											20
Northwind Traders Syrup					50								50
Total general		300	315	137	165	253	405	265	427	140	260	160	115	2942

Imagen 10.6 La tabla dinámica resume las ventas de productos por ciudad.

11 El modelo de datos desde VBA

Además del modelo de objetos existente de VBA, Excel tiene un modelo de objetos "Data Model"(OM), que puede utilizarse para cargar y refrescar por medio de programación las fuentes de datos y trabajar con el modelo de datos. Podemos encontrar muchos objetos nuevos en el **Examinador de objetos** buscando la palabra "model", como se muestra en la Imagen 11.1.

En las últimas versiones de Excel se han añadido un gran número de objetos para apoyar el acceso desde VBA al modelo de datos. El siguiente procedimiento muestra cómo usar la propiedad `Model` del objeto `Workbook` para obtener información sobre el modelo de datos que creamos en la sección anterior.

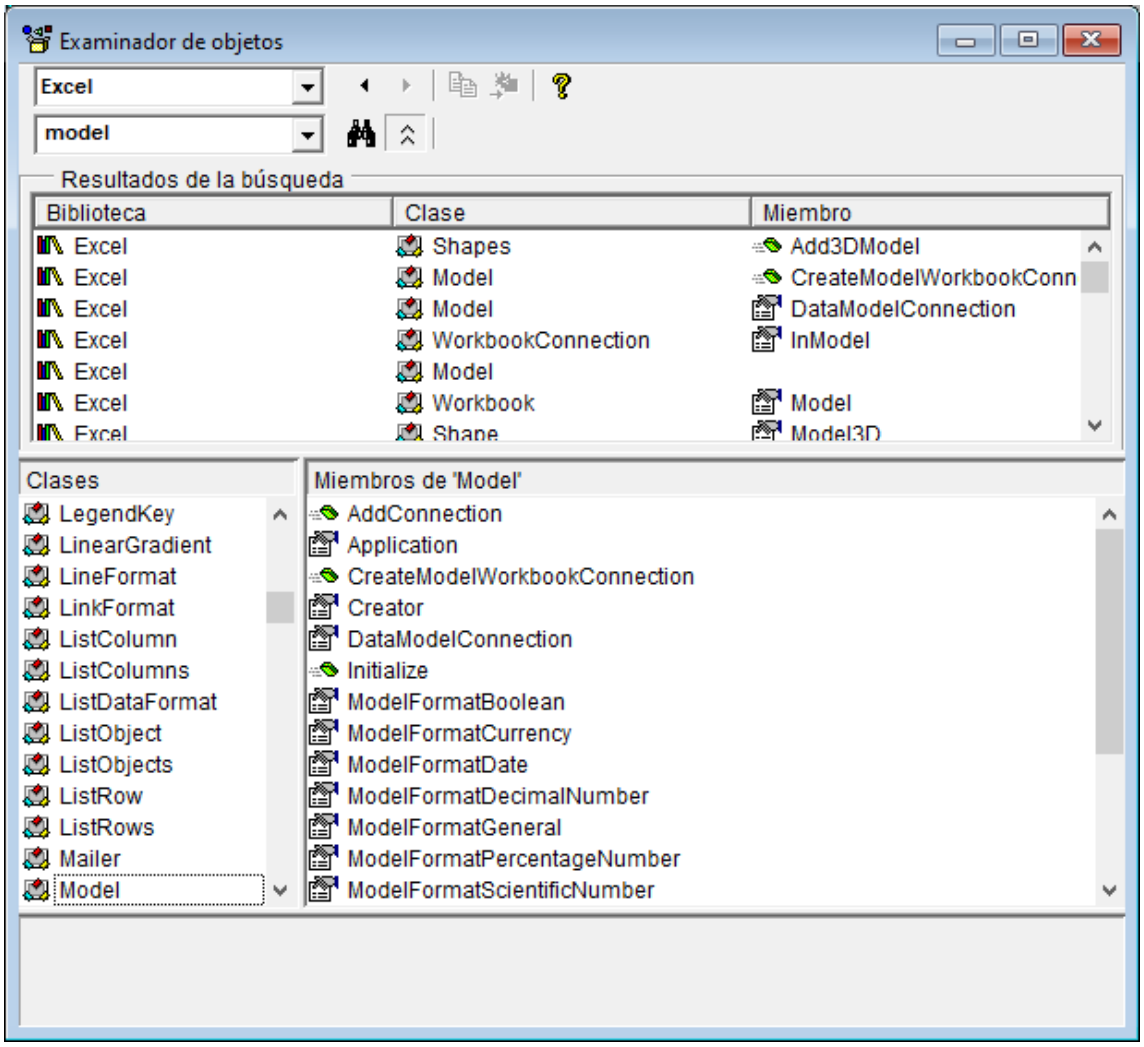


Imagen 11.1 Usando el Examinador de objeto para explorar los n uevos objetos del modelo de objetos.

```
Sub InfoModeloObjetos()
    Dim wkb As Workbook
    Dim tbl As Variant

    Set wkb = ActiveWorkbook
    Debug.Print "Nombre del modelo: " & wkb.Model.Name
    Debug.Print "Relaciones: " & _
    wkb.Model.ModelRelationships.Count
    Debug.Print "Número de tablas: " & _
    wkb.Model.ModelTables.Count
    Debug.Print "--NOMBRE DE LAS TABLAS--"
    For Each tbl In wkb.Model.ModelTables
        Debug.Print tbl.Name
    Next
End Sub
```

Cuando ejecutamos este procedimiento en la ventana **Inmediato**, se muestra la siguiente información:

```
Nombre del modelo: ThisWorkbookDataModel
Relaciones: 2
Número de tablas: 3
--NOMBRE DE LAS TABLAS--
Order Details
Orders
Products
```

En el siguiente ejercicio trabajaremos con el objeto `ModelChanges` que contiene información sobre los cambios hechos en el modelo de datos cuando sucede el evento `Workbook_ModelChange`. Aunque se pueden hacer varios cambios en el modelo de datos, este ejercicio se enfoca en detectar si se agregó una tabla nueva al modelo.

Para realizar este ejercicio debes completar previamente el anterior.

1. En el editor de VBA selecciona el nombre del proyecto de VBAProject (Capítulo 22 – Modelo de datos.xlsm) y haz clic en **Insertar – Módulo**.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub CambiosTabla()
    Dim strCmdTxt_1 As String
    Dim strCmdTxt_2 As String

    strCmdTxt_1 = """Order Details"",""Orders"",""Products"""
    strCmdTxt_2 = strCmdTxt_1 & ",""Customers"",""Employees"""
    With ActiveWorkbook.Connections("Northwind 2007") _
        .OLEDBConnection
        .CommandText = strCmdTxt_2
        .Refresh
    End With
End Sub
```

   Este procedimiento modifica la propiedad `CommandText` del objeto `OLEDBConnection` para incluir dos tablas más (**Customers** y **Employees**) en el modelo de datos. El método `Refresh` le dice a Excel que actualice el modelo de datos con los nuevos datos. Después de ejecutar este procedimiento las nuevas tablas deben aparecer en el panel **Campos de tabla dinámica** de la hoja. Para detectar el cambio en el modelo de datos, debes escribir el procedimiento de eventos `Workbook_ModelChange` como se indica a continuación:

3. Haz doble clic en el objeto `ThisWorkbook` en el **Explorador de proyectos**.
4. Introduce el siguiente procedimiento de evento:

```
Private Sub Workbook_ModelChange( _
```

```
ByVal Changes As ModelChanges)
    Dim colTblNames As ModelTableNames
    Dim tblCount As Long
    Dim i As Integer

    Set colTblNames = Changes.TablesAdded
    tblCount = colTblNames.Count
    If tblCount > 0 Then
        Debug.Print "Se han añadido " & tblCount & " tablas."
    Else
        Debug.Print _
        "No hay tablas nuevas en el modelo de datos."
    End If
    For i = 1 To tblCount
        Debug.Print colTblNames.Item(i)
    Next i
End Sub
```

Este procedimiento de evento se activa cuando Excel detecta que se han hecho cambios en el modelo de datos. La variable `Changes` representa el objeto `ModelChanges` e indica el tipo de cambio que se hizo. Los cambios pueden ser del tipo:

- Agregar, cambiar y eliminar columnas (propiedades `ColumnsAdded`, `ColumnsChanged` y `ColumnsDelete`).
- Agregar, cambiar, borrar, renombrar y refrescar (recalcular) tablas (propiedades `TablesAdded`, `TablesChanged`, `TablesDeleted`, `TableNamesChanged` y `TablesModified`).
- Cambiar una o más relaciones en el modelo (propiedad `RelationshipChange`).
- Agregar medidas (propiedad `MeasuresAdded`).
- Hacer un cambio desconocido (propiedad `UnknowChange`).

Cuando se añaden tablas al modelo, se usa la propiedad `Changes.TablesAdded` para encontrar los nombres de las tablas añadidas. Esta propiedad devuelve una colección de objetos `ModelTableNames` que contiene los nombres de las nuevas tablas.

```
Set colTblNames = Changes.TablesAdded
```

Puedes contar cuantos objetos hay en la colección `colTblNames` usando la propiedad `Count` del objeto `ModelTableNames`.

```
tblCount = colTblNames.Count
```

Se usa el siguiente código para obtener los nombres de las tablas añadidas al modelo de objetos:

```
For i = 1 To tblCount
    Debug.Print colTblNames.Item(i)
```

```
Next i
```

5. Ejecuta el procedimiento `CambiosTabla`.
   Se activará de forma automática el procedimiento `Workbook_ModelChange` que escribiste en el paso anterior. Ahora deberías ver la siguiente información en la ventana **Inmediato** cuando se completa su ejecución:

```
Se han añadido 2 tablas.
Customers
Employees
```

Deberías ver también las tablas agregadas en el panel **Campos de tabla dinámica** en el libro, activando la ficha **Todas** (ver Imagen 11.2):

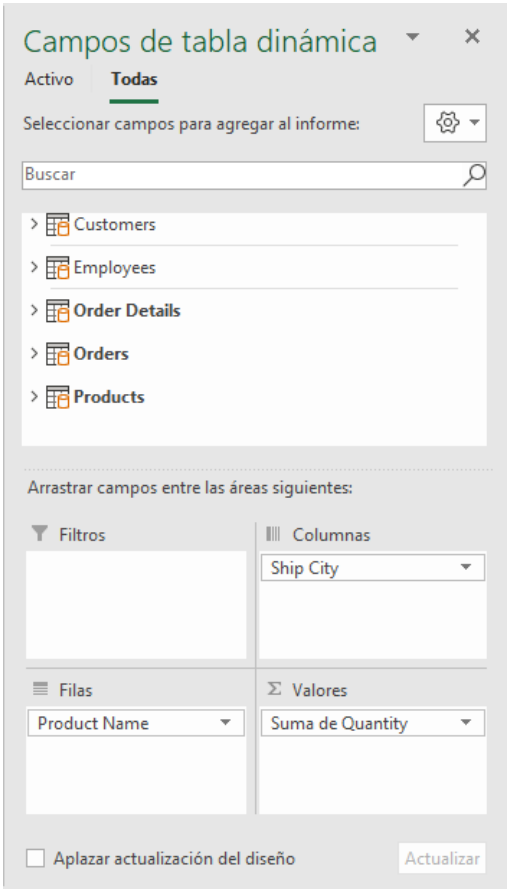


Imagen 11.2 El panel Campos de tabla dinámica mostrando las nuevas tablas.

6. Modifica la propiedad `CommandText` del procedimiento `CambiosTabla` insertando la siguiente línea:

```
.CommandText = strCmdTxt_1
```

7. Ejecuta el procedimiento de nuevo.
   Esto eliminará las tablas Customers y Employees del modelo de datos. Ahora deberían aparecer únicamente las tres tablas originales en el panel **Campos de tabla dinámica** y el siguiente texto en la ventana **Inmediato**.

```
No hay tablas nuevas en el modelo de datos.
```

12 Resumen

En este capítulo has trabajado con dos objetos muy importantes en Excel utilizados para el análisis de datos: Tablas dinámicas y gráficos dinámicos. Has aprendido a utilizar VBA para manipular estos dos objetos y crear rápidamente informes que te permitirán examinar fácilmente grandes cantidades de datos extraídos de un rango de celdas de una hoja de cálculo de Excel o de una fuente de datos externa, como Microsoft Access.

En este capítulo también se ha presentado la segmentación de datos, la herramienta que permite filtrar visualmente los datos de la tabla dinámica. En las dos últimas secciones del capítulo has aprendido a utilizar el modelo de datos de Excel para cargar datos de varias tablas, crear relaciones entre ellas, mostrar los datos en una tabla dinámica y a utilizar varios objetos y propiedades para obtener información sobre el modelo de datos.

En el siguiente capítulo se trabajará con Power Query, la herramienta de extracción, transformación y carga (ETL por sus siglas en inglés) para manipular información.

Capítulo 23
Power Query

Si necesitas dar formato a los datos de forma habitual y quieres automatizar este trabajo (transformación, limpieza y carga de datos), Excel cuenta con una potente herramienta que te ayudará en esta tarea.

Se trata de **Power Query**, también conocida en otras versiones de Excel como “Obtener y transformar”.

Podemos acceder a esta herramienta desde el grupo **Obtener y transformar datos** de la ficha **Datos** (ver Imagen 1.1). Con la tecnología de Power Query se pueden crear potentes consultas que simplifican el proceso de introducción de datos en Excel, así como el de combinarlos y transformarlos.

Comenzaremos con un rápido ejercicio de introducción al análisis de datos con Power Query.

1 El botón Obtener datos

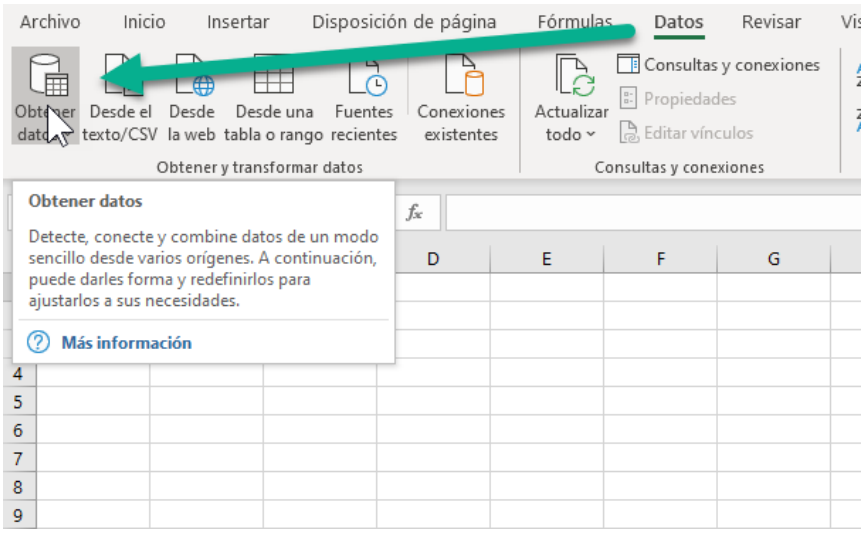


Imagen 1.1 El botón Obtener datos en el grupo Obtener y transformar datos de la ficha Datos se utiliza para cargar, manipular y refinar datos.

Cuando hacemos clic en el botón **Obtener datos**, Excel muestra una lista de tipos de fuentes de datos que podemos usar para crear una consulta (ver Imagen 1.2).

Atención

Quizá las fuentes de datos que aparecen en el manual no estén disponibles para tu versión de Excel. Los ejercicios de este capítulo están basados en Office 365 (2019) Profesional.

- La categoría **Desde un archivo** permite importar datos desde un archivo como un libro de Excel, csv, xml y JSON. Además, podemos importar datos desde una carpeta que contenga varios archivos (muy útil cuando necesitamos crear un informe desde numerosos archivos).
- La categoría **Desde una base de datos** permite importar datos desde cualquier tipo de base de datos (SQL Server, Access, Analysis Services, Oracle, MySQL y muchas otras).
- La categoría **Desde Azure** permite importar datos desde los servicios de datos de Microsoft Azure.

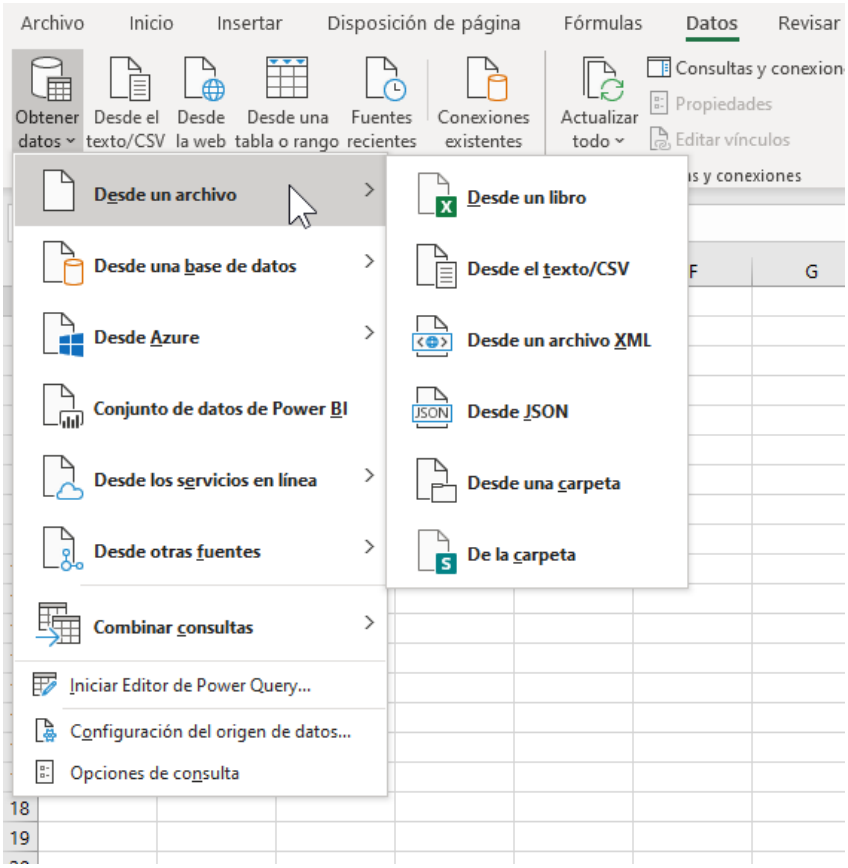


Imagen 1.2 Podemos obtener datos desde numerosos tipos de archivos e incluso carpetas que contengan varios archivos.

- La categoría **Desde los servicios en línea** permite importar datos desde aplicaciones como Facebook.

- La categoría **Desde otras fuentes** muestra una lista de otras fuentes que nos pueden proporcionar datos para las consultas. También podemos comenzar desde una consulta en blanco.

La categoría **Combinar consultas** contiene opciones para combinar y anexar consultas y así crear consultas más complejas.

Las demás opciones incluidas en el botón **Obtener datos** contienen opciones que nos permitirán abrir Power Query, administrar la configuración de las fuentes de datos y ver las opciones de consulta.

2 Las consultas

Para crear una consulta se empieza por seleccionar una fuente de datos utilizando el botón **Obtener datos**. Como hemos visto, los datos pueden encontrarse en un archivo local en nuestros equipos, en la nube o pueden encontrarse en un servicio web. La fuente de datos seleccionada se graba como "paso número 1" en el editor de consultas de Power Query. Nuestra consulta probablemente contendrá más de un paso. Tras hacer una conexión con los datos se mostrará el editor de consultas, donde podremos dar forma a los datos para adaptarlos a nuestras necesidades. La cinta del editor de consultas consta de las siguientes fichas:

- La ficha **Inicio**: Muestra botones de tareas comunes, como gestión de columnas y filas, ordenación, acceso al Editor avanzado y propiedades de la consulta, actualización de la vista previa, combinación de consultas, transformación de datos, carga de datos y también creación de nuevas consultas.
- La ficha **Transformar**: Proporciona acceso a tareas de transformación de datos comunes: formato de columnas, limpieza de datos o dinamizar columnas. Aquí encontraremos también opciones para mover y copiar columnas y trabajar con tablas.
- La ficha **Agregar columna**: contiene botones para agregar columnas personalizadas, dar formato a los datos y fusionar y duplicar columnas.
- La ficha **Vista**: Contiene diferentes opciones de visualización.

Desde la cinta de opciones del editor de Power Query podemos eliminar columnas y filas, agregar columnas, cambiar el tipo de dato, dar formato a los datos, realizar cálculos y muchas tareas más.

El editor de Power Query grabará cada paso que realicemos desde los comandos de esta cinta de opciones, desde los datos originales hasta que finalicemos la transformación. Cada tarea de transformación de datos que realicemos se etiqueta automáticamente para que sea fácil localizar cada paso. Podemos renombrar estos pasos para hacer que tengan más significado para nosotros.

Es posible borrar o modificar un paso, cambiar el orden e insertar nuevos pasos. Una consulta no es más que una colección de pasos ordenados, cuyo último paso devuelve un resultado a una hoja de cálculo o al modelo de datos de Excel.

Los pasos de Power Query están escritos en un lenguaje denominado **M**. Podemos ver el código de las consultas en el **Editor avanzado** y en la **Barra de fórmulas**. Es posible editar el código manualmente y agregar nuevos pasos a la consulta.

Las consultas de Power Query se guardan conjuntamente con el libro de Excel. Un libro puede contener varias consultas. Estas consultas se pueden ver haciendo clic en el botón **Consultas y conexiones** de la ficha **Datos** de la cinta de opciones de Excel.

Si disponemos de una licencia de Office Professional o Professional Plus, podemos compartir las consultas con otras personas. En vez de enviar los libros por correo electrónicos, podemos guardar la consulta para que tengan acceso a ella.

Las consultas son muy flexibles. Se pueden duplicar fácilmente si necesitas hacer algunos cambios sin modificar la consulta original. También podemos combinar dos consultas uniendo dos tablas con una declaración SQL, o podemos anexar una consulta debajo de otra. Una consulta también puede servir como fuente de datos para otra consulta y así crear transformaciones de datos más complejas.

Vamos a familiarizarnos con la interfaz de usuario de Power Query creando un proyecto completo. El objetivo es combinar y depurar los datos para poder elaborar un resumen de las oficinas de correos activas y cerradas por cada estado de Estados Unidos. Aunque este proyecto puede realizarse utilizando las herramientas Cortar y Pegar fórmulas y con otras utilidades que incorpora la cinta de opciones de Excel, la opción de usar Power Query es más eficiente. Como las consultas son grupos de pasos, no tendrás que realizar las mismas tareas si los datos cambian, simplemente hay que hacer clic en el botón **Actualizar** y se volverán a ejecutar los pasos sin que tengamos que hacer nada más.

1. Copia la carpeta Power Query a C:\Archivos Manual VBA.
2. Crea un libro nuevo de Excel.
3. Guárdalo en la carpeta Power Query con el nombre Oficinas- Consultas.xlsx.

Paso 1: obtener datos de un libro de Excel

---

4. En la ficha **Datos** haz clic en **Obtener datos > Desde un archivo > Desde un libro**.
5. Selecciona el archivo PostOffice_NY_NJ.xlsx y haz clic en **Importar**.
6. En la ventana **Navegador**, haz clic en la casilla de verificación **Seleccionar varios elementos**, lo que te permitirá seleccionar los dos elementos (ver Imagen 2.1). Observa que en el panel de la derecha se muestran los contenidos del elemento seleccionado.
7. Haz clic en el desplegable del botón **Cargar** y selecciona **Cargar en**, como se muestra en la Imagen 2.1.
8. En la ventana **Importar datos** haz clic en el botón de opción **Tabla** y selecciona **Hoja de cálculo nueva**. Selecciona también **Agregar estos datos al Modelo de datos**.

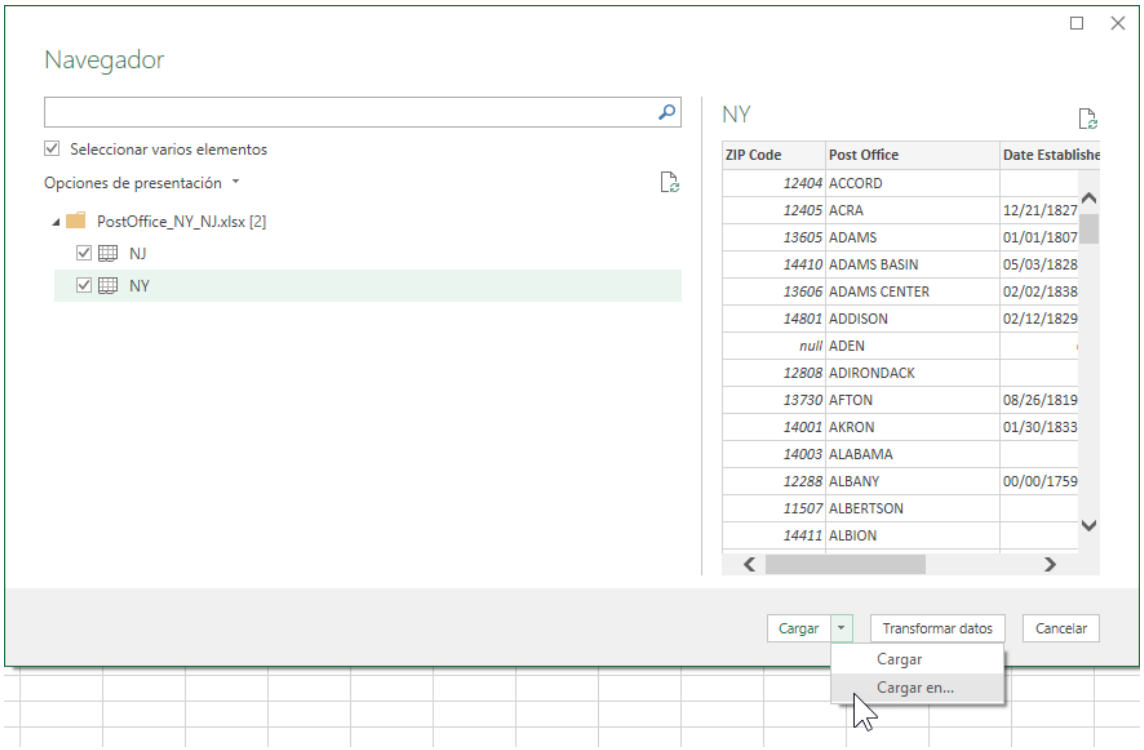


Imagen 2.1 La ventana Navegador muestra los datos disponibles en el archivo seleccionado.

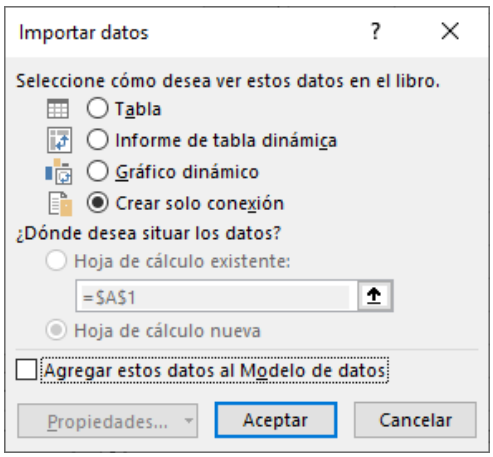


Imagen 2.2 En el cuadro Importar datos podemos especificar cómo y dónde queremos cargar los datos.

9. Haz clic en el botón **Aceptar** en el cuadro de diálogo.
   Excel crea dos tablas, ya que seleccionaste dos elementos en la ventana **Navegador**.
   Se abre en la parte derecha de la ventana el panel **Consultas y conexiones** (ver Imagen 2.3). Este panel también se puede mostrar desde la ficha datos.
   Si quieres ver la información de cada consulta, sitúa el ratón encima para tener una vista previa.
   En la parte inferior de la vista previa puedes encontrar opciones que te permitirán ver los datos, eliminar la consulta del libro o editarla. Observa que Excel muestra el número de filas que se cargaron con éxito. Si se hubiera producido algún error, se mostraría un hipervínculo para obtener más información sobre él.

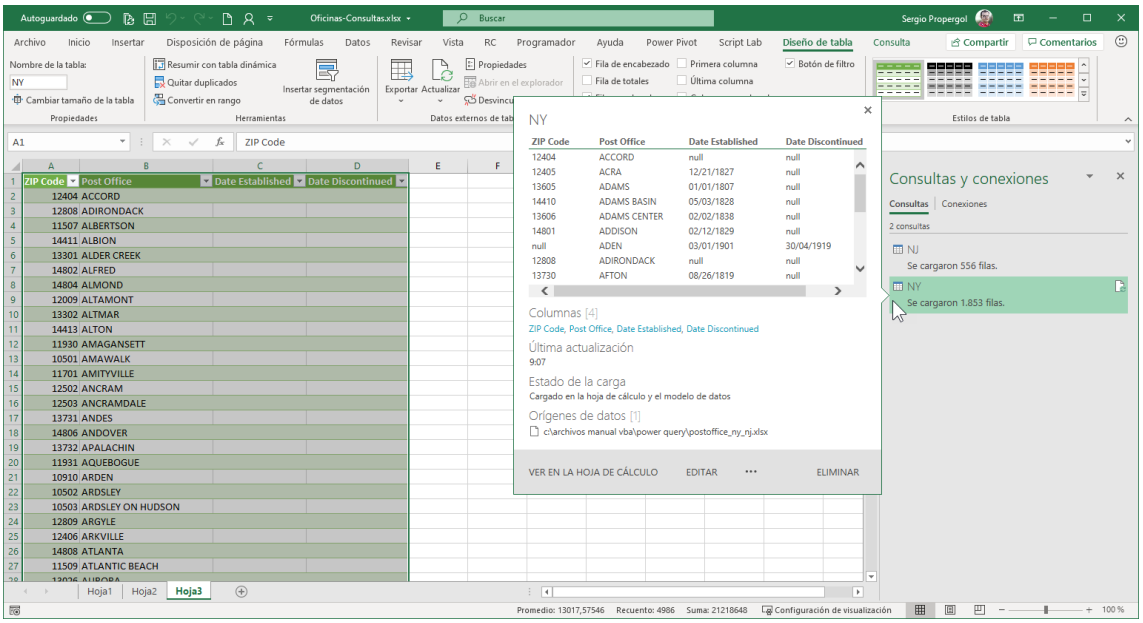


Imagen 2.3 El libro con dos consultas.

Si te fijas en los datos cargados, verás que se han copiado en dos tablas de Excel (una por cada hoja del archivo importado). La cinta de opciones muestra la ficha contextual Diseño de tabla. En este momento podríamos modificar el formato de la tabla o crear una tabla dinámica, pero vamos a trabajar con Power Query.

Con los datos importados no tenemos forma de identificar los datos que pertenecen a Nueva York (NY) o a Nueva Jersey (NJ). Necesitamos una columna en cada tabla para mostrar el estado.

Paso 2: Agregar, renombrar y mover una columna nueva

---

10. En el panel **Consultas y conexiones**, haz clic con el botón derecho en la consulta NJ y selecciona **Editar**.
    Aparece el editor de Power Query mostrando cuatro pasos en la sección **Pasos aplicados**. Estos son creados automáticamente cuando se cargan los datos (ver Imagen 2.4).

    Podemos eliminar un paso haciendo clic en el botón **X** delante del nombre del paso. Ten en cuenta que esto no se puede deshacer. Un paso eliminado debe ser creado de nuevo. Fíjate también en que algunos pasos tienen un icono de configuración (ver Imagen 2.4 y Imagen 2.5). Estos pasos se pueden editar usando el mismo cuadro de diálogo que se utilizó para crearlos. Simplemente hay que hacer clic en el icono. El icono de configuración desaparecerá si haces algún cambio no válido en la fórmula del paso. La Imagen 2.5 muestra el cuadro de diálogo utilizado para editar el paso **Origen**:

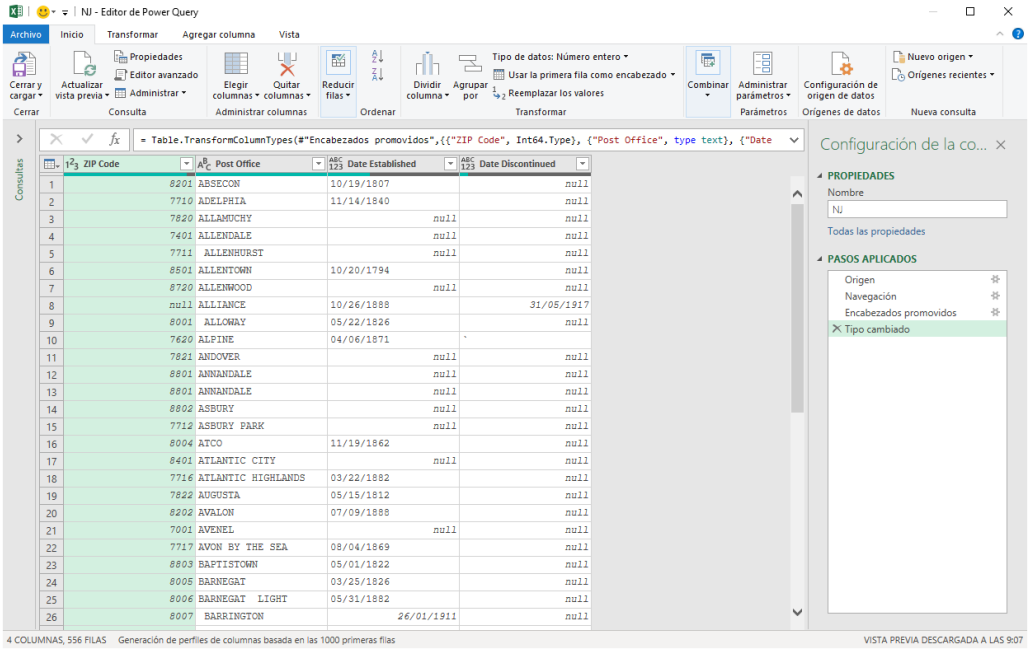


Imagen 2.4 El editor de Power Query muestra los pasos datos y el estado de los datos tras aplicar la transformación seleccionada.

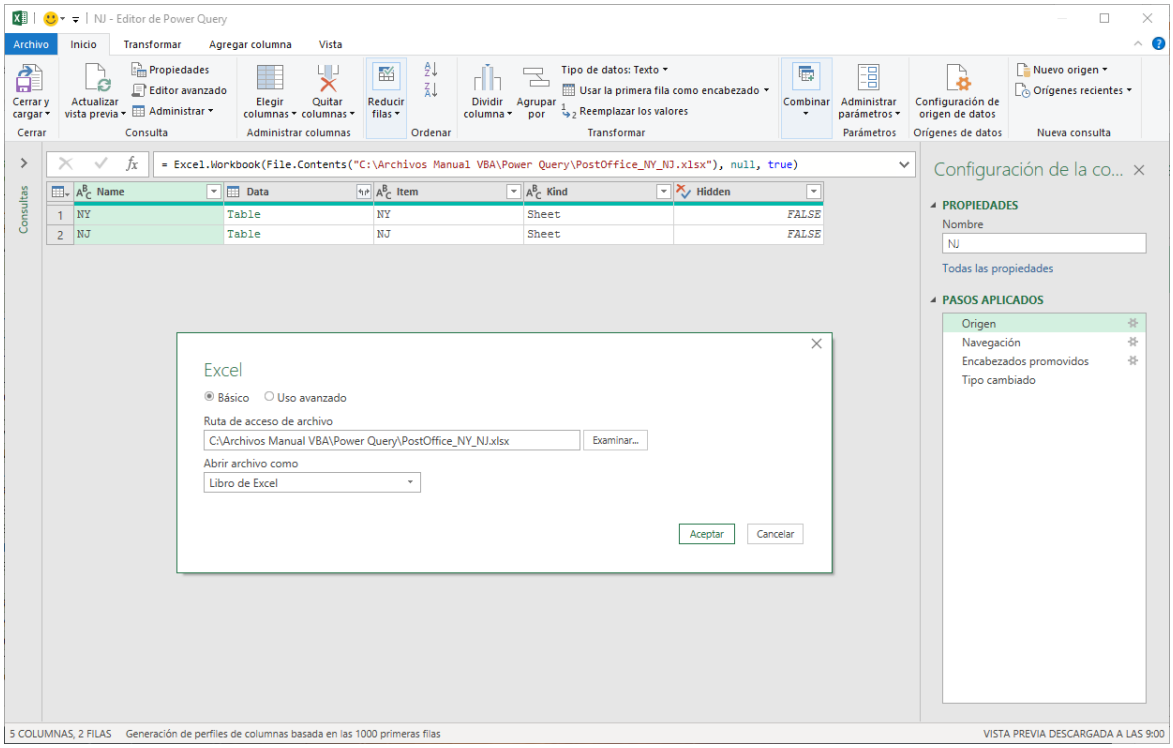


Imagen 2.5 Desde el cuadro de diálogo podemos editar el origen de los datos.

11. Haz clic en el paso **Encabezados promovidos** y fíjate en la barra de fórmulas. En caso de que no se muestre haz clic en la ficha **Vista – Barra de fórmulas** (ver Imagen 2.6). Es posible utilizar la barra de fórmulas para comprobar o editar la expresión M utilizada para realizar esa transformación. Podemos también usar la barra de fórmulas para crear un paso nuevo haciendo clic en el icono **fx**. La barra de fórmulas puede expandirse para ver el contenido completo haciendo clic en la flecha hacia abajo.

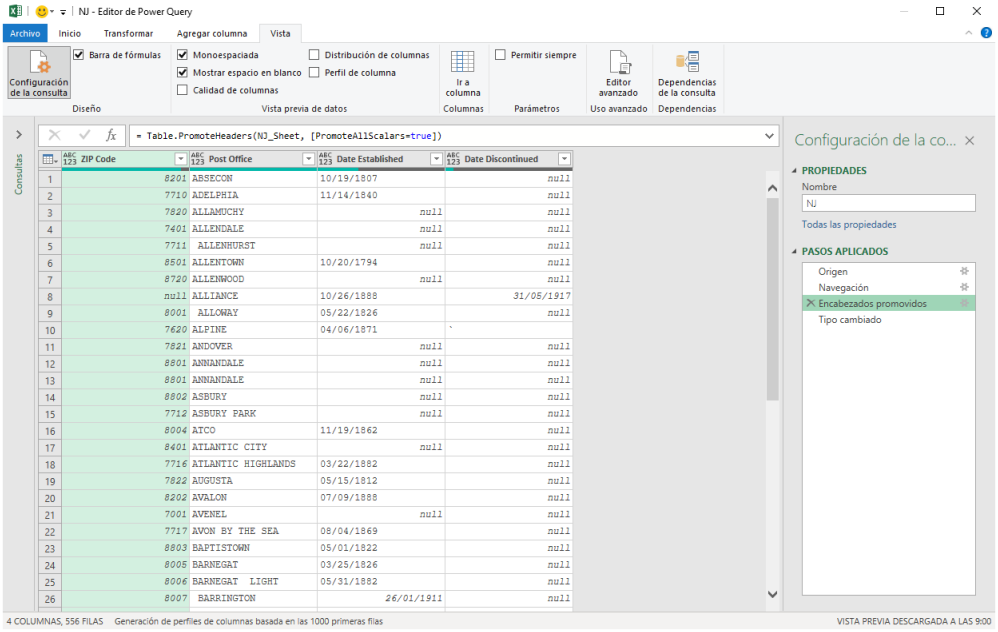


Imagen 2.6 Examinando la barra de fórmulas del paso seleccionado.

12. Selecciona el paso **Tipo cambiado** y observa que la fórmula hace referencia al paso anterior (Encabezados promovidos) (ver Imagen 2.7).

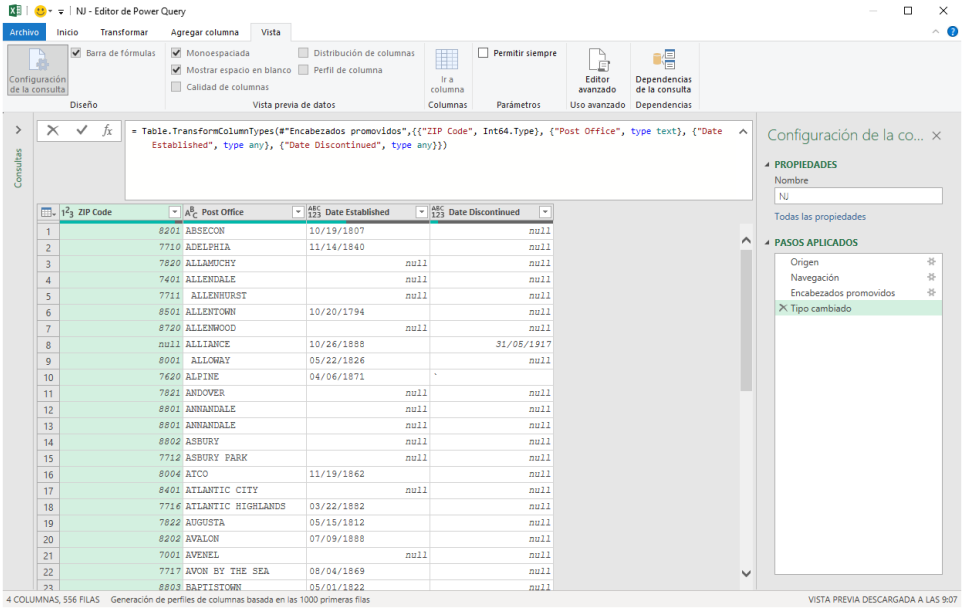


Imagen 2.7 La barra de fórmulas desplegada.

Atención

Los pasos cuyo nombre contiene uno o varios espacios deben escribirse como #"nombre consulta".

El paso **Tipo cambiado** se aplica automáticamente a los datos de todas las columnas.

Tipos de datos

Cada columna en Power Query tiene un tipo de dato específico. No es necesario declarar el tipo de dato de cada valor. Power Query lo hace automáticamente. No obstante, si obtenemos errores por un tipo de dato incorrecto, hay comandos en la cinta que los corregirán. Esta es la lista de tipos de datos que pueden usarse:

Tipo de dato	Descripción
Número decimal	Almacena valores numéricos, tanto enteros como decimales.
Moneda	Almacena valores de moneda.
Número entero	Almacena números enteros.
Porcentaje	Almacena valores divididos entre 100.
Fecha/Hora	Almacena fechas y horas.
Fecha	Almacena fechas entre el 01/01/0001 y el 31/12/9999.
Hora	Almacena unidades de hora.
Fecha/Hora/Zona horaria	Almacena los tres tipos de dato simultáneamente (fecha, hora y zona horaria).
Duración	Almacena la diferencia entre dos unidades de tiempo, fecha o zona horaria.
Texto	Almacena texto Unicode. Es sensible a mayúsculas.
Verdadero/Falso	Almacena valores lógicos.
Binario	Utilizado para almacenar imágenes y el contenido de archivos.

El lenguaje M es muy rígido. Eso significa que se generarán errores si los argumentos pasados a las funciones (o los valores que intentas combinar en expresiones) no coinciden con el tipo de dato esperado en la columna. Por ejemplo, la combinación de texto con un número como:

“Ejemplo” & 1

Utilizando el operador & generará un error porque los textos no se pueden combinar con números sin una conversión previa. Para evitar el error, puede usarse la función **NumberToText**.

“Ejemplo” & Number.To.Text (1)

13. Para agregar una columna nueva a la tabla actual, selecciona **Añadir columna – Columna personalizada**.
14. Introduce los datos que se muestran en la Imagen 2.8.

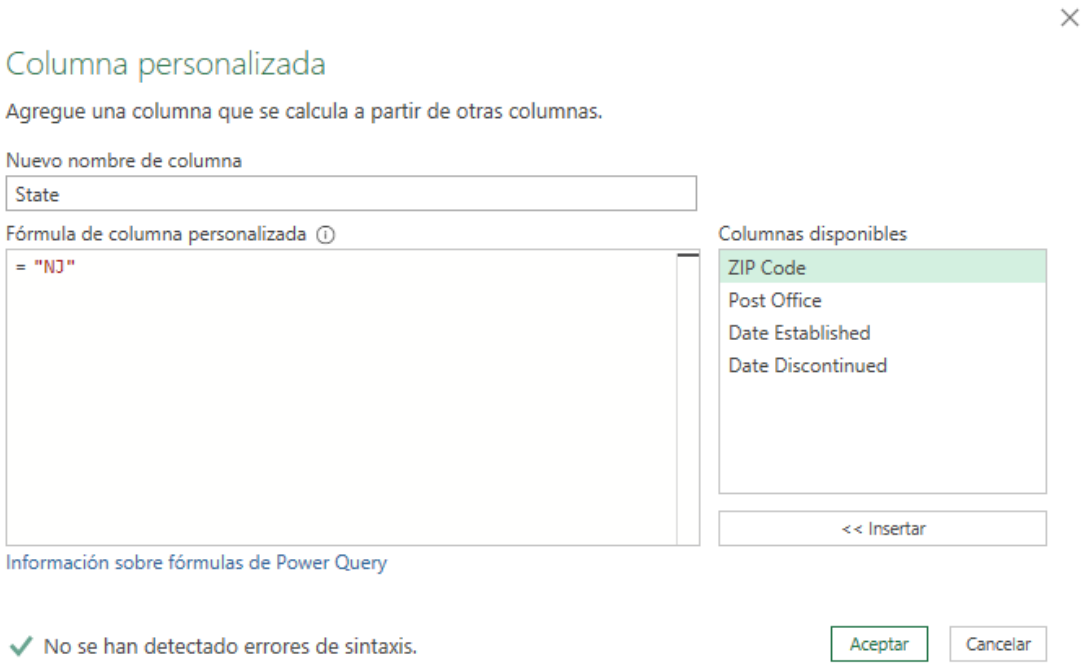


Imagen 2.8 Creando una nueva columna.

Power Query añade una nueva columna. De este modo ya se puede identificar que los códigos postales pertenecen al estado de Nueva Jersery (ver Imagen 2.9).

1. Haz clic con el botón derecho del ratón en el nuevo paso agregado y selecciona **Cambiar nombre**. Escribe “Agrega columna State”.
2. Haz clic con el botón derecho del ratón en el encabezado de la columna State y selecciona **Mover > Al principio**.
   La columna State debería aparecer delante de la columna ZIP code.
   Repite los mismos pasos para la consulta NY.
3. Haz clic en el panel **Consultas** en la parte izquierda de la ventana para desplegar el panel de navegación de consultas.
4. Selecciona la consulta NY para abrirla en el editor de Power Query.
   Observa que el panel **Configuración de la consulta** contiene los mismos pasos que la consulta NJ excepto los dos últimos (Agrega columna State y Columnas reordenadas (ver Imagen 2.10).

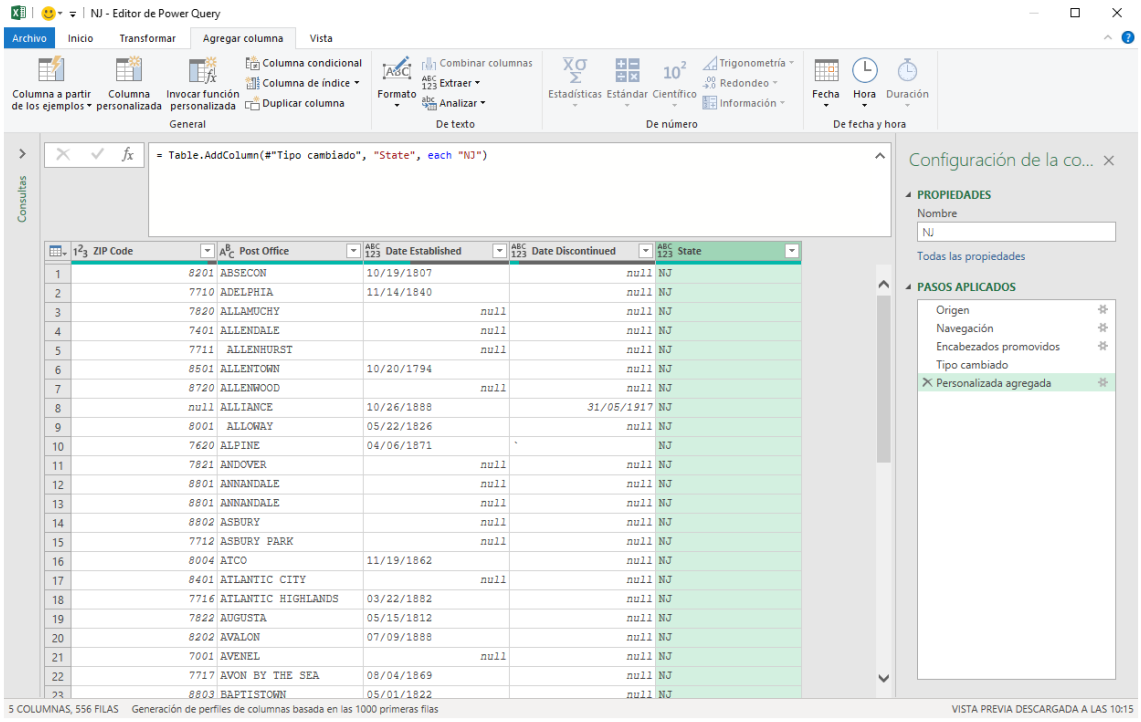


Imagen 2.9 La columna State aparece en forma de nuevo paso llamado Personalizada creada.

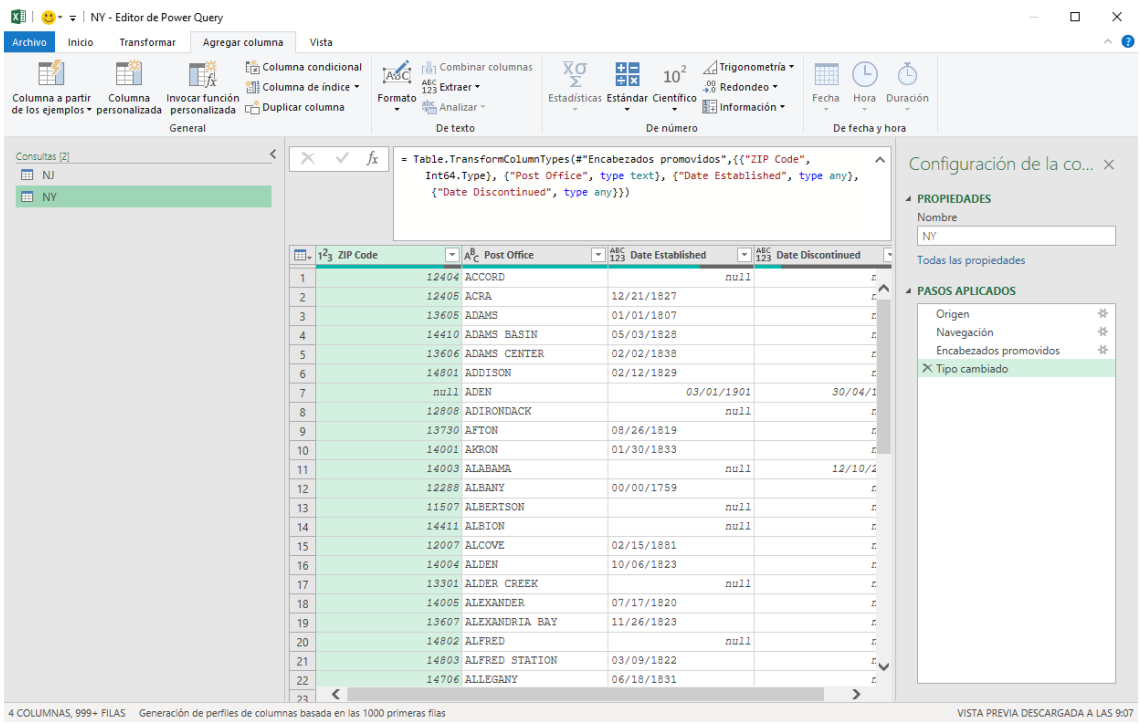


Imagen 2.10 Cargando la consulta NY en el editor de Power Query.

5. Agrega la columna State, renombra el paso y mueve la columna al principio (pasos 13 a 16).
6. Haz clic en la ficha **Inicio > Cerrar y cargar** para guardar los cambios, cerrar el editor de Power Query y cargar los datos en la ubicación por defecto.
   Excel actualiza las tablas para incluir la columna State (ver Imagen 2.11).

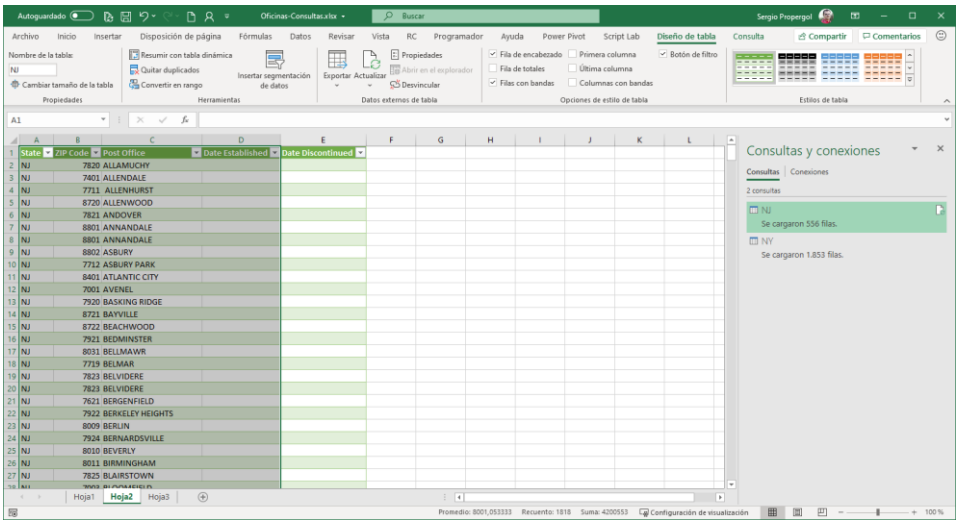


Imagen 2.11 La tabla de Excel ahora contiene una nueva columna.

7. Guarda el libro Oficinas-Consultas.xlsx pero mantenlo abierto.
   Al guardar el archivo, Excel también guarda las consultas creadas.

   A continuación introduciremos un estado nuevo desde otra fuente de datos: un archivo CSV.

Paso 3: Importar datos desde un archivo de texto

---

8. Haz clic en la ficha **Datos > Obtener datos > Desde un archivo > Desde el texto /CSV**.
9. Selecciona el archivo PostOffice_CT.csv en la carpeta Power Query e impórtalo.
   Excel muestra la vista previa del archivo como se muestra en la Imagen 2.12.

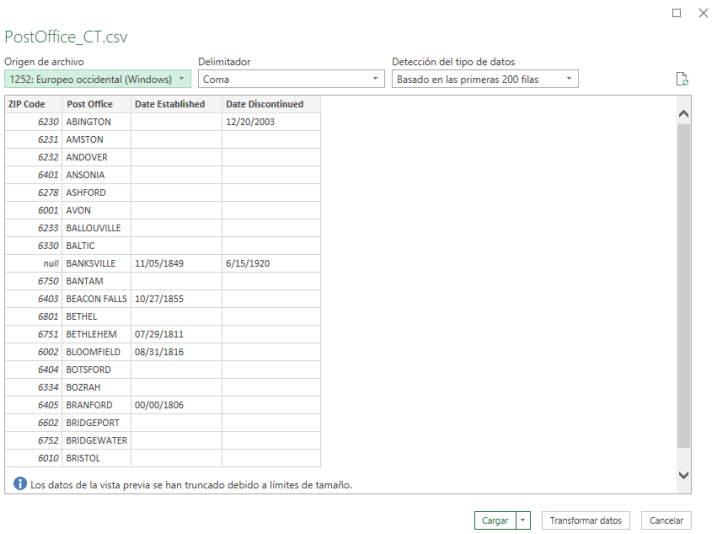


Imagen 2.12 Excel muestra esta vista cuando se carga un archivo CSV.

10. Haz clic en el desplegable del botón **Cargar** y selecciona **Cargar en**.
11. En cuadro el de diálogo **Importar datos** selecciona **Tabla >Hoja de cálculo nueva > Agregar estos datos al Modelo de datos**.
    Excel carga los datos en una nueva hoja y agrega la nueva consulta al panel **Consultas y conexiones** (ver Imagen 2.13).

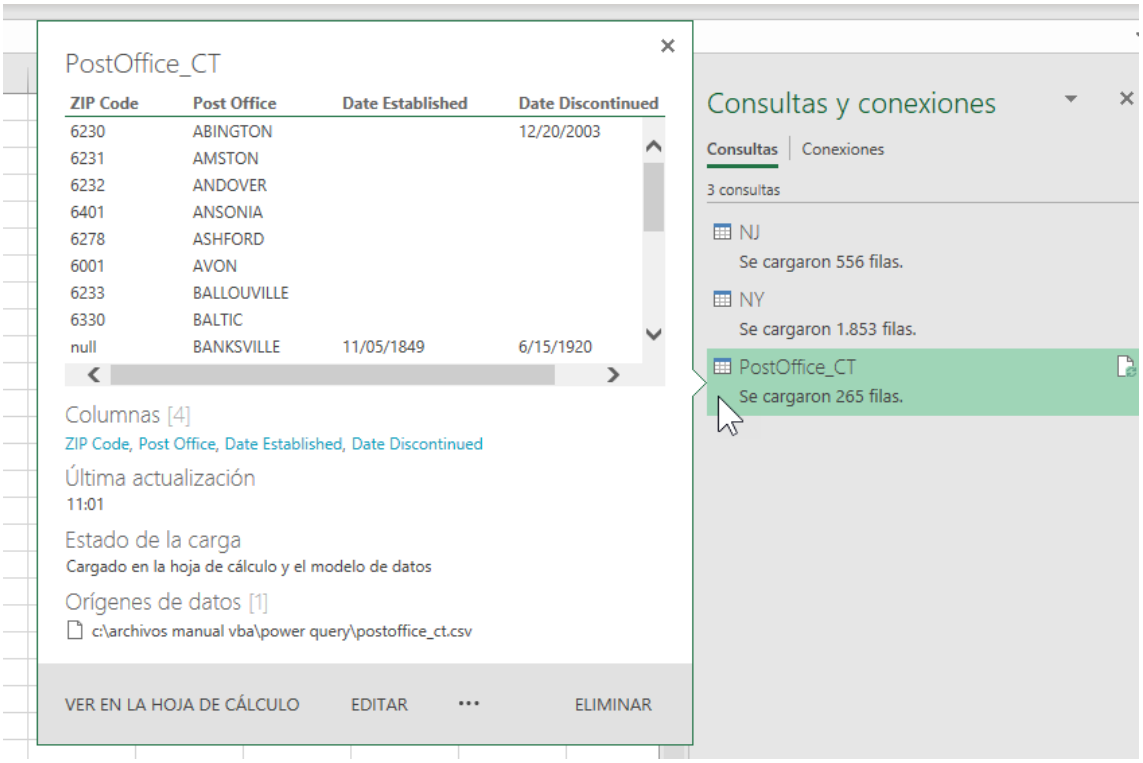
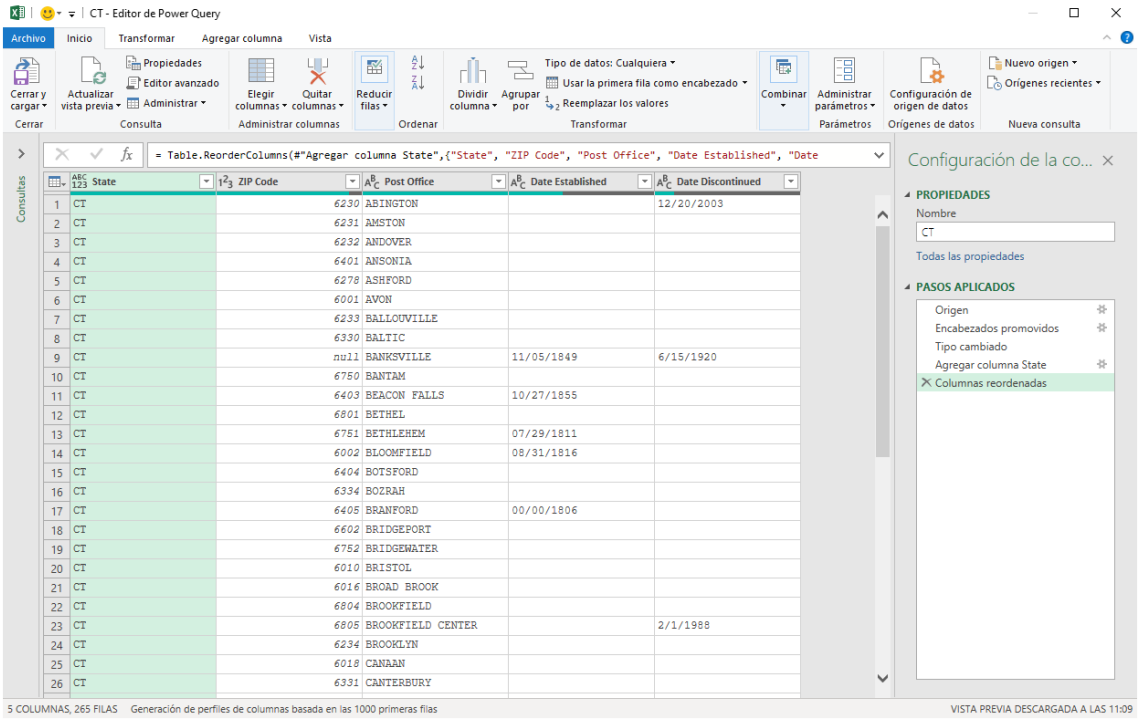


Imagen 2.13 Podemos modificar la consulta haciendo clic en el botón Editar.

12. Selecciona la consulta en el panel y haz clic en el botón **Editar** de la vista previa.
    Excel carga el archivo en el editor de PowerQuery. Parece que los datos se han importado correctamente en las cuatro columnas necesarias. Lo único que falta es la columna personalizada con el nombre del estado.
13. Agrega la columna State y sitúala al principio de la tabla, igual que en los pasos 13-16.
14. En el panel **Consultas y conexiones**, cambia el nombre a la consulta haciendo clic con el botón derecho sobre ella y seleccionando la opción correspondiente. El nombre que debes darle es “CT”. La Imagen 2.14 muestra el editor de Power Query tras los cambios hechos en los pasos 27-28.
15. Haz clic en **Archivo > Cerrar y cargar**.
    El libro Oficinas-Consultas ahora contiene tres consultas, que necesitarás para su manipulación a continuación.
16. Presiona **Ctrl + G** para guardar el libro con todas sus consultas.



	State	ZIP Code	Post Office	Date Established	Date Discontinued
1	CT	6230	ABINGTON		12/20/2003
2	CT	6231	AMSTON		
3	CT	6232	ANDOVER		
4	CT	6401	ANSONIA		
5	CT	6278	ASHFORD		
6	CT	6001	AVON		
7	CT	6233	BALLOUVILLE		
8	CT	6330	BALTIC		
9	CT	null	BANKSVILLE	11/05/1849	6/15/1920
10	CT	6750	BANTAM		
11	CT	6403	BEACON FALLS	10/27/1855	
12	CT	6801	BETHEL		
13	CT	6751	BETHLEHEM	07/29/1811	
14	CT	6002	BLOOMFIELD	08/31/1816	
15	CT	6404	BOTSFORD		
16	CT	6334	BOZRAH		
17	CT	6405	BRANFORD	00/00/1806	
18	CT	6602	BRIDGEPORT		
19	CT	6752	BRIDGEWATER		
20	CT	6010	BRISTOL		
21	CT	6016	BROAD BROOK		
22	CT	6804	BROOKFIELD		
23	CT	6805	BROOKFIELD CENTER		2/1/1988
24	CT	6234	BROOKLYN		
25	CT	6018	CANAAN		
26	CT	6331	CANTERBURY		

Imagen 2.14 La consulta CT tras la modificación.

Paso 4: Combinar los datos con la herramienta Anexar consultas

17. Para combinar los datos de los tres estados, haz clic con el botón derecho del ratón en la consulta NJ y selecciona **Anexar**.
    Aparece el cuadro de diálogo **Anexar** con dos cuadros desplegables. La tabla principal ya está seleccionada.
18. Haz clic en la opción **Tres o más tablas**.
    Excel muestra otra vista del cuadro **Anexar** donde puedes seleccionar las tablas que necesitas combinar.
19. Introduce los datos que se muestran en la Imagen 2.15. y haz clic en **Aceptar**.

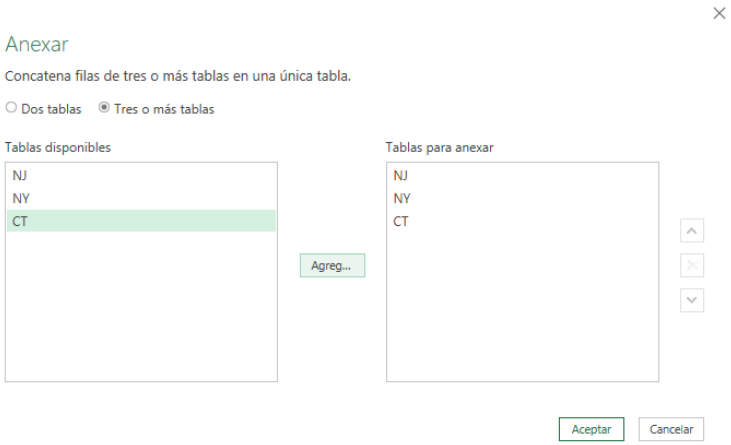
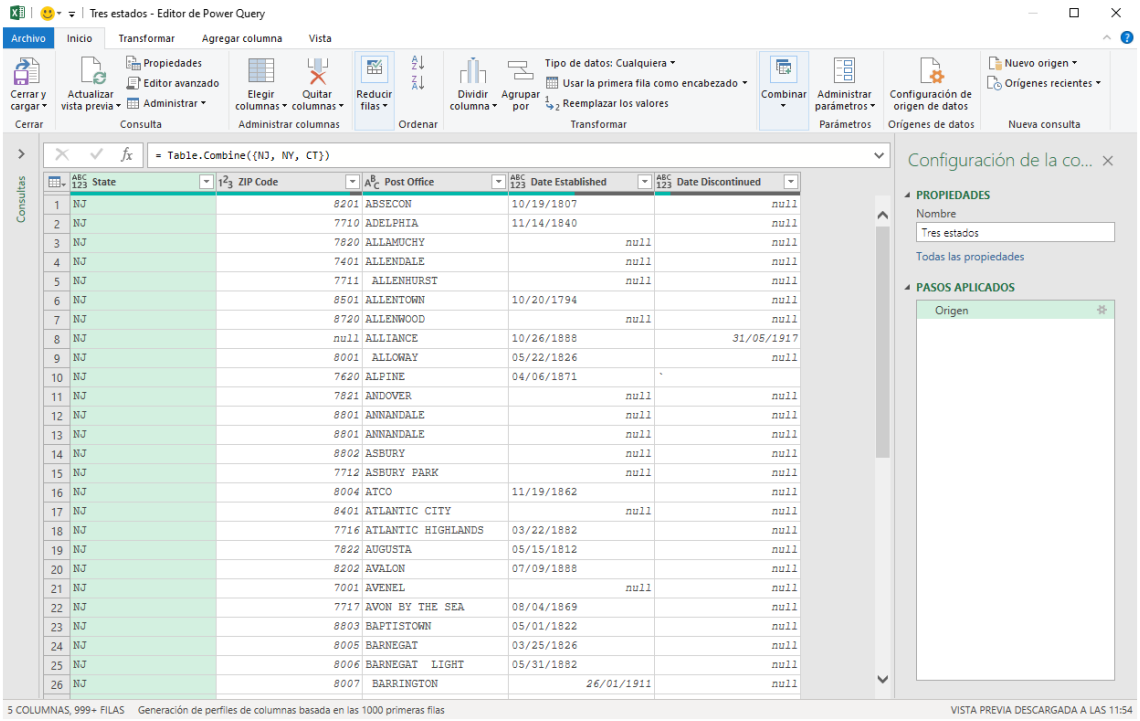


Imagen 2.15 Anexando tres tablas.

Excel abre el editor de Power Query. Puedes ver que se ha creado una consulta nueva llamada Anexar1.

20. Cambia el nombre de la consulta a “Tres estados” como se muestra en la Imagen 2.16.



	State	ZIP Code	Post Office	Date Established	Date Discontinued
1	NJ	8201	ABSECON	10/19/1807	null
2	NJ	7710	ADELPHIA	11/14/1840	null
3	NJ	7820	ALLAMUCHY	null	null
4	NJ	7401	ALLENDALE	null	null
5	NJ	7711	ALLENHURST	null	null
6	NJ	8501	ALLENTOWN	10/20/1794	null
7	NJ	8720	ALLENWOOD	null	null
8	NJ	null	ALLIANCE	10/26/1888	31/05/1917
9	NJ	8001	ALLOWAY	05/22/1826	null
10	NJ	7620	ALPINE	04/06/1871	
11	NJ	7821	ANDOVER	null	null
12	NJ	8801	ANNANDALE	null	null
13	NJ	8801	ANNANDALE	null	null
14	NJ	8802	ASBURY	null	null
15	NJ	7712	ASBURY PARK	null	null
16	NJ	8004	ATCO	11/19/1862	null
17	NJ	8401	ATLANTIC CITY	null	null
18	NJ	7716	ATLANTIC HIGHLANDS	03/22/1882	null
19	NJ	7822	AUGUSTA	05/15/1812	null
20	NJ	8202	AVALON	07/09/1888	null
21	NJ	7001	AVENEL	null	null
22	NJ	7717	AVON BY THE SEA	08/04/1869	null
23	NJ	8803	BAPTISTOWN	05/01/1822	null
24	NJ	8005	BARNEGAT	03/25/1826	null
25	NJ	8006	BARNEGAT LIGHT	05/31/1882	null
26	NJ	8007	BARRINGTON	26/01/1911	null

Imagen 2.16 El editor de Power Query muestra el resultado de la operación de anexado.

Observa que la barra de fórmulas muestra la instrucción que combina las tres tablas:

= Table.Combine({NJ, NY, CT})

Podrías editar la formula **Table.Combine** para incluir más elementos sin tener que abrir de nuevo el cuadro de diálogo Anexar.

21. Haz clic en **Archivo > Cerrar y cargar**.
    La consulta Tres estados aparecerá en el panel de consultas y la hoja activa mostrará la tabla de Excel con las oficinas de correos de los tres estados.
22. Presiona **Ctrl + G** para guardar el libro.

    Ahora que ya tenemos los datos en un solo lugar, vamos a proceder a su limpieza. Comenzaremos por eliminar los registros duplicados.

Paso 5: Limpiar los datos

23. Haz doble clic en la consulta Tres estados para abrir el editor de Power Query.

24. Selecciona la columna Post Office y haz clic en la ficha **Inicio > Eliminar filas > Eliminar duplicados**.

    Antes de dar forma a los datos en su versión final, necesitamos incluir un poco de lógica en la consulta. Para mostrar el recuento de oficinas de correos abiertas y cerradas, vamos a añadir una columna personalizada para categorizarlas.

25. Haz clic en **Agregar columna > Columna personalizada** y rellena el cuadro de diálogo con los siguientes datos:

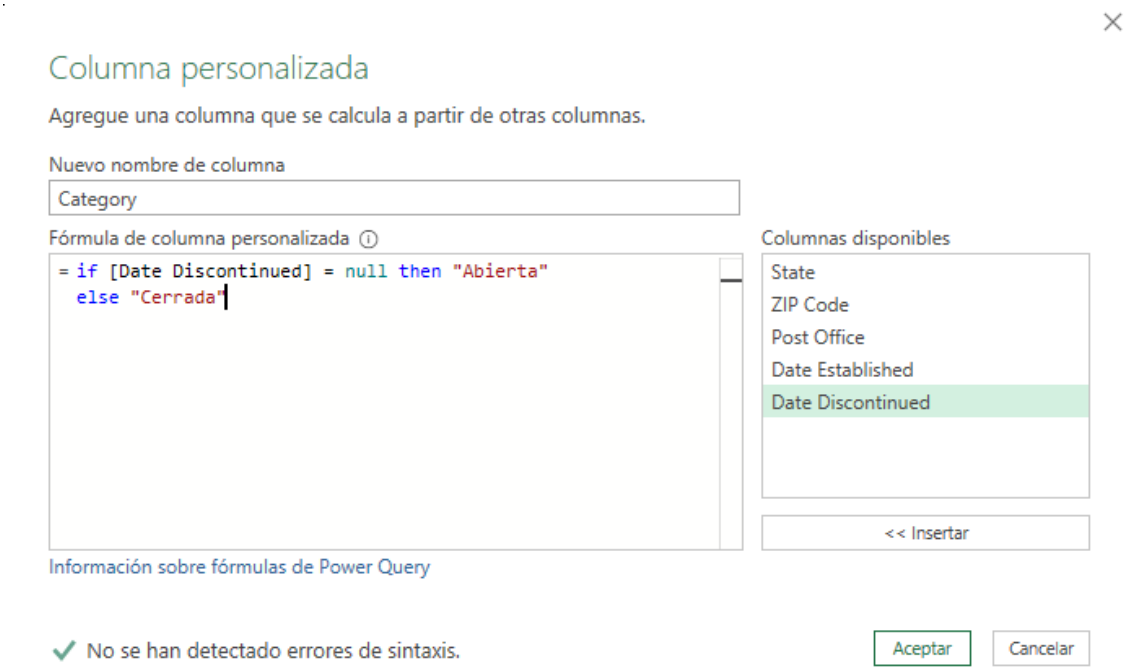


Imagen 2.17 Introduciendo una expresión lógica en una columna personalizada.

El editor de Power Query muestra ahora la columna Category con los valores Abierta o Cerrada.

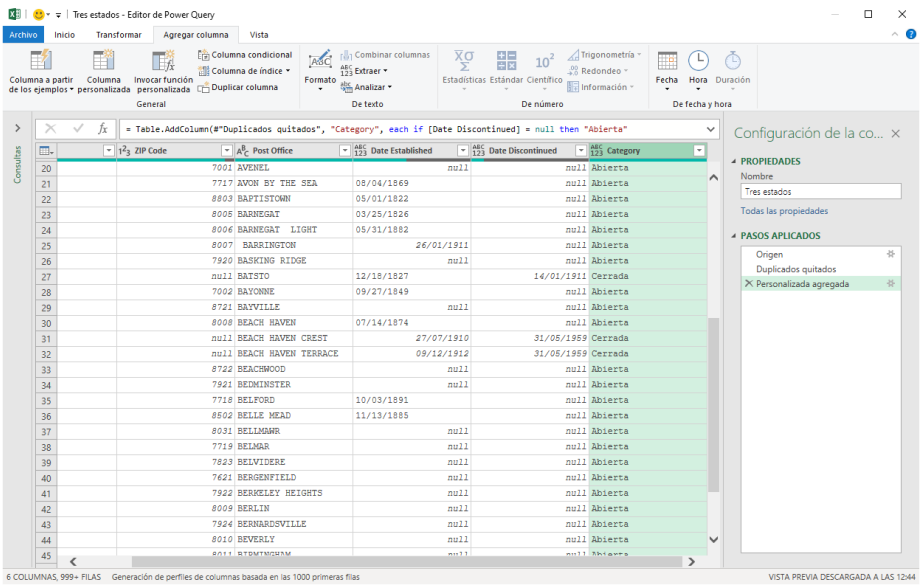


Imagen 2.18 La columna Category ha sido obtenida por una expresión lógica.

26. Renombra el nuevo paso como Agregar columna Category.

Paso 6: Dar forma a los datos

Para llegar al resultado final, solo necesitaremos dos columnas: Estado y Category. Hay varias formas de eliminar columnas de una tabla. Podemos elegir la opción **Eliminar columnas** para eliminar las columnas seleccionadas. También existe otra forma usando el botón **Elegir columnas** en la ficha **Inicio**.

27. Haz clic en la ficha **Inicio > Elegir columnas**, deja activadas State y Category y haz clic en **Aceptar**.

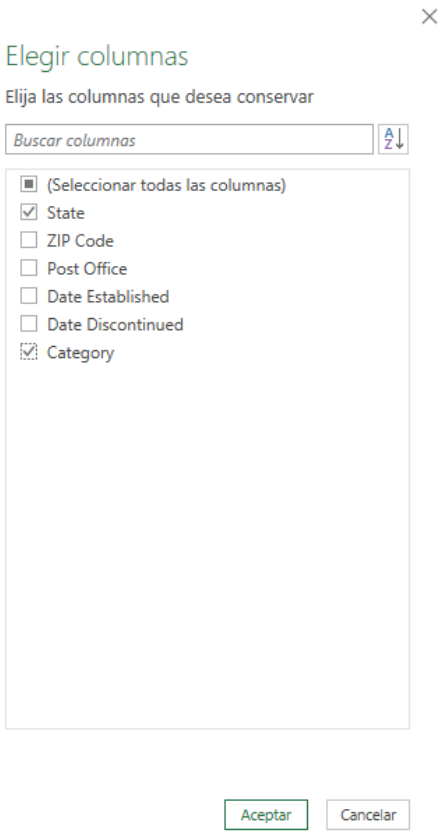


Imagen 2.19 Seleccionando las columnas a mantener.

La tabla se ha reducido ahora a dos columnas. Demos forma a los datos para llegar al resultado final (conocer el número de oficinas abiertas y cerradas) utilizando la opción **Agrupar por**.

28. Haz clic en la ficha **Transformar > Agrupar por** y rellena el cuadro de diálogo como se muestra en la Imagen 2.20. Haz clic en **Aceptar**.

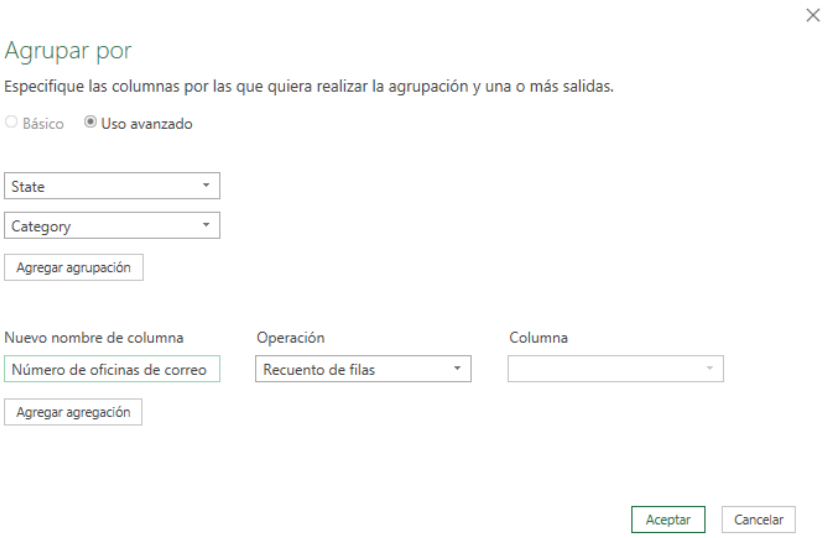


Imagen 2.20 Creando los criterios de agrupación.

El editor de Power Query muestra los datos agrupados como se muestra en la Imagen 2.21

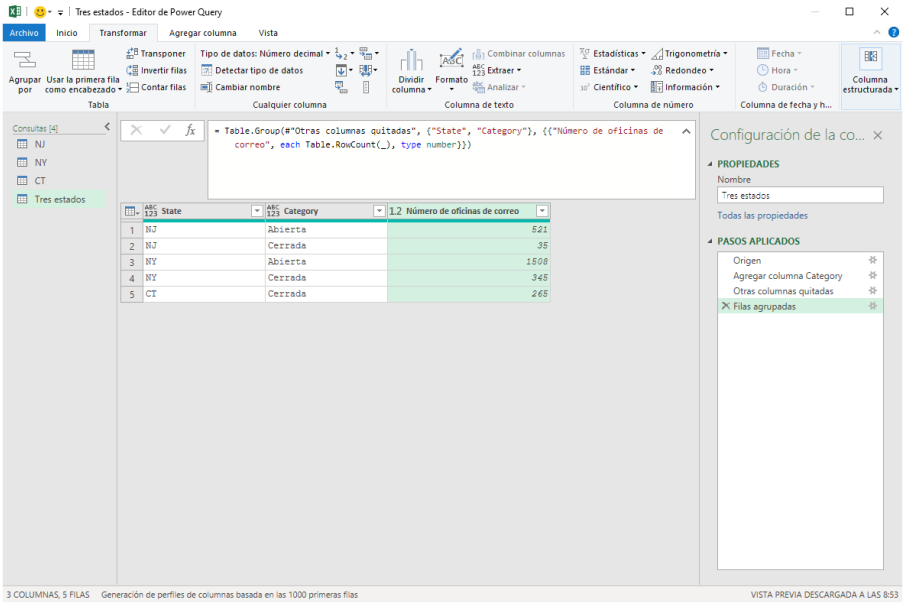


Imagen 2.21 El comando Agrupar por aplicado a una consulta.

Atención

Para agrupar datos en el editor de Power Query, selecciona la columna que quieres agrupar y presiona el botón **Agrupar por** en la ficha **Transformar**. En el cuadro **Agrupar por** (ver Imagen 2.20) es posible agrupar varias columnas seleccionando el botón de opción **Uso avanzado**. En la parte inferior del cuadro podrás elegir la operación de resumen que se debe realizar con las columnas seleccionadas en la parte superior (Recuento de filas, Suma, Promedio, Mín, Máx, etc).

Vamos a darle el toque final al resultado ordenando los datos de forma ascendente:

29. Haz clic en el desplegable del encabezado de la columna State y selecciona **Orden ascendente**.
30. Haz clic en el desplegable del encabezado de la columna Category y selecciona **Orden ascendente**.
    El editor de Power Query muestra los datos ordenados (ver Imagen 2.22).

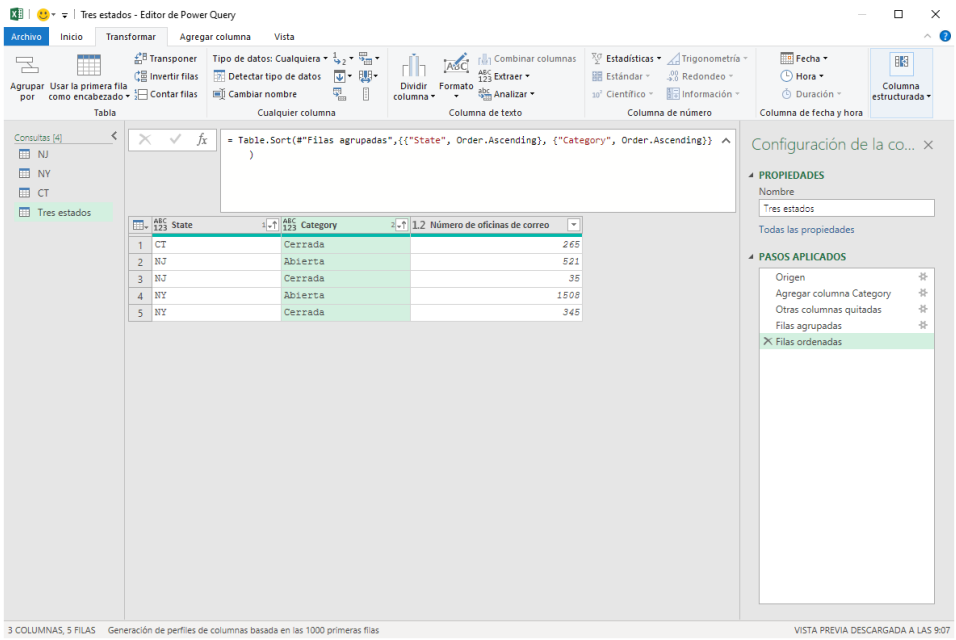


Imagen 2.22 El editor de Power Query muestra los datos finales.

31. Haz clic en la ficha **Archivo > Cerrar y cargar** para cerrar el editor de Power Query y cargar los datos en la hoja.
    La Imagen 2.23 muestra todas las consultas creadas y su resultado.

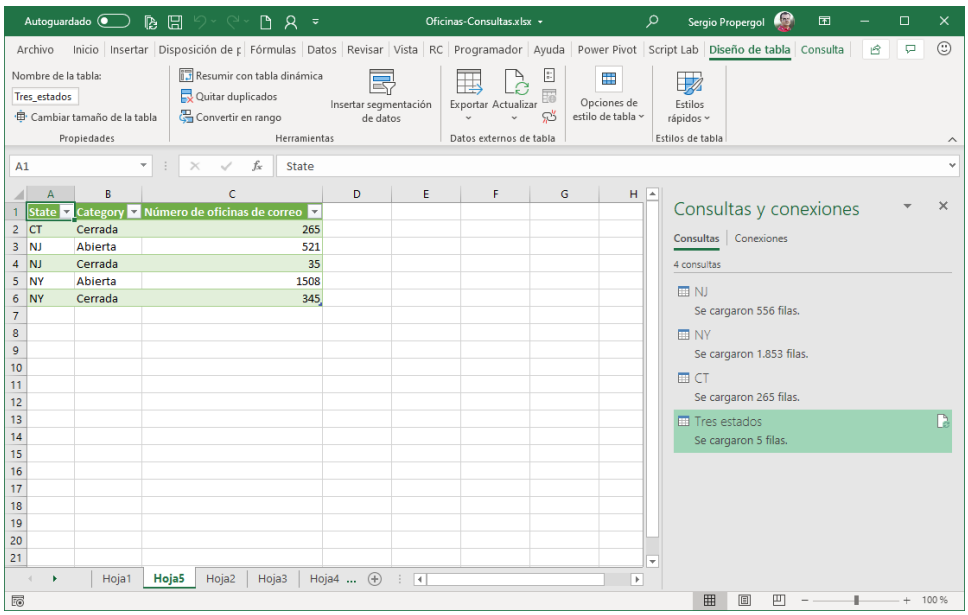


Imagen 2.23 El libro de Excel muestra el resultado final de todos los trabajos de transformación realizados.

3 El editor avanzado

Ahora que hemos creado nuestras primeras consultas con Power Query, vamos a echar un vistazo al código M que el editor ha creado. Podemos obtener una vista general del código accediendo al editor avanzado. Simplemente haz doble clic en la consulta Tres estados desde el libro de Excel y, cuando se muestre el editor de Power Query, haz clic en el botón **Editor avanzado**. La Imagen 3.1 muestra el código de la consulta:

Imagen 3.1 Mirando el código M de la consulta Tres estados en el editor avanzado.

4 Power Query, lenguaje de fórmulas de Excel y VBA

Lamentablemente, el lenguaje M utilizado en las fórmulas de Power Query no tiene ningún parecido con las fórmulas de Excel o el lenguaje VBA. Si pensamos en construir soluciones de datos complejas utilizando Power Query, tendremos que invertir algún tiempo en aprender a utilizar el lenguaje. En el siguiente enlace se hace referencia al lenguaje de fórmulas de Power Query:

https://docs.microsoft.com/es-es/powerquery-m/

La siguiente tabla muestra algunas fórmulas interesantes:

Descripción	Fórmula
Crea una columna personalizada.	`= Excel.Workbook([Content])`
Crea una columna en un archivo CSV.	`= Csv.Document([Content])`
Crea una columna concatenando valores de dos columnas existentes.	`= [NombreColumna1] & " " & [NombreColumna2]`

Descripción	Fórmula
Elimina los dos primeros caracteres del título de la columna.	`= Text.Range([NombreColumna], 2)`

Escribe un valor en una columna en función de una condición.	`= if[NombreColumna]= "V" then "Vacaciones"`
	
	`else "Otros"`

Cambia el nombre de la columna a “Ventas”.	`= Table.RenameColumns(Source,`
	
	`{{"ColumnaAntigua", "Ventas"}})`

Obtiene una lista de encabezados de la tabla.	`= Table.ColumnNames(Source)`
Obtiene el nombre de la segunda columna. Las listas en Power Query comienzan con 0.	`= Table.ColumnNames(Source){1}`

5 Algunas funciones del lenguaje M

Podemos aprender fácilmente algunas funciones del lenguaje M de Power Query simplemente escribiendo una función en la barra de fórmulas y presionando **Intro**. Se mostrará la sintaxis y ejemplos de uso de la función. Por ejemplo, para buscar el uso de la función Table.Combine, escribamos =Table.Combine en la barra de fórmulas y presionemos **Intro**. El resultado se muestra en la Imagen 5.1:

Importante: El lenguaje M es sensible a mayúsculas. Para evitar errores debemos prestar atención a las letras mayúsculas y minúsculas al escribir las expresiones.

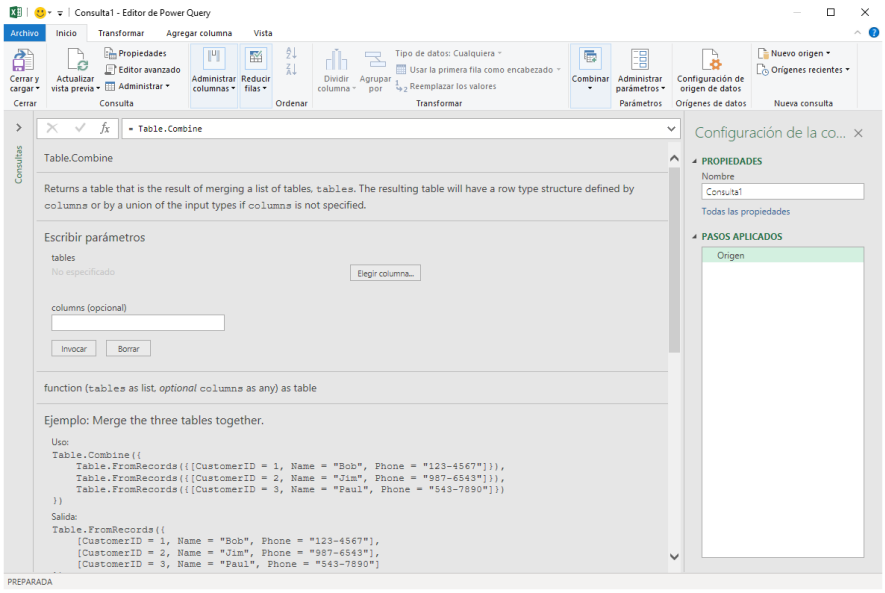


Imagen 5.1 Echando un vistazo a una de las 700 funciones de Power Query introduciendo el nombre en la barra de fórmulas.

Vamos a invertir un par de minutos en examinar la información de algunas funciones de la tabla anterior. Aunque para esto podemos utilizar la barra de fórmulas en cualquier consulta, en el siguiente ejercicio usaremos una consulta en blanco:

1. En la cinta de opciones de Excel haz clic en **Datos > Obtener datos > De otras fuentes > Consulta en blanco**.
   Aparecerá el editor de Power Query listo para recibir instrucciones. El panel **Pasos aplicados** muestra un solo paso llamado Origen, que se encuentra vacío. Si abres el editor avanzado en este momento encontrarás el siguiente script:

```
let
    Origen = ""
in
    Origen
```

2. En la barra de fórmulas de Power Query escribe "=Text.Range" y presiona **Intro**.
   El panel de datos mostrará la información sobre la función requerida. Verás también un botón llamado **Invocar**, que permite probar la función por ti mismo.
3. Introduce los parámetros para la función y presiona en **Invocar**. Puedes usar las palabras "Hola mundo" y extraer la palabra comenzando en el índice 6 como se muestra en la descripción de la función.
   El editor de Power Query ahora muestra el resultado de la función introducida (ver Imagen 5.2). Observa que el panel izquierdo de la ventana contiene una propiedad llamada **Función invocada** y la barra de fórmulas muestra el siguiente código:

= Consulta1("Hola mundo", 6, null)

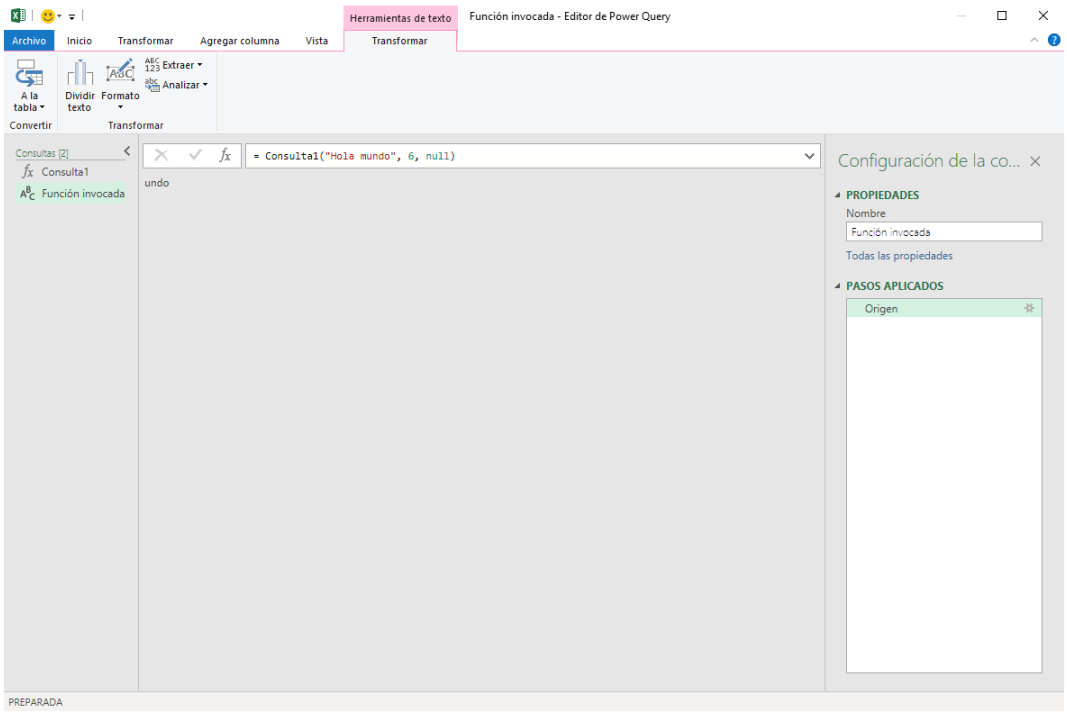


Imagen 5.2 El resultado de invocar una función en Power Query.

Cuando activas el editor avanzado, se muestra lo siguiente:

let

Origen = Consulta1("Hola mundo", 6, null)

in

Origen

Puedes continuar probando otras funciones en la barra de fórmulas e invocándolas para ver el resultado. Puedes usar una consulta en blanco para probar este tipo de experimentos, como hacías con la ventana **Inmediato** de VBA.

4. Selecciona la consulta Consulta1 en el panel de la izquierda y cámbiale el nombre a "Funciones de prueba".
5. Utiliza el botón **Cerrar y cargar** para guardar los cambios. Excel crea la conexión **Funciones de prueba** que se muestra en la Imagen 5.3.

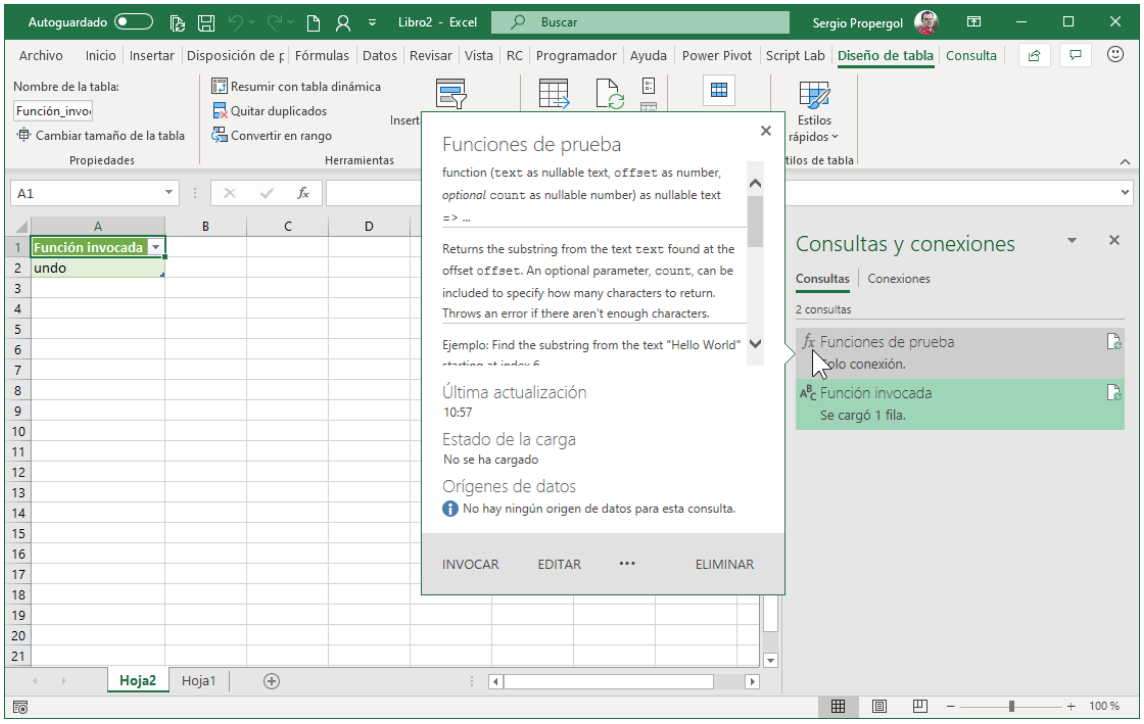


Imagen 5.3 La consulta creada no contiene datos, así que Excel solo crea la conexión.

6 Crear una consulta desde una tabla

Además de crear consultas desde fuentes de datos externas, podemos usar Power Query para trabajar con tablas que se encuentran en el mismo libro. Únicamente debes hacer clic en la ficha **Datos > Obtener datos > Desde otras fuentes > Desde una tabla o rango**. Si el rango seleccionado no forma parte de una tabla, se convertirá automáticamente en una.

7 Power Query y VBA

Para dar soporte a Power Query, el modelo de objetos de VBA ofrece los objetos `Queries` y `WorkbookQuery` con sus propiedades y métodos (Imagen 7.1 e Imagen 7.2). Podemos probar algunas de estas propiedades en la ventana **Inmediato** como se muestra a continuación. Antes de eso, asegurémonos de que el libro Oficinas-Consultas.xlsx esté activo.

```
?ThisWorkbook.Queries.Count
 4
?ThisWorkbook.Queries.Parent.Name
Oficinas-Consultas.xlsx
?ThisWorkbook.Queries.Item(1).Name
NJ
```

Además de la compatibilidad del modelo de objetos con Power Query, también podemos utilizar la grabadora de macros para automatizar el proceso de creación y actualización de consultas. Sin embargo, no es posible grabar las acciones realizadas en el editor de consultas de Power Query.

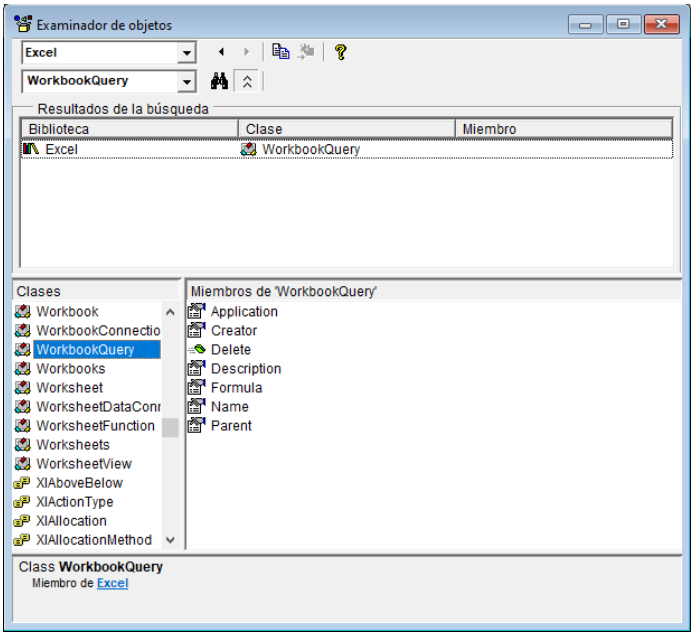


Imagen 7.1 Propiedades y métodos del objeto WorkbookQuery.

1. Crea un libro nuevo de Excel haciendo clic en **Archivo > Nuevo > Libro en blanco**.
2. Haz clic en la ficha **Programador** (Desarrollador) y a continuación en **Grabar macro**.
3. En el cuadro de diálogo **Grabar macro** escribe `CrearConsulta` como nombre de la macro, selecciona “En este libro” para almacenar la macro y presiona en **Aceptar**.
4. Haz clic en **Obtener datos > Desde la web**.
5. Introduce la URL https://livingwage.mit.edu/states/06 y haz clic en **Aceptar**.
6. En la ventana **Navegador** haz clic en **Seleccionar varios elementos** y selecciona las tres tablas como se muestra en la Imagen 7.3.

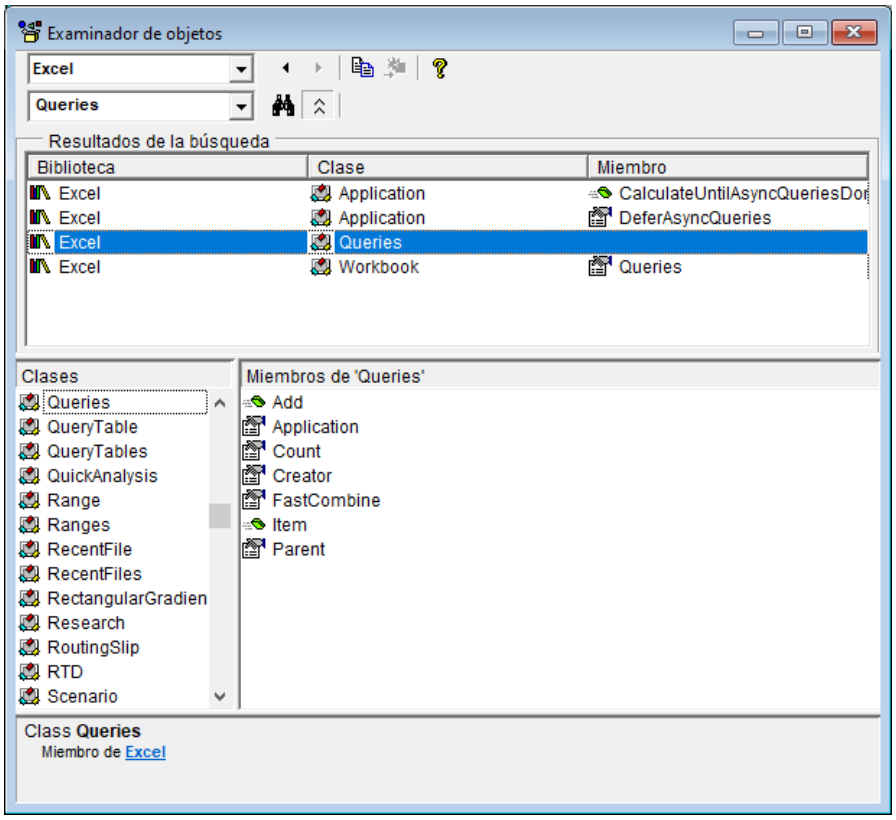


Imagen 7.2 Propiedades y métodos del objeto Queries.

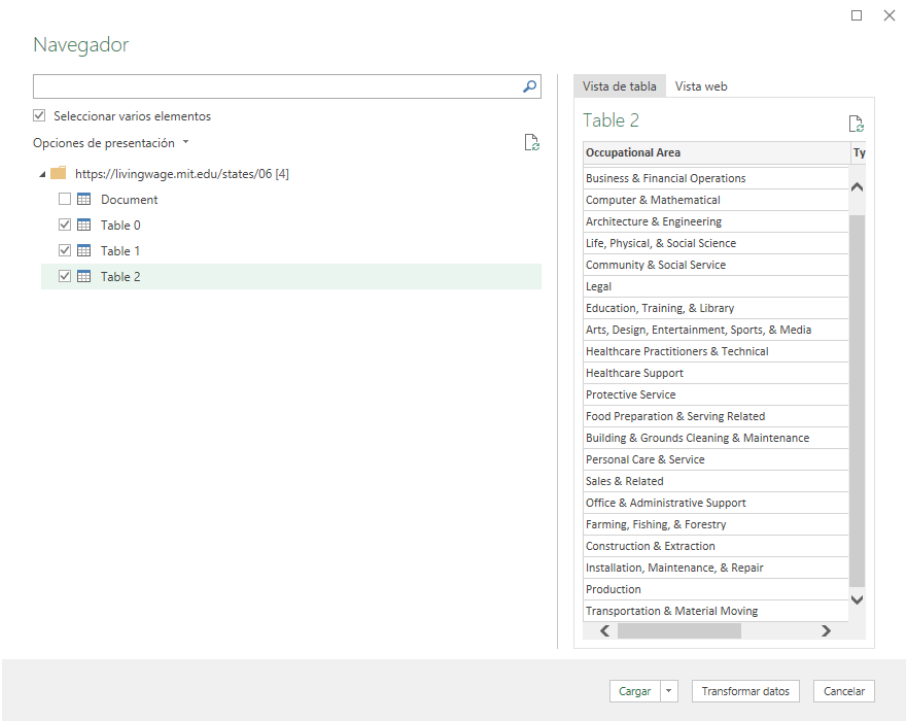
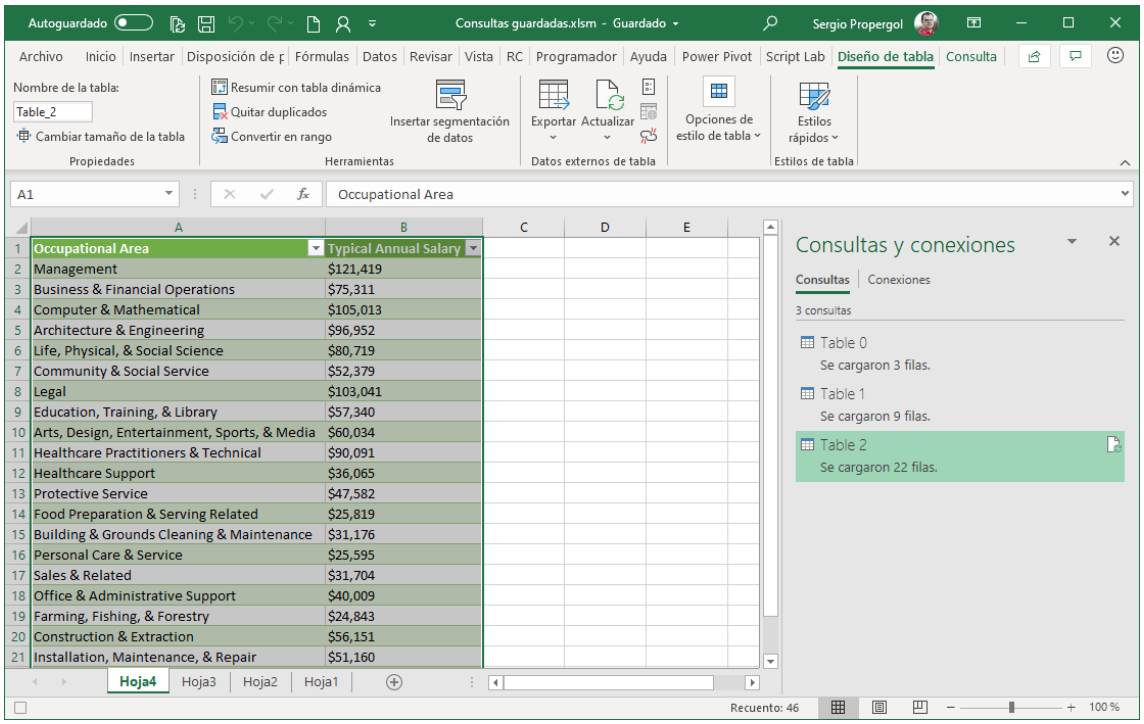


Imagen 7.3 Seleccionando los datos para grabar una consulta con la grabadora de macros.

7. Haz clic en el desplegable del botón **Cargar** y selecciona **Cargar en**.
8. En el cuadro **Importar datos** selecciona **Tabla** y desactiva la casilla **Agregar estos datos al modelo de datos** en caso de que esté seleccionada.
   Se cargarán las tres tablas en el libro, como muestra la Imagen 7.4.

9. Detén la grabación haciendo clic en la ficha **Programador** y presionando el botón **Parar grabación**.
10. Guarda el libro como Consultas grabadas.xlsm.



	A	B	C	D	E
1	Occupational Area	Typical Annual Salary			
2	Management	$121,419			
3	Business & Financial Operations	$75,311			
4	Computer & Mathematical	$105,013			
5	Architecture & Engineering	$96,952			
6	Life, Physical, & Social Science	$80,719			
7	Community & Social Service	$52,379			
8	Legal	$103,041			
9	Education, Training, & Library	$57,340			
10	Arts, Design, Entertainment, Sports, & Media	$60,034			
11	Healthcare Practitioners & Technical	$90,091			
12	Healthcare Support	$36,065			
13	Protective Service	$47,582			
14	Food Preparation & Serving Related	$25,819			
15	Building & Grounds Cleaning & Maintenance	$31,176			
16	Personal Care & Service	$25,595			
17	Sales & Related	$31,704			
18	Office & Administrative Support	$40,009			
19	Farming, Fishing, & Forestry	$24,843			
20	Construction & Extraction	$56,151			
21	Installation, Maintenance, & Repair	$51,160			

Imagen 7.4 Se han generado tres tablas después de seleccionarlas en la ventana Navegador.

Vamos a abrir ahora el editor de VBA para examinar el código grabado (ver Imagen 7.5).

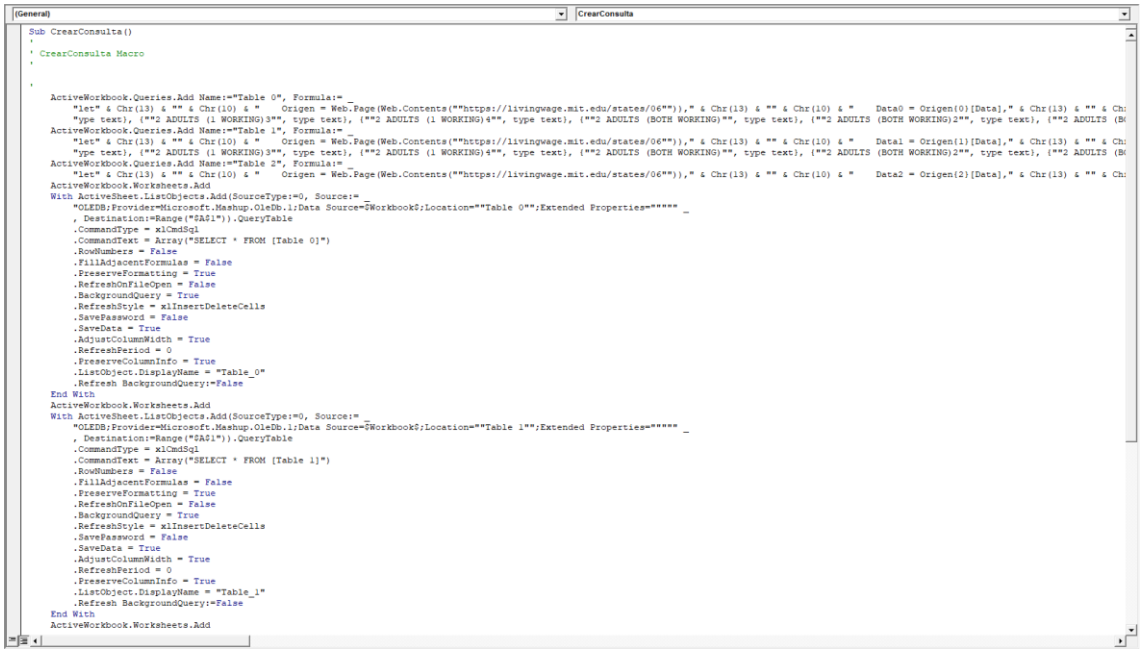


Imagen 7.5 Creación de consultas automatizadas con la grabadora de macros.

Observemos que el método `Queries.Add` se usa para añadir un nuevo objeto `WorkbookQuery` a la colección `Queries`. Este método requiere que le demos el nombre de la

consulta y la fórmula M. También podemos introducir una descripción, pero es opcional. La función `Web.Contents` se usa para descargar datos de la web. Esta función necesita la URL en forma de cadena de texto. La función `Web.Page` devuelve el contenido de la página HTML en forma de tabla. La función `Table.TransformColumnTypes` modifica el tipo de datos de las columnas.

Para la conexión se usa el proveedor `Microsoft.Mashup.OleDb.1`. Además del proveedor, la cadena de conexión incluye `$Workbook` como valor del argumento `Data Source` y el nombre de la consulta en el argumento `Location`. El método `QueryTable.Refresh` actualiza la `QueryTable`. El argumento opcional `BackgroundQuery` especifica si la consulta se debe actualizar en segundo plano. El valor `False` devuelve el control al procedimiento solo después de que todos los datos se hayan ubicado en la hoja. Con el valor `True`, el control se devuelve al procedimiento tan pronto como se haya establecido la conexión con la base de datos y la consulta se haya enviado.

Si utilizamos la grabadora de macros para grabar consultas usando diferentes opciones podremos descubrir otras funciones útiles para automatizar Excel.

8 Otros recursos de aprendizaje de Power Query

En este capítulo hemos visto solo algunos ejemplos simples de Power Query, pues el contenido de este manual no lo aborda. Para obtener más información sobre los procesos de extracción y transformación de datos, aquí van unas sugerencias:

- **Power Query 101**
  Este tutorial muestra como conectarse a una fuente de datos web, seleccionar tablas para importar, reemplazar y filtrar valores y, finalmente, cargar la consulta en una hoja del libro.
  https://support.microsoft.com/es-es/office/power-query-101-008b3f46-5b14-4f8b-9a07-d3da689091b5
- **Combinar datos de varias fuentes**
  Este tutorial muestra cómo combinar un archivo local de Excel con un feed de OData, realizar agregaciones para generar un informe de ventas totales por producto y año.
  https://support.office.com/es-es/article/combinar-datos-de-varios-or%C3%ADgenes-de-datos-power-query-70cfe661-5a2a-4d9d-a4fe-586cc7878c7d

9 Resumen

En este capítulo se ofrece una breve introducción a la importación de datos y a las transformaciones de datos conocidas como Power Query (“Obtener y transformar” en algunas versiones de Excel).

Después de importar y combinar los datos de un libro de Excel y un archivo de texto csv, aprendiste a transformar los datos en bruto mediante una serie de pasos para generar una tabla resumen de Excel. Aprendiste a utilizar el editor de consultas de Power Query para editar los pasos que se generaban dinámicamente mediante varios botones de comando en la cinta del editor. También aprendiste utilizar la barra de fórmulas del editor para obtener

información y probar expresiones y funciones de código M, y cómo el editor avanzado muestra todos los pasos de la consulta y permite editarlos de forma manual. También descubriste algunas características del lenguaje M, como que diferencia entre mayúsculas y minúsculas, los tipos de datos y las expresiones lógicas. Finalmente aprendiste cómo la grabación de macros y el modelo de objetos de Excel pueden ayudarte a escribir procedimientos VBA que automaticen el proceso de creación y actualización de consultas.

El siguiente capítulo se centra en la programación del editor de VBA.

Capítulo 24
Gestión del editor de VBA

Trabajando en capítulos anteriores del manual, has adquirido un conocimiento práctico de muchas de las herramientas disponibles en el editor de VBA para crear, modificar y solucionar problemas de VBA en los procedimientos. VBA también te permite programar tu propio entorno de desarrollo. Por ejemplo, es posible:

- Controlar los proyectos de VBA.
    - Obtener o establecer las propiedades del proyecto.
    - Agregar o quitar componentes individuales.
- Controlar el código de VBA.
    - Agregar, eliminar y modificar código.
    - Guardar el código en un archivo o insertar el código desde un archivo.
    - Buscar información específica en el código.
- Controlar formularios.
    - Diseñar formularios mediante programación.
    - Agregar o quitar controles de formulario dinámicamente.
- Trabajar con referencias.
    - Añadir una referencia a una biblioteca de objetos externos.
    - Comprobar si hay referencias rotas.
- Controlar el editor de VBA.
    - Trabajar con varias ventanas.
    - Agregar o cambiar menús y barras de herramientas.

Este capítulo te introduce a los objetos, métodos y propiedades disponibles para automatizar tu entorno de programación VBA.

1 El modelo de objetos del editor de VBA

Para programar y manipular el editor de VBA con código necesitamos acceder a algunos objetos contenidos en la biblioteca de **Microsoft Visual Basic for Applications Extensibility 5.3**. Para asegurarnos de que podemos ejecutar los procedimientos de este capítulo, realiza los siguientes pasos:

1. Crea un nuevo libro de Excel en **C:\Archivos Manual VBA** y llámalo **Capítulo 24 – Editor VBA.xlsm**.
2. Para confiar en el proyecto de VBA, haz clic en la ficha **Programador** (Desarrollador) y a continuación haz clic en **Seguridad de macros**. Excel mostrará el cuadro **Centro de confianza**. Haz clic en **Configuración de macros** para ver algunas opciones. Activa la casilla de verificación **Confiar en el acceso al modelo de objetos de proyectos de VBA**.

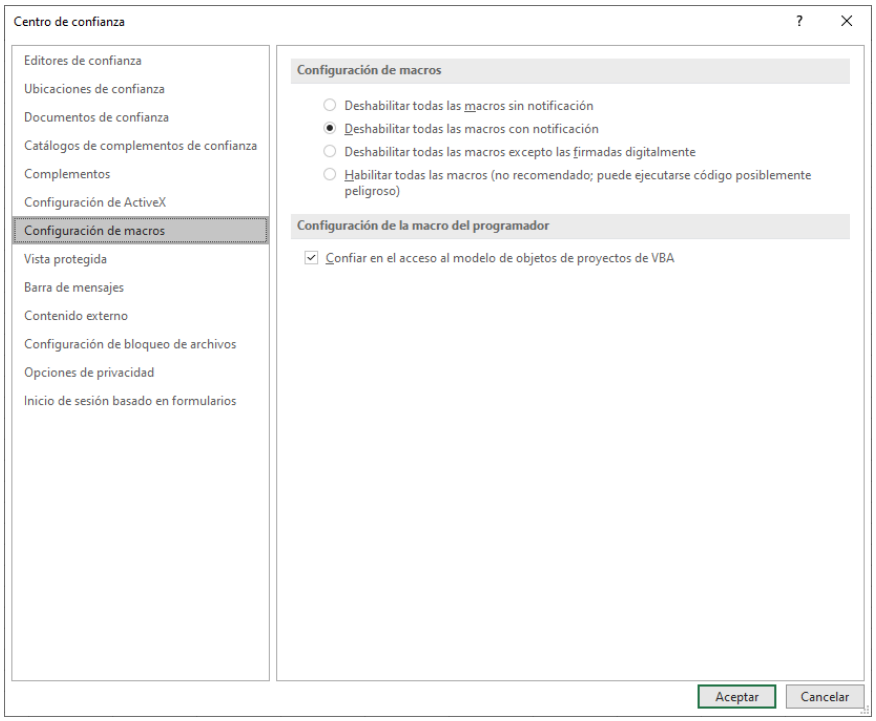


Imagen 1.1 Debemos dar acceso al modelo de objetos del proyecto VBA para permitir la modificación del editor.

3. Haz clic en el botón **Visual Basic** en la ficha **Programador** para activar el editor de VBA. En la ventana **Propiedades**, cambia el nombre al proyecto por `Cap24CodigoFuente`.
4. Haz clic en el menú **Herramientas – Referencias**. Activa la casilla de Microsoft Visual Basic for Applications Extensibility 5.3, como se muestra en la Imagen 1.2 y haz clic en **Aceptar**.

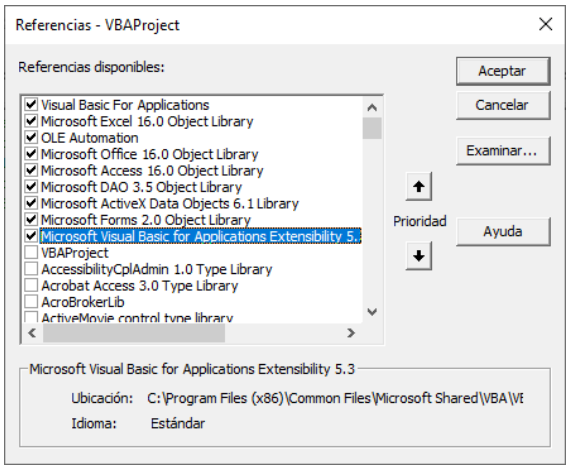


Imagen 1.2 Estableciendo la referencia para permitir la modificación del entorno de programación.

2 Los objetos del editor de VBA

En el **Examinador de objetos**, la biblioteca de Microsoft Visual Basic for Applications Extensibility 5.3 aparece como VBIDE. Usaremos este nombre al hacer referencia en el código a la biblioteca.

El objeto de mayor nivel dentro de este modelo de objetos es el VBE, que representa al editor por sí mismo.

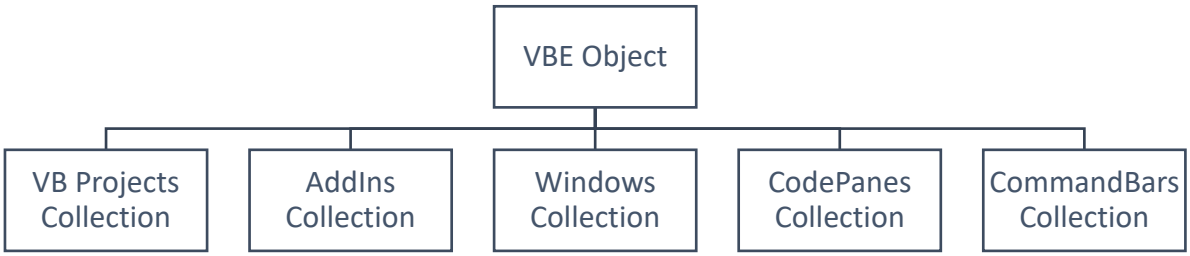


Imagen 2.1 El modelo de objetos VBE contiene cinco colecciones de objetos.

El modelo de objetos contiene las siguientes cinco colecciones de objetos:

- **VBProjects**: Contiene todos los objetos `VBProject` abiertos actualmente en el entorno de desarrollo. Se utilizamos para establecer propiedades al proyecto. También nos permite acceder a las colecciones `VBComponents` y `References`.
    - **VBComponents**: se usa para acceder, añadir o eliminar componentes a un proyecto. Por componente se entienden formularios, módulos estándar o módulos de clase.
    - **References**: Se usa para añadir o eliminar referencias a un proyecto VBA. Cada proyecto puede hacer referencia a una o más bibliotecas. También se utiliza para conocer qué referencias se encuentran seleccionadas en cierto momento en el cuadro de diálogo Referencias, en un proyecto VBA específico.
- **AddIns**: Permite acceder a los objetos `AddIn`. Los `AddIns` (o complementos) son aplicaciones o utilidades que aumentan las funcionalidades de Excel o de otros productos de Office.
- **Windows**: Se usa para acceder a las ventanas del editor, como el Explorador de proyectos, la ventana Propiedades o las ventanas Código que estén abiertas.
- **CodePanes**: Se usa para abrir paneles de código en un proyecto. Una ventana Código puede contener uno o más paneles.
- **CommandBars**: Contiene todas las barras de herramientas en un proyecto, incluyendo los menús contextuales.

3 Acceso al proyecto VBA

La única forma de conocer si hemos activado la casilla de verificación en el Centro de confianza es mediante la captura de errores (se habló de este tema en el Capítulo 9). El siguiente procedimiento muestra un mensaje si el acceso al proyecto VBA no es de confianza (no se ha activado la casilla). Las instrucciones para cambiar la configuración de seguridad se muestran en un cuadro de texto colocado en un nuevo libro de Excel.

1. En el editor de VBA del libro Capítulo 24 – Editor VBA.xlsm inserta un módulo nuevo.
2. En la ventana Código, Introduce el siguiente procedimiento:

```
Sub AccesoProyecto()
    Dim objVBProject As VBProject
    Dim strMsg1 As String
    Dim strMsg2 As String
    Dim response As Integer

    On Error Resume Next
    If Application.Version >= "16.0" Then
        Set objVBProject = ActiveWorkbook.VBProject
        strMsg2 = "El acceso al proyecto VBA "
        strMsg2 = strMsg2 + "debe ser habilitado siguiendo "
        strMsg2 = strMsg2 + "las siguientes instrucciones."
        strMsg2 = strMsg2 + vbCrLf + vbCrLf
        strMsg2 = strMsg2 + _
        " Haz clic en 'Aceptar' para ver las instrucciones,"
        strMsg2 = strMsg2 + " o en 'Cancelar' para salir."
        If Err.Number <> 0 Then
            strMsg1 = _
            "Cambia la configuración de seguridad para "
            strMsg1 = strMsg1 & _
            "permitir el acceso al proyecto VBA:"
            strMsg1 = strMsg1 & Chr(10) & "1. "
            strMsg1 = strMsg1 & _
            "Haz clic en Programador > Seguridad de macros."
            strMsg1 = strMsg1 & Chr(10) & "2. "
            strMsg1 = strMsg1 & _
            "Activa la casilla 'Confiar en el acceso al  " _
            & "modelo de objetos de proyectos de VBA'. "
            strMsg1 = strMsg1 & Chr(10) & _
            "3. Haz clic en Aceptar."
            response = MsgBox(strMsg2,vbCritical + vbOKCancel, _
            "Acceso al proyecto no permitido")
            If response = 1 Then
                Workbooks.Add
                With ActiveSheet
                    .Shapes.AddTextbox _
                    (msoTextOrientationHorizontal, _
                    Left:=0, Top:=0, Width:=300, _
                    Height:=100).Select
                    Selection.Characters.Text = strMsg1
```

```
                    .Shapes(1).Fill.PresetTextured _
                    PresetTexture:=msoTextureBlueTissuePaper
                    .Shapes(1).Shadow.Type = msoShadow6
                End With
            End If
            Exit Sub
        End If
        MsgBox "Hay " & objVBProject.References.Count & _
        " referencias a proyectos en " & objVBProject.Name & "."
    End If
End Sub
```

El procedimiento comienza comprobando la versión de Excel del libro. Si la casilla de verificación del Centro de confianza se encuentra desactivada, no se puede dar un valor a la variable `objVBProject` causando un error. El procedimiento captura este error con la instrucción `On Error Resume Next`. Si se produce un error, el objeto `Err` devolverá un valor diferente a cero. En este momento podemos decirle al usuario que la configuración de seguridad debe ajustarse para que el procedimiento pueda ejecutarse. En vez de comunicárselo mediante un cuadro de mensaje, se muestra en un libro nuevo para poder seguir los pasos fácilmente (ver Imagen 3.1).

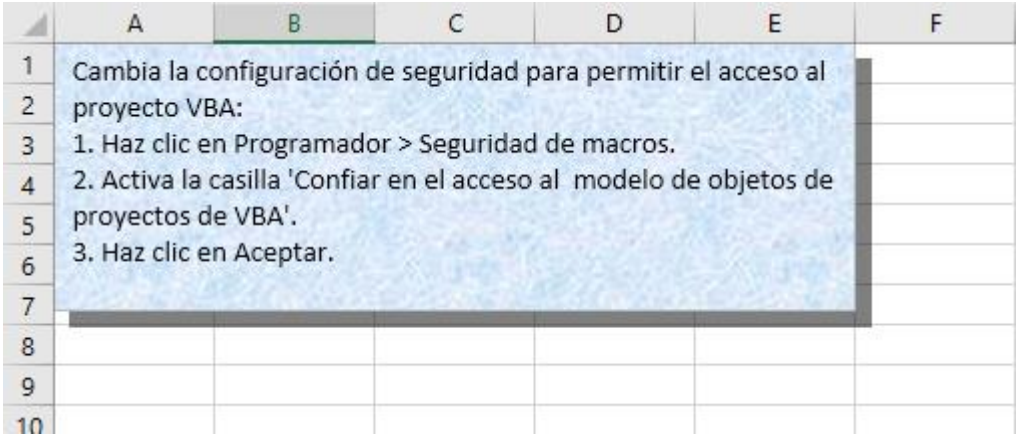


Imagen 3.1 Instrucciones para permitir el acceso al modelo de objetos VBA generado por el procedimiento.

4 Buscar información sobre un proyecto VBA

Como ya sabemos, todos los libros de Excel tienen un proyecto VBA llamado `VBAProject`. Podemos cambiar el nombre del proyecto a otro más descriptivo, introduciendo un valor en la propiedad `Name` de la ventana **Propiedades** o accediendo al cuadro de diálogo **Propiedades de** `VBAProject` desde el menú **Herramientas**. También es posible hacer este cambio con un procedimiento. Si el proyecto que deseamos editar está actualmente seleccionado en la ventana del **Explorador de proyectos**, simplemente escribiremos la siguiente declaración en la ventana **Inmediato** para reemplazar el nombre predeterminado por otro personalizado:

```
Application.VBE.ActiveVBProject.Name = "Cap24CodigoFuente"
```

En caso de que el proyecto VBA que queremos cambiar no se encuentre activo, podemos usar la siguiente instrucción:

```
Workbooks("Capítulo 24 – Editor VBA.xlsm").VBProject.Name = _
"Cap24CodigoFuente"
```

Si lo que queremos es cambiar la descripción del proyecto, podemos escribir la siguiente instrucción en la ventana Inmediato (en una sola línea).

```
Workbooks("Capítulo 24 – Editor VBA.xlsm").VBProject.Description
= "Programando el editor de VBA"
```

También podemos conocer si el proyecto se ha guardado utilizando la propiedad `Saved` del objeto `VBProject`:

```
MsgBox Application.VBE.ActiveVBProject.Saved
```

VBA devuelve `False` si los cambios en el proyecto no se han guardado.

Para conocer el número de objetos (componentes) que contiene un proyecto de VBA específico usamos el siguiente código:

```
MsgBox Workbooks("Capítulo 24 – Editor VBA.xlsm").VBProject _
.VBComponents.Count
```

Y para conocer el nombre del objeto `VBComponent` seleccionado en cierto momento, escribiremos las siguientes líneas en la ventana inmediato, presionando Intro después de cada una:

```
Set objVBComp = Application.VBE.SelectedVBComponent
MsgBox objVBComp.Name
```

También podemos conocer rápidamente el número de referencias definidas en el cuadro de diálogo Referencias escribiendo la siguiente instrucción en la ventana **Inmediato**:

```
?Application.VBE.ActiveVBProject.References.Count
```

5 Protección del Proyecto VBA

Para evitar que otras personas puedan ver el código que hemos escrito, podemos bloquear los proyectos VBA. Para bloquear un proyecto, necesitamos realizar las siguientes tareas:

1. En el **Explorador de proyectos**, haz doble clic en el nombre del proyecto a proteger y a continuación, haz clic en **Propiedades de [Nombre del proyecto]** en el menú contextual.
2. En el cuadro de diálogo **Propiedades**, haz clic en la pestaña **Protección** y activa la casilla **Bloquear el proyecto para visualización**. Introduce una contraseña y confírmala. A continuación, haz clic en **Aceptar**.
   La próxima vez que abras el libro se te pedirá introducir la contraseña para ver el código del proyecto. No existe una forma de especificar mediante programación una contraseña para bloquear un proyecto VBA, pero sí para determinar si se encuentra bloqueado o no. Para ello se comprueba la propiedad `Protection` del objeto

`VBProject`. El siguiente procedimiento `Function` muestra cómo comprobar la propiedad `Protection` del proyecto VBA en un libro de Excel.

1. En el mismo módulo del procedimiento anterior introduce el siguiente:

```
Function ProyProtegido() As Boolean
    Dim objVBProj As VBProject

    Set objVBProj = ActiveWorkbook.VBProject
    If objVBProj.Protection = vbext_pp_locked Then
        ProyProtegido = True
    Else
        ProyProtegido = False
    End If
End Function
```

2. Para probar la función anterior escribe `MsgBoxProyProtegido()` en la ventana **Inmediato** y presiona **Intro**.
   La función `MsgBox` muestra `False` cuando un proyecto no está protegido y `True` en caso de que sí lo esté.

6 Trabajar con módulos

Los módulos estándar, los módulos de clase y los módulos de objeto (`Thisworkbook`, `Worksheet`, `UserForm`) son miembros de la colección `VBComponents` del objeto `VBProject`. Para determinar de qué tipo de objeto se trata usamos la propiedad `Type` del objeto `VBComponent`. El código de cada `VBComponent` se almacena en un `CodeModule`.

Las siguientes secciones muestran algunos procedimientos que acceden a la colección `VBComponents` para realizar algunas acciones:

- Enumerar todos los módulos de un libro.
- Agregar un módulo.
- Eliminar un módulo.
- Eliminar el código de un módulo.
- Eliminar módulos vacíos.
- Copiar módulos (importación y exportación).

6.1 Enumerar todos los módulos de un libro

El siguiente procedimiento genera una lista de todos los módulos que contiene el libro **Capítulo 24 – Editor VBA.xlsm**. El nombre de los módulos y su tipo se almacenan en una matriz bidimensional y luego se escriben en una hoja. Como la propiedad `Type` del objeto `VBComponent` devuelve una constante o un valor numérico que determina el tipo de objeto, el procedimiento usa una función para mostrar la correspondiente descripción del objeto.

1. Inserta un módulo nuevo en el proyecto VBA del archivo **Capítulo 24 – Editor VBA.xlsm**.

2. En la ventana **Código**, introduce los siguientes procedimientos `Sub` y `Function`:

```
Sub EnumeraModulos()
    Dim objVBComp As VBComponent
    Dim listArray()
    Dim i As Integer

    If ThisWorkbook.VBProject.Protection = vbext_pp_locked Then
        MsgBox "Por favor, desprotege el proyecto " & _
        "para ejecutar el procedimiento."
        Exit Sub
    End If
    i = 2
    For Each objVBComp In ThisWorkbook.VBProject.VBComponents
        ReDim Preserve listArray(1 To 2, 1 To i - 1)
        listArray(1, i - 1) = objVBComp.Name
        listArray(2, i - 1) = TipoModulo(objVBComp)
        i = i + 1
    Next
    With ActiveSheet
      .Cells(1, 1).Resize(1, 2).Value = Array("Nombre Módulo", _
        "Tipo")
        .Cells(2, 1).Resize(UBound(listArray, 2), _
        UBound(listArray, _
        1)).Value = Application.Transpose(listArray)
        .Columns("A:B").AutoFit
    End With
    Set objVBComp = Nothing
End Sub
Function TipoModulo(comp As VBComponent)
    Select Case comp.Type
        Case vbext_ct_StdModule
            TipoModulo = "Módulo estándar"
        Case vbext_ct_ClassModule
            TipoModulo = "Módulo de clase"
        Case vbext_ct_MSForm
            TipoModulo = "Microsoft Form"
        Case vbext_ct_ActiveXDesigner
            TipoModulo = "Formulario"
        Case vbext_ct_Document
            TipoModulo = "Módulo de documento"
```

```
            Case Else
                TipoModulo = "Unknown"
        End Select
    End Function
```

3. Ejecuta el procedimiento `EnumeraModulos` y activa la ventana de Excel para ver el resultado.
   Si el proyecto VBA está protegido al ejecutar el procedimiento, recibirás un mensaje de aviso.

6.2 Agregar un módulo a un libro

Se usa el método `Add` de la colección `VBComponents` para agregar un nuevo módulo al libro. El procedimiento `CreaModulo` mostrado abajo, pregunta al usuario por el nombre y el tipo de módulo. Cuando se ha proporcionado esta información, se llama al procedimiento `CreaModulo`

1. Crea un módulo nuevo en el libro **Capítulo 24 – Editor VBA.xlsm**.
2. En la ventana **Código** introduce los siguientes procedimientos:

```
Sub CreaModulo()
    Dim modType As Integer
    Dim strName As String
    Dim strPrompt As String

    strPrompt = _
    "Introduce un número que represente el tipo de módulo:"
    strPrompt = strPrompt & vbCr & "1 (Módulo estándar)"
    strPrompt = strPrompt & vbCr & "2 (Módulo de clase)"
    modType = Val(InputBox(prompt:=strPrompt, _
    Title:="Insertar módulo"))
    If modType = 0 Then Exit Sub
   strName = InputBox("Introduce el nombre que desees para " & _
    "el nuevo módulo", "Nombre del módulo")
    If strName = "" Then Exit Sub
    AgregaModulo modType, strName
End Sub
Sub AgregaModulo(modType As Integer, strName As String)
    Dim objVBProj As VBProject
    Dim objVBComp As VBComponent
    If InStr(1, "1, 2", modType) = 0 Then Exit Sub
    Set objVBProj = ThisWorkbook.VBProject
    Set objVBComp = objVBProj.VBComponents.Add(modType)
    objVBComp.Name = strName
```

```
    Application.Visible = True
    Set objVBComp = Nothing
    Set objVBProj = Nothing
End Sub
```

3. Ejecuta el procedimiento `CreaModulo`. Introduce un *1* para crear un módulo estándar cuando se solicite y presiona **Aceptar**. Introduce `ModuloPrueba` como nombre del módulo y presiona **Aceptar**. Cuando el procedimiento finalice la ejecución deberías ver el módulo nuevo en el Explorador de proyectos. No lo elimines, lo necesitarás para el próximo ejercicio.

6.3 Eliminar un módulo

Utiliza el siguiente procedimiento para eliminar el módulo que acabas de agregar.

1. Crea un módulo nuevo e introduce el siguiente procedimiento:

```
Sub BorraModulo(strName As String)
    Dim objVBProj As VBProject
    Dim objVBComp As VBComponent

    Set objVBProj = ThisWorkbook.VBProject
    Set objVBComp = objVBProj.VBComponents(strName)
    objVBProj.VBComponents.Remove objVBComp
    Set objVBComp = Nothing
    Set objVBProj = Nothing
End Sub
```

2. Ejecuta el procedimiento escribiendo la siguiente instrucción en la ventana **Inmediato** y presionando **Intro**:

```
BorraModulo"ModuloPrueba"
```

6.4 Eliminar todo el código de un módulo

Usaremos la propiedad `CountOfLines` del objeto `CodeModule` para conocer el número de líneas de código en un módulo. Para borrar una o varias líneas de código usaremos el método `DeleteLines` del objeto `CodeModule`. El primer argumento especifica la primera línea a eliminar y es obligatorio. El segundo argumento (opcional) especifica el número total de líneas que deseamos borrar. Si no indicamos este número, sólo se eliminará una línea. El siguiente procedimiento elimina todo el código del módulo especificado:

1. Inserta un módulo nuevo.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub BorraCodigoModulo(strName As String)
    Dim objVBProj As VBProject
    Dim objVBCode As CodeModule
    Dim firstLn As Long
```

```
    Dim totLn As Long

    Set objVBProj = ThisWorkbook.VBProject
    Set objVBCode = objVBProj.VBComponents(strName).CodeModule
    With objVBCode
        firstLn = 1
        totLn = .CountOfLines
        .DeleteLines firstLn, totLn
    End With
    Set objVBProj = Nothing
    Set objVBCode = Nothing
End Sub
```

3. Inserta un módulo nuevo y llámalo `PruebaBorrar`.
4. Copia el primer procedimiento del capítulo en el módulo `PruebaBorrar`.
5. En la ventana **Inmediato**, introduce la siguiente instrucción:

```
BorraCodigoModulo"PruebaBorrar"
```

Cuando presionas **Intro**, todo el código de `PruebaBorrar` se elimina. No borres el módulo vacío. Lo eliminaremos con VBA en el siguiente ejemplo.

6.5 Eliminar módulos vacíos

A lo largo de un proyecto de VBA, hemos podido insertar un gran número de módulos nuevos. Aunque la mayoría de estos módulos contendrán código válido, habrá probablemente alguno que se haya quedado en blanco. Podemos eliminar estos módulos vacíos de una pasada con un procedimiento VBA. Para eliminar un módulo, usamos el método `Remove` de la colección `VBComponets`. El siguiente procedimiento recorre la colección `VBComponents` y comprueba si se trata de un módulo estándar o de clase. Si el módulo contiene menos de tres líneas, damos por hecho que el módulo está vacío y está listo para eliminarse. La información sobre los módulos eliminados se muestra en la ventana **Inmediato**.

1. Inserta un módulo nuevo.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub BorraModulosVacios()
    Dim objVBComp As VBComponent

    Const vbext_ct_StdModule As Long = 1
    Const vbext_ct_ClassModule As Long = 2
    For Each objVBComp In ActiveWorkbook.VBProject.VBComponents
        Select Case objVBComp.Type
            Case vbext_ct_StdModule, vbext_ct_ClassModule
                If objVBComp.CodeModule.CountOfLines < 3 Then
                    Debug.Print "(Eliminado) " & _
                    objVBComp.Name & vbTab & _
```

```
                        "declaraciones: " & objVBComp.CodeModule. _
                        CountOfDeclarationLines & vbTab & _
                        "Total líneas de código: " & _
                        objVBComp.CodeModule.CountOfLines
                        ActiveWorkbook.VBProject.VBComponents. _
                        Remove objVBComp
                    End If
                End Select
            Next
            Set objVBComp = Nothing
        End Sub
```

3. Ejecuta el procedimiento.
   El módulo `PruebaBorrar` creado en un ejercicio anterior se eliminará del proyecto actual de VBA. Comprueba la ventana **Inmediato** para obtener más información sobre el módulo eliminado.

6.6 Copiar un módulo (importar/exportar)

Algunas veces puede ser útil copiar módulos entre proyectos VBA. No hay un único método para realizar esta tarea. Para copiar un módulo podemos hacerlo de la siguiente forma:

1. Para exportar el módulo a un archivo de texto.
   El método `Export` del objeto `VBComponent` guarda el componente en un archivo de texto. Debemos especificar el nombre del archivo al cual queremos exportar el componente. Este nombre de archivo debe ser único o se producirá un error.
2. Para importar el módulo de un archivo de texto.
   El método `Import` del objeto `VBComponent` agrega el componente al proyecto desde un archivo. Debemos especificar la ruta y el nombre del archivo del cual queremos hacer la importación. El libro que recibirá el componente importado debe estar abierto.

Imaginemos que queremos copiar el Modulo1 del libro **Capitulo24 – Editor VBA.xlsm** a otro libro llamado **Capítulo 24b – Editor VBA.xlsm**. El siguiente procedimiento requiere tres argumentos para la operación de copiado: el nombre del libro que contiene el módulo a copiar, el nombre del libro que recibirá el módulo copiado y el nombre del módulo que se va a copiar.

1. Inserta un módulo nuevo en el libro **Capítulo 24 – Editor VBA.xlsm**.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub CopiaModulo(wkbFrom As String, wkbTo As String, _
strFromMod As String)
    Dim wkb As Workbook
    Dim strFile As String
```

```
    Set wkb = Workbooks(wkbFrom)
    strFile = wkb.Path & "\CodigoVB.bas"
    wkb.VBProject.VBComponents(strFromMod).Export strFile
    On Error Resume Next
    Set wkb = Workbooks(wkbTo)
    If Err.Number <> 0 Then
        Workbooks.Open wkbTo
        Set wkb = Workbooks(wkbTo)
    End If
    wkb.VBProject.VBComponents.Import strFile
    wkb.Save
    Set wkb = Nothing
End Sub
```

3. Crea un libro nuevo y llámalo **Capítulo 24b – Editor VBA.xlsm**. Guárdalo en la carpeta C:\Archivos Manual VBA..
   Este archivo se usará en el procedimiento `CopiaModulo`.
4. En la ventana **Inmediato** escribe la siguiente línea de código y presiona **Intro**:

   ```
   CopiaModulo "Capítulo 24 - Editor VBA.xlsm","Capítulo 24b - Editor VBA.xlsm","Módulo1"
   ```

   Cuando ejecutas la declaración anterior, el procedimiento `CopiaModulo` exporta el Módulo1 del libro **Capítulo 24 - Editor VBA.xlsm** a un archivo llamado **CodigoVB.bas**. A continuación, el archivo es importado al libro **Capítulo 24b - Editor VBA.xlsm** y se guarda. Puede que quieras agregar alguna instrucción adicional al procedimiento para eliminar el archivo **CodigoVB.bas** del ordenador. Utiliza el método `Kill` que aprendiste en un capítulo anterior.

5. Activa el proyecto VBA del libro **Capítulo 24b - Editor VBA.xlsm** y fíjate que se ha creado la carpeta Módulos con el Módulo1 que copiaste. El contenido del módulo debería ser el mismo que en el libro original.

6.7 Copiar todos los módulos (importar/exportar)

En ocasiones puede que necesitemos transferir todo el código VBA de u proyecto a otro. El siguiente procedimiento exporta todos los módulos del libro especificado a un archivo de texto externo y luego los importa en otro libro. Si el libro de destino no se encuentra abierto ocurre un error. El procedimiento captura ese error ejecutando el código que abre el libro requerido. Si el archivo de texto ya existe en la misma carpeta, el procedimiento elimina el antiguo antes de que los módulos del proyecto se exporten.

El siguiente ejercicio supone que el libro **Capítulo 24b - Editor VBA.xlsm** se encuentra abierto.

1. Inserta un módulo nuevo en el libro **Capítulo 24 - Editor VBA.xlsm**.
2. En la ventana **Código**, introduce el siguiente procedimiento:

```
Sub CopiaTodosModulos(wkbFrom As String, wkbTo As String)
```

```
    Dim objVBComp As VBComponent
    Dim wkb As Workbook
    Dim strFile As String

    Set wkb = Workbooks(wkbFrom)
    On Error Resume Next
    Workbooks(wkbTo).Activate
    If Err.Number <> 0 Then Workbooks.Open wkbTo
        strFile = wkb.Path & "\CodigoVB.bas"
    If Dir(strFile) <> "" Then Kill strFile
    For Each objVBComp In wkb.VBProject.VBComponents
        If objVBComp.Type <> vbext_ct_Document Then
            objVBComp.Export strFile
          Workbooks(wkbTo).VBProject.VBComponents.Import strFile
        End If
    Next
    Set objVBComp = Nothing
    Set wkb = Nothing
End Sub
```

3. En la ventana **Inmediato**, introduce la siguiente instrucción y presiona **Intro** para ejecutarla:

```
CopiaTodosModulos "Capítulo 24 - Editor VBA.xlsm","Capítulo 24 - Editor VBA.xlsm"
```

Al ejecutar la instrucción anterior, el procedimiento `CopiaTodosModulos` copiará todos los módulos del libro **Capítulo 24 - Editor VBA.xlsm** al libro **Capítulo 24b - Editor VBA.xlsm**. Como en el libro de destino puede existir algún módulo con el mismo nombre, puedes modificar el procedimiento para copiar únicamente los módulos que no tengan conflictos con los nombres. Observa el procedimiento `CopiaTodosModulos2`:

```
Sub CopiaTodosModulos2(wkbFrom As String, wkbTo As String)
    Dim objVBComp As VBComponent
    Dim wkb As Workbook
    Dim strFile As String

    Set wkb = Workbooks(wkbFrom)
    On Error Resume Next
    Workbooks(wkbTo).Activate
    If Err.Number <> 0 Then Workbooks.Open wkbTo
    strFile = wkb.Path & "\vbCode.bas"
    If Dir(strFile) <> "" Then Kill strFile
    For Each objVBComp In wkb.VBProject.VBComponents
```

```
            If objVBComp.Type <> vbext_ct_Document Then
                objVBComp.Export strFile
                With Workbooks(wkbTo)
                If Len(.VBProject.VBComponents( _
                    objVBComp.Name).Name) = 0 Then
                    Workbooks(wkbTo).VBProject. _
                    VBComponents.Import strFile
                End If
                End With
            End If
        Next
        Set objVBComp = Nothing
        Set wkb = Nothing
    End Sub
```

7 Trabajar con procedimientos

Ya sabemos que los módulos contienen procedimientos y que en algunas ocasiones podemos necesitar:

- Enumerar los procedimientos que contiene un módulo (o todos).
- Agregar o eliminar un procedimiento de un módulo.
- Crear un procedimiento de eventos.

La siguiente sección muestra cómo realizar estas tareas.

7.1 Enumerar los procedimientos que contiene un módulo (o todos)

Los módulos contienen líneas de declaración de procedimiento y líneas de código (las que ejecutan las acciones). Es posible obtener el número de líneas que contienen declaraciones de procedimientos con la propiedad `CountOfDeclarationLines` del objeto `CodeModulo`. Usando la propiedad `CountOfLines` obtendremos el número de líneas de código en el módulo.

Cada línea de código pertenece a un solo procedimiento. Utilizamos la propiedad `ProcOfLine` para devolver el nombre del procedimiento en el que se localiza una línea específica. Esta propiedad requiere dos argumentos: el número de línea que queremos comprobar y la constante que especifica el tipo de procedimiento a localizar. Todos los procedimientos `Sub` y `Function` utilizan la constante `vbext_pk_proc`. El siguiente procedimiento muestra en la ventana **Inmediato** una lista de todos los módulos y todos los procedimientos en el proyecto de VBA.

1. Inserta un módulo nuevo en el libro **Capítulo 24 – Editor VBA.xlsm**.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub EnumeraProcedimientos()
    Dim objVBProj As VBProject
```

```
    Dim objVBComp As VBComponent
    Dim objVBCode As CodeModule
    Dim strCurrent As String
    Dim strPrevious As String
    Dim x As Integer

    Set objVBProj = ThisWorkbook.VBProject
    For Each objVBComp In objVBProj.VBComponents
        If InStr(1, "1, 2", objVBComp.Type) Then
            Set objVBCode = objVBComp.CodeModule
            Debug.Print objVBComp.Name
            For x = objVBCode.CountOfDeclarationLines + 1 To _
                objVBCode.CountOfLines
                strCurrent = _
                objVBCode.ProcOfLine(x, vbext_pk_Proc)
                If strCurrent <> strPrevious Then
                    Debug.Print vbTab & objVBCode.ProcOfLine( _
                    x, vbext_pk_Proc)
                    strPrevious = strCurrent
                End If
            Next
        End If
    Next
    Set objVBCode = Nothing
    Set objVBComp = Nothing
    Set objVBProj = Nothing
End Sub
```

3. Ejecuta el procedimiento.
   Cuando finaliza la ejecución, puede verse la lista de procedimientos en la ventana **Inmediato**.

7.2 Agregar un procedimiento

Es bastante fácil escribir el código de un procedimiento en un módulo. Se utiliza el método `InsertLines` del objeto `CodeModule` para insertar una o varias líneas de código en un lugar determinado.

El método `InsertLines` requiere dos argumentos: el número de línea en el que se quiere insertar el código y la cadena que contiene el código que se quiere insertar. El siguiente ejemplo escribe un procedimiento que crea un libro nuevo y le cambia el nombre de la hoja a `Hoja Prueba`. El procedimiento se inserta al final del código del módulo especificado.

1. Inserta un módulo nuevo en el proyecto.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub NuevoProcedimiento(strModName As String)
    Dim objVBCode As CodeModule
    Dim objVBProj As VBProject
    Dim strProc As String

    Set objVBProj = ThisWorkbook.VBProject
    Set objVBCode = objVBProj.VBComponents( _
    strModName).CodeModule
    strProc = "Sub CrearLibro()" & Chr(13)
    strProc = strProc & Chr(9) & "Workbooks.Add" & Chr(13)
    strProc = strProc & Chr(9) & _
    "ActiveSheet.Name = ""Hoja Prueba""" & Chr(13)
    strProc = strProc & "End Sub"
    Debug.Print strProc
    With objVBCode
        .InsertLines .CountOfLines + 1, strProc
    End With
    Set objVBCode = Nothing
    Set objVBProj = Nothing
End Sub
```

3. En la ventana **Inmediato**, introduce la siguiente instrucción para ejecutar el procedimiento anterior y presiona **Intro**:

```
NuevoProcedimiento "Módulo1"
```

Cuando el procedimiento finaliza la ejecución, el Módulo 1 contendrá un nuevo procedimiento llamado `CrearLibro`. En la ventana **Inmediato** se mostrará el código del procedimiento.

7.3 Eliminar un procedimiento

Utilizamos el método `DeleteLines` del objeto `CodeModule` para eliminar una o varias líneas de código. El método `DeleteLines` tiene dos argumentos: el primero es obligatorio y el segundo opcional. Debemos especificar el número de la primera línea que queremos borrar. El número total de líneas es opcional. Antes de eliminar el procedimiento completo es necesario localizar la línea en la que comienza el procedimiento. Esto se hace con la propiedad `ProcStartLine` del objeto `CodeModule`. Esta propiedad requiere dos argumentos: una cadena de texto con el nombre del procedimiento y la clase de procedimiento a borrar. Usamos la constante `vbext_pc_proc` para eliminar un procedimiento `Sub` o `Function`.

El siguiente procedimiento elimina un procedimiento específico de un módulo concreto:

1. Inserta un módulo nuevo en el proyecto del libro **Capítulo 24 – Editor VBA.xlsm**.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub EliminaProcedimiento(strModName As String, strProcName _
As String)
```

```
    Dim objVBProj As VBProject
    Dim objVBCode As CodeModule
    Dim firstLn As Long
    Dim totLn As Long

    Set objVBProj = ThisWorkbook.VBProject
    Set objVBCode = _
    objVBProj.VBComponents(strModName).CodeModule
    With objVBCode
        firstLn = .ProcStartLine(strProcName, vbext_pk_Proc)
        totLn = .ProcCountLines(strProcName, vbext_pk_Proc)
        .DeleteLines firstLn, totLn
    End With
    Set objVBProj = Nothing
    Set objVBCode = Nothing
End Sub
```

3. En la ventana **Inmediato** introduce la siguiente instrucción para ejecutar el procedimiento anterior:

```
EliminaProcedimiento "Módulo1", "CrearLibro"
```

Cuando el procedimiento finaliza la ejecución, el `Módulo1` ya no debería contener el procedimiento `CrearLibro`, generado automáticamente en el ejercicio anterior.

Antes de eliminar el procedimiento de un módulo concreto, es recomendable comprobar si tanto el módulo como el procedimiento existen. Una buena idea es crear dos funciones que devuelvan esta información. Puedes llamar a estas funciones si necesitas comprobar si existe un módulo o un procedimiento.

4. Introduce la siguiente función en la ventana **Código** del `Módulo6`:

```
Function ExisteModulo(strModName As String) As Boolean
    Dim objVBProj As VBProject

    Set objVBProj = ThisWorkbook.VBProject
    On Error Resume Next
    ExisteModulo = Len(objVBProj.VBComponents(strModName).Name)
<> 0
End Function
```

La función devolverá `True` si la longitud del nombre del módulo es un número diferente a 0. En caso contrario devolverá `False`.

5. Introduce la siguiente función en el `Módulo6`:

```
Function ExisteProcedimiento(strModName As String, _
```

```
strProcName As String) As Boolean
    Dim objVBProj As VBProject

    Set objVBProj = ThisWorkbook.VBProject
    On Error Resume Next
    ' Primero busca si el módulo especificado existe
    If ExisteModulo(strModName) = True Then
      ExisteProcedimiento = objVBProj.VBComponents(strModName) _
      .CodeModule.ProcStartLine(strProcName, vbext_pk_Proc) <> 0
    End If
End Function
```

7.4 Crear un procedimiento de eventos

Aunque se podría crear un procedimiento de eventos utilizando el método `InsertLines` del objeto `CodeModule` como se ha hecho anteriormente, hay una forma más fácil. Como los procedimientos de eventos tienen una estructura específica y normalmente requieren un número fijo de argumentos, VBA ofrece un método especial para manejar esta tarea. El método `CreateEventProc` del objeto `CodeModule` crea un procedimiento de eventos con la declaración y los argumentos necesarios. Todo lo que necesitamos hacer es especificar el nombre del evento que queremos añadir y el nombre del objeto fuente del evento. El método `CreateEventProc` devuelve la línea en la que se inicia el cuerpo del procedimiento de evento. A continuación, usamos el método `InsertLines` del objeto `CodeModule` para insertar el código en el cuerpo del procedimiento.

El siguiente procedimiento añade una nueva hoja de trabajo al libro activo y crea el procedimiento de eventos `Worksheet_SelectionChange` en su módulo de código:

1. Inserta un módulo nuevo en el proyecto del libro **Capítulo 24 – Editor VBA.xlsm**.
2. Introduce el siguiente procedimiento:

```
Sub ProcEvento()
    Dim objVBCode As CodeModule
    Dim wks As Worksheet
    Dim firstLine As Long

    ' Agrega una hoja nueva
    Set wks = ActiveWorkbook.Worksheets.Add
    ' Crea una referencia al módulo
    ' de la nueva hoja
    Set objVBCode = wks.Parent.VBProject.VBComponents( _
    wks.Name).CodeModule
    ' Crea un procedimiento de evento y devuelve el número
    ' de línea en la que comienza el cuerpo del evento
    firstLine = objVBCode.CreateEventProc( _
```

```
        "SelectionChange", "Worksheet")
        Debug.Print "Primera línea del procedimiento: " & firstLine
        ' proceed to add code to the body of the event procedure
        objVBCode.InsertLines firstLine + 1, Chr(9) & _
        "Dim myRange As Range"
        objVBCode.InsertLines firstLine + 2, Chr(9) & _
        "On Error Resume Next"
        objVBCode.InsertLines firstLine + 3, Chr(9) & _
        "Set myRange = Intersect(Range(""A1:A10""),Target)"
        objVBCode.InsertLines firstLine + 4, _
        Chr(9) & "If Not myRange Is Nothing Then"
        objVBCode.InsertLines firstLine + 5, _
        Chr(9) & Chr(9) & _
        "MsgBox ""No está permitida la introducción de datos."""
        objVBCode.InsertLines firstLine + 6, _
        Chr(9) & "End If"
        Set objVBCode = Nothing
        Set wks = Nothing
    End Sub
```

3. Ejecuta el procedimiento.
   Al hacerlo se insertará una nueva hoja y en su módulo se creará el siguiente procedimiento de eventos:

```
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
    Dim myRange As Range
    On Error Resume Next
    Set myRange = Intersect(Range("A1:A10"), Target)
    If Not myRange Is Nothing Then
        MsgBox "No está permitida la introducción de datos."
    End If
End Sub
```

4. Para probar el procedimiento, activa la hoja nueva y haz clic en alguna regla del rango **A1:A10**. El procedimiento de evento provocará que aparezca un mensaje.

8 Trabajar con UserForms

Anteriormente en el manual aprendimos a crear y a trabajar con formularios personalizados (`UserForms`). La creación de `UserForms` se hace más fácil utilizando el método manual; sin embargo, a veces puede ser necesario utilizar VBA para crear un formulario rápido sobre la marcha y mostrarlo correctamente en la pantalla del usuario.

Para agregar Un **UserForm** al proyecto activo mediante programación se utiliza el método **Add** de la colección **VBComponents** y la constante **vbext_ct_MSForm** para el tipo de componente a añadir:

```
Dim objVBComp As VBComponent
Set objVBComp = Application.VBE.ActiveVBProject. _
VBComponents.Add(vbext_ct_MSForm)
```

Para cambiar el nombre del **UserForm**, se usa la propiedad **Name** del objeto **VBComponent**. Para modificar otras propiedades del **UserForm**, se usa la colección de propiedades de **VBComponent**. Por ejemplo, para cambiar el nombre y el título del **UserForm**, usamos el siguiente bloque de instrucciones.

```
With objVBComp
    .Name = "MiFormulario"
    .Properties("Caption") = "Mi formulario"
End With
```

Este es el procedimiento completo:

```
Sub CreaFormulario()
    Dim objVBComp As VBComponent

    Set objVBComp = Application.VBE.ActiveVBProject. _
    VBComponents.Add(vbext_ct_MSForm)
    With objVBComp
        .Name = "MiFormulario"
        .Properties("Caption") = "Mi Formulario"
    End With
    Set objVBComp = Nothing
End Sub
```

Para eliminar el UserForm del proyecto, usamos el método **Remove** de la colección **VBComponents**:

```
Set objVBComp = Application.VBE.ActiveVBProject. _
VBComponents("MiFormulario")
Application.VBE.ActiveVBProject.VBComponents.Remove objVBComp
```

8.1 Crear y manejar UserForms

El siguiente ejercicio crea un formulario sencillo como se muestra en la Imagen 8.1 y escribe los procedimientos necesarios para cada control.

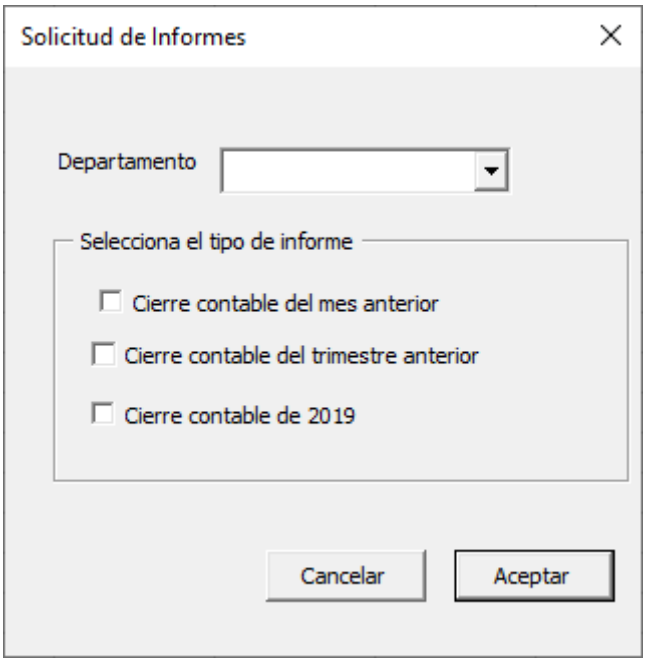


Imagen 8.1 Formulario creado de forma automática mediante programación.

1. Inserta un módulo nuevo en el proyecto VBA.
2. Introduce el siguiente procedimiento:

```
Sub CreaUserForm()
    Dim objVBProj As VBProject
    Dim objVBComp As VBComponent
    Dim objVBFrm As UserForm
    Dim objChkBox As Object
    Dim x As Integer
    Dim sVBA As String

    Set objVBProj = Application.VBE.ActiveVBProject
    Set objVBComp = objVBProj.VBComponents.Add(vbext_ct_MSForm)
    With objVBComp
        ' Lee el nombre y otras propiedades predeterminadas _
          del formulario
       Debug.Print "Nombre predeterminado " & .Name
       Debug.Print "Título: " & .DesignerWindow.Caption
       Debug.Print _
       "El formulario está abierto en la ventana Diseñador: " _
       & .HasOpenDesigner
       Debug.Print "Nombre del formulario " & .Name
       Debug.Print "Ancho predeterminado " & _
       .Properties("Width")
       Debug.Print "Alto predeterminado " & _
```

```
    .Properties("Height")
    ' Establece el nombre, el título y el tamaño del _
      formulario
    .Name = "SolicitudInformes"
    .Properties("Caption") = "Solicitud de Informes"
    .Properties("Width") = 250
    .Properties("Height") = 250
End With
Set objVBFrm = objVBComp.Designer
With objVBFrm
    With .Controls.Add("Forms.Label.1", "lbName")
        .Caption = "Departamento:"
        .AutoSize = True
        .Width = 55
        .Top = 30
        .Left = 20
    End With
    With .Controls.Add("Forms.Combobox.1", "cboDept")
        .Width = 110
        .Top = 30
        .Left = 80
    End With
    ' Crea un marco
    With .Controls.Add("Forms.Frame.1", "frReports")
        .Caption = "Selecciona el tipo de informe"
        .Top = 60
        .Left = 18
        .Height = 96
    End With
    ' Agrega tres casillas de verificación
    Set objChkBox = _
    .frReports.Controls.Add("Forms.CheckBox.1")
    With objChkBox
        .Name = "chk1"
        .Caption = "Cierre contable del mes anterior"
        .WordWrap = False
        .Left = 15
        .Top = 12
        .Height = 20
        .Width = 186
```

```
        End With
        Set objChkBox = _
        .frReports.Controls.Add("Forms.CheckBox.1")
        With objChkBox
            .Name = "chk2"
            .Caption = "Cierre contable del trimestre anterior"
            .WordWrap = False
            .Left = 12
            .Top = 32
            .Height = 20
            .Width = 186
        End With
        Set objChkBox = _
        .frReports.Controls.Add("Forms.CheckBox.1")
        With objChkBox
            .Name = "chk3"
            .Caption = "Cierre contable de " & Year(Now) - 1
            .WordWrap = False
            .Left = 12
            .Top = 54
            .Height = 20
            .Width = 186
        End With
        ' Agrega y posiciona los botones Aceptar y Cancelar
        With .Controls.Add("Forms.CommandButton.1", "cmdOK")
            .Caption = "Aceptar"
            .Default = "True"
            .Height = 20
            .Width = 60
            .Top = objVBFrm.InsideHeight - .Height - 20
            .Left = objVBFrm.InsideWidth - .Width - 10
        End With
        With .Controls.Add("Forms.CommandButton.1", "cmdCancel")
            .Caption = "Cancelar"
            .Height = 20
            .Width = 60
            .Top = objVBFrm.InsideHeight - .Height - 20
            .Left = objVBFrm.InsideWidth - .Width - 80
        End With
    End With
```

```
' Rellena el combobox
With objVBComp.CodeModule
    x = .CountOfLines
    .InsertLines x + 1, "Sub UserForm_Initialize()"
    .InsertLines x + 2, vbTab & "With Me.cboDept"
    .InsertLines x + 3, vbTab & vbTab & ".addItem _
     ""Márketing"""
    .InsertLines x + 4, vbTab & vbTab & ".addItem _
    ""Ventas"""
    .InsertLines x + 5, vbTab & vbTab & ".addItem _
    ""Finanzas"""
    .InsertLines x + 6, vbTab & vbTab & _
    ".addItem ""I + D"""
    .InsertLines x + 7, vbTab & vbTab & _
    ".addItem ""Recursos Humanos"""
    .InsertLines x + 8, vbTab & "End With"
    .InsertLines x + 9, "End Sub"
    ' Escribe el procedimiento que gestiona el botón _
     Cancelar
    Dim firstLine As Long
    With objVBComp.CodeModule
        firstLine = .CreateEventProc("Click", "cmdCancel")
        .InsertLines firstLine + 1, " Unload Me"
    End With
    ' Escribe el procedimiento para el botón Aceptar
    sVBA = "Private Sub cmdOK_Click()" & vbCrLf
    sVBA = sVBA & " Dim ctrl As Control" & vbCrLf
    sVBA = sVBA & " Dim chkflag As Integer" & vbCrLf
    sVBA = sVBA & " Dim strMsg As String" & vbCrLf
    sVBA = sVBA & " If Me.cboDept.Value = """" Then " & _
    vbCrLf
    sVBA = sVBA & _
    " MsgBox ""Selecciona el departamento."" & vbCrLf
    sVBA = sVBA & " Me.cboDept.SetFocus " & vbCrLf
    sVBA = sVBA & " Exit Sub" & vbCrLf
    sVBA = sVBA & " End If" & vbCrLf
    sVBA = sVBA & " For Each ctrl In Me.Controls " & vbCrLf
    sVBA = sVBA & " Select Case ctrl.Name" & vbCrLf
    sVBA = sVBA & " Case ""chk1"", ""chk2"", ""chk3""" _
    & vbCrLf
```

```
            sVBA = sVBA & " If ctrl.Value = True Then" & vbCrLf
            sVBA = sVBA & " strMsg = strMsg & vbCrLf & ctrl _
            .Caption "  & Chr(13) & vbCrLf
            sVBA = sVBA & " chkflag = 1" & vbCrLf
            sVBA = sVBA & " End If" & vbCrLf
            sVBA = sVBA & " End Select" & vbCrLf
            sVBA = sVBA & " Next" & vbCrLf
            sVBA = sVBA & " If chkflag = 1 Then" & vbCrLf
            sVBA = sVBA & " MsgBox ""Ejecutar informes de "" " & _
            vbCrLf
            sVBA = sVBA & " Me.cboDept.Value & "":""""
            sVBA = sVBA & " & Chr(13) & Chr(13) & strMsg" & vbCrLf
            sVBA = sVBA & " Else" & vbCrLf
            sVBA = sVBA & _
            " MsgBox ""Selecciona el tipo de informe."""" & vbCrLf
            sVBA = sVBA & " End If" & vbCrLf
            sVBA = sVBA & "End Sub"
            .AddFromString sVBA
        End With
        Set objVBComp = Nothing
End Sub
```

En el procedimiento anterior, la siguiente instrucción crea un formulario en blanco:

```
Set objVBComp = objVBProj.VBComponents.Add(vbext_ct_MSForm)
```

A continuación, se muestran en la ventana **Inmediato** el nombre predeterminado y otras propiedades del formulario (Título, ancho y alto). Antes de acceder al contenido del `UserForm` necesitamos una referencia al diseñador de objetos de `VBComponents`:

```
Set objVBFrm = objVBComp.Designer
```

Se utilizan varios bloques `With … End With` para agregar los controles al formulario en blanco y situarlos utilizando las propiedades `Top` y `Left`. Las propiedades `InsideHeight` e `InsideWidth` se usan para mover los botones **Aceptar** y **Cancelar** a la parte inferior del `UserForm`. Estas propiedades devuelven el ancho y el alto del espacio disponible dentro del formulario.

El código restante crea varios procedimientos de evento para el `UserForm` y sus controles. El primero de ellos es `UserForm_Initialize()`, que rellenará el cuadro combinado con los nombres de departamentos antes de que el formulario se muestre en la pantalla. A continuación, se crean los procedimientos de evento para los botones de comando (**Aceptar** y **Cancelar**). El procedimiento `cmdCancel_Click()` descarga el formulario y `cmdOK_click()` muestra un cuadro de mensaje con la información sobre los tipos de informes seleccionados en las casillas de verificación. El código para el procedimiento de evento puede ser introducido con varios métodos. Uno de ellos es

usar la instrucción `InsertLines`. La otra es crear una cadena de texto que almacene el código y luego agregarla al código del módulo con el método `AddFromString`. El código introducido por el método `AddFromString` se inserta en la primera línea del módulo. El método `AddFromFile` se puede usar para agregar código almacenado en un archivo de texto.

3. Ejecuta el procedimiento `CreaUserForm`.
   Cuando finaliza la ejecución, el editor de VBA mostrará el formulario de la Imagen 8.1.
4. Selecciona el menú **Ver – Código** o presiona **F7** y revisa los procedimientos creados de forma automática.
5. Haz clic en **Ejecutar – Ejecutar Sub / UserForm** o presiona **F5** para mostrar y trabajar con el formulario.
   Si quieres mostrar tu formulario personalizado en una posición específica en la pantalla, añade el siguiente procedimiento de evento al procedimiento `CreaUserForm` usando algunas de las técnicas mostradas en el capítulo.

```
Private Sub UserForm_Activate()
    With ReportSelector
        .Top = 100
        .Left = 25
    End With
End Sub
```

9 Trabajar con referencias

Mientras estamos escribiendo nuestros procedimientos a menudo necesitamos acceder a objetos almacenados en bibliotecas externas. Por ejemplo, en este capítulo hemos usado objetos definidos en la biblioteca **Microsoft Visual Basic for Applications Extensibility 5.3**. En otros capítulos del manual hemos trabajado con objetos que se encuentran en las bibliotecas **Microsoft Word 16.0** o **Microsoft ActiveX Data Objects 6.1**.

Hay dos formas de “informar” y activar un modelo de objetos (referencia) a Excel: con enlace temprano y enlace tardío.

Se usa el enlace temprano (también se llama vinculación temprana) cuando se expone el modelo de datos en tiempo de diseño. Esto se hace seleccionando el menú **Herramientas – Referencias** en la pantalla del editor de VBA. El cuadro de diálogo **Referencias** enumera los archivos con los que se puede enlazar. “Enlazar” significa poner un modelo de objetos a disposición de Excel. Para manipular una aplicación específica en un proyecto de Excel, debemos seleccionar la casilla de verificación junto al nombre de la biblioteca que queremos utilizar.

La vinculación tardía (o enlace tardío) se realiza cuando se vincula la biblioteca de objetos en tiempo de ejecución. En lugar de utilizar el cuadro de diálogo **Referencias**, se utilizan las funciones `GetObject` o `CreateObject` durante la ejecución del código.

Al añadir una referencia a la biblioteca de objetos a través del cuadro de diálogo **Referencias** (vinculación temprana), tienes la posibilidad de obtener asistencia en la programación de los objetos que deseas incluir en el código, evitando así muchos errores de sintaxis. También puedes ver el modelo de objetos desde el **Examinador de objetos** y tener acceso a las constantes predefinidas de la aplicación. Además, el código se ejecuta más rápido porque las referencias a las bibliotecas externas se comprueban y compilan en tiempo de diseño. Sin embargo, los problemas surgen cuando movemos el código a otros ordenadores que no tienen instaladas estas bibliotecas externas. Los procedimientos que funcionaban perfectamente bien en nuestro equipo, de repente, en otros ordenadores empiezan a mostrar errores en tiempo de compilación que no se pueden capturar usando las técnicas habituales de manejo de errores. Para asegurarte de que los usuarios finales tienen las referencias y las bibliotecas de objetos necesarias, debes escribir un código que compruebe no solo si estas bibliotecas están presentes en los equipos de las personas que vayan a utilizarlos sino también si son de la versión correcta. Esta sección muestra cómo hacerlo:

- Enumerar las referencias a las bibliotecas de objetos externos seleccionadas en el cuadro de diálogo **Referencias**.
- Agregar una referencia a una biblioteca en tiempo de ejecución.
- Eliminar las referencias a las bibliotecas que faltan.
- Comprobar si hay referencias rotas.

9.1 Crear una lista de referencias

El objeto `Reference` de la colección `References` representa una referencia a una biblioteca o a un proyecto VBA. Podemos usar varias propiedades del objeto `Reference` para:

- Averiguar si la referencia se ha activado (propiedad `BuiltIn`).
- Determinar si la referencia está rota (propiedad `IsBroken`).
- Averiguar el número de versión de la referencia (propiedades `Major` y `Minor`).
- Obtener la descripción de la referencia tal y como aparece en el **Examinador de objetos** (propiedad `Description`).
- Devolver la ruta completa al libro o al archivo de referencias DLL, OCX, TLD o OLF (propiedad `FullPath`).
- Determinar el identificador único global para la referencia (propiedad `Guid`).
- Determinar el tipo de referencia (propiedad `Type`).

El siguiente procedimiento muestra en la ventana **Inmediato** los nombres de todos los proyectos VBA y rutas completas de las referencias seleccionadas en cada uno, así como los nombres de los componentes.

1. Inserta un módulo nuevo en el proyecto VBA del libro **Capítulo 24 – Editor VBA.xlsm**.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub ReferenciasComplementos()
    Dim objVBPrj As VBIDE.VBProject
    Dim objVBCom As VBIDE.VBComponent
    Dim vbRef As VBIDE.Reference
```

```
    ' Enumera los proyectos VBA, las referencias
    ' y los componentes
    For Each objVBPrj In Application.VBE.VBProjects
        Debug.Print objVBPrj.Name
        Debug.Print vbTab & "Referencias"
        For Each vbRef In objVBPrj.References
            With vbRef
                Debug.Print vbTab & vbTab & .Name & "-->" & _
                .FullPath
            End With
        Next
        Debug.Print vbTab & "Componentes"
        For Each objVBCom In objVBPrj.VBComponents
            Debug.Print vbTab & vbTab & objVBCom.Name
        Next
    Next
    Set vbRef = Nothing
    Set objVBCom = Nothing
    Set objVBPrj = Nothing
End Sub
```

3. Ejecuta el procedimiento.
   Cuando finalice la ejecución, la ventana **Inmediato** mostrará información sobre todos los proyectos VBA que se encuentren abiertos en este momento.

9.2 Agregar una referencia

El método `AddFromFile` de la colección `References` se usa para agregar una referencia a un proyecto desde un archivo. Debemos especificar el nombre del archivo que contiene la biblioteca, incluyendo toda la ruta. El siguiente procedimiento agrega una referencia a la biblioteca **Microsoft Scripting Runtime**, almacenada en el archivo dcrrun.dll.

1. Inserta un módulo nuevo.
2. En la ventana **Código** introduce el siguiente procedimiento:

```
Sub AgregaRef()
    Dim objVBProj As VBProject
    Set objVBProj = ThisWorkbook.VBProject
    On Error GoTo GestionErrores
    objVBProj.References.AddFromFile _
    "C:\Windows\System32\scrrun.dll"
    MsgBox "Se ha establecido la referencia a la biblioteca " _
    & "Microsoft Scripting Runtime."
```

```
    Application.SendKeys "%tr"
Salir:
    Set objVBProj = Nothing
    Exit Sub
GestionErrores:
    MsgBox "Ya existe la referencia a la biblioteca " & _
    "Microsoft Scripting Runtime."
    GoTo Salir
End Sub
```

3. Ejecuta el procedimiento **AgregaRef**. Cuando aparezca el cuadro de mensaje haz clic en **Aceptar**. Si la referencia a la biblioteca fue activada durante el procedimiento, el cuadro de diálogo **Referencias** mostrará la biblioteca activada.
4. Cierra el cuadro **Referencias** si lo has abierto.

Cada tipo de biblioteca tiene un identificador único (GUID) que se almacena en el registro de Windows. Si conocemos el GUID de la referencia, podemos agregarla usando el método **AddFromGuid**. Este método requiere tres argumentos: una cadena de texto que representa el GUID de la referencia, la versión mayor y la versión menor. (ver entrada sobre la nomenclatura de versiones de software en Wikipedia, https://es.wikipedia.org/wiki/Versionado_de_software). El método **AddFromGuid** busca en el registro la referencia que deseamos añadir.

El siguiente procedimiento muestra en la ventana **Inmediato** los nombres, GUIDs y números de versión instalados en nuestro equipo de los proyectos VBA activos. El procedimiento también agrega una referencia a la biblioteca **Microsoft DAO 3.6**, si ésta no se ha añadido todavía.

1. En la ventana **Código** donde introdujiste el procedimiento anterior, introduce este otro:

```
Sub AgregaRefGUID()
    Dim objVBProj As VBProject
    Dim i As Integer
    Dim strName As String
    Dim strGuid As String
    Dim strMajor As Long
    Dim strMinor As Long

    Set objVBProj = ActiveWorkbook.VBProject
    ' Busca las bibliotecas instaladas previamente
    For i = 1 To objVBProj.References.Count
        strName = objVBProj.References(i).Name
        strGuid = objVBProj.References(i).GUID
        strMajor = objVBProj.References(i).Major
```

```
            strMinor = objVBProj.References(i).Minor
            Debug.Print strName & " - " & strGuid & _
            ", " & strMajor & ", " & strMinor
        Next i
        ' Agrega la referencia a Microsoft DAO 3.6
        On Error Resume Next
        ThisWorkbook.VBProject.References.AddFromGuid _
        "{00025E01-0000-0000-C000-000000000046}", 5, 0
    End Sub
```

2. Ejecuta el procedimiento.
   Al ejecutarse, crea la siguiente lista de referencias en la ventana Inmediato (puede variar con tus resultados):

```
VBA - {000204EF-0000-0000-C000-000000000046}, 4, 2
Excel - {00020813-0000-0000-C000-000000000046}, 1, 9
stdole - {00020430-0000-0000-C000-000000000046}, 2, 0
Office - {2DF8D04C-5BFA-101B-BDE5-00AA0044DE52}, 2, 8
VBIDE - {0002E157-0000-0000-C000-000000000046}, 5, 3
MSForms - {0D452EE1-E08F-101A-852E-02608C4D0BB4}, 2, 0
Scripting - {420B2830-E718-11CF-893D-00A0C9054228}, 1, 0
```

Observa que en el listado no aparece la biblioteca Microsoft DAO 3.6 Object Library. Esto es debido a que se añadió después de haber ejecutado el bloque `For … Next`.

9.3 Eliminar una referencia

Para eliminar una referencia que ya no nos sea necesaria, usamos el método `Remove` de la colección `References`. El siguiente procedimiento elimina la referencia a Microsoft DAO 3.6 Object Library creada en el procedimiento anterior.

1. En la ventana **Código** donde introdujiste el procedimiento anterior, ahora escribe el siguiente:

```
Sub EliminaRef()
    Dim objVBProj As VBProject
    Dim objRef As Reference
    Dim sRefFile As String

    Set objVBProj = ActiveWorkbook.VBProject
    ' Recorre todas las referencias y elimina
    ' La referencia a la biblioteca DAO
    For Each objRef In objVBProj.References
        If InStr(1, objRef.Description, "DAO 3.6") > 0 Then
            objVBProj.References.Remove objRef
```

```
            Exit For
        End If
    Next objRef
End Sub
```

2. Ejecuta el procedimiento `EliminaRef`. Cuando finalice la ejecución, abre el cuadro de diálogo Referencias para comprobar que la referencia a **Microsoft DAO 3.6 Object Library** ya no se encuentra activa. Asegúrate de que se encuentra activo el libro **Capítulo 24 – Editor VBA.xlsm** antes de ejecutar el procedimiento.

Además de eliminar referencias a bibliotecas de objetos externas, podemos eliminar cualquier referencia en otros proyectos VBA. Esto se hace comprobando la propiedad `BuiltIn` del objeto `Reference` y eliminando la referencia cuando dicha propiedad es `False`.

```
For Each objRef in objVBProjReferences
    If Not objRef.BuiltIn Then _
    objVBProj.References.Remove objRef
Next objRef
```

9.4 Comprobar referencias rotas

Si la versión de la biblioteca solicitada no se encuentra instalada en nuestro equipo o no es la correcta, la biblioteca se marcará como "ausente" en el cuadro de diálogo **Referencias**. Podemos utilizar la propiedad `IsBroken` para encontrar estas referencias incorrectas.

La propiedad `IsBroken` devuelve el valor `True` si el objeto `Reference` ya no contiene una referencia válida en el registro. Si la referencia es válida, devuelve `False`. El código para comprobar si hay referencias rotas debe incluirse o llamarse desde el procedimiento de eventos `Workbook_Open` antes de intentar añadir nuevas referencias mediante código.

El siguiente procedimiento comprueba si existen referencias rotas:

1. En el módulo `ThisWorkbook` del libro **Capítulo 24 – Editor VBA.xlsm**, introduce el siguiente procedimiento:

```
Private Sub Workbook_Open()
    Dim objVBProj As VBProject
    Dim objRef As Reference
    Dim refBroken As Boolean

    Set objVBProj = ThisWorkbook.VBProject
    ' Recorre las referencias activas
    ' en el cuadro de diálogo Referencias
    For Each objRef In objVBProj.References
        ' Si la referencia está rota,
        ' se obtiene su nombre y GUID
        If objRef.IsBroken Then
```

```
                Debug.Print objRef.Name
                Debug.Print objRef.GUID
                refBroken = True
            End If
        Next
        If refBroken = False Then
            Debug.Print "Todas las referencias son válidas."
        End If
    End Sub
```

2. Guarda y cierra el libro **Capítulo 24 – Editor VBA.xlsm**, pero sin salir de Excel.
3. Abre de nuevo el archivo.
   Cuando se abre el libro, Excel ejecuta el código del procedimiento `Workbook_Open`.
4. Activa el editor de VBA y presiona **Ctrl + G** para mostrar la ventana **Inmediato**. Si se encuentran referencias rotas en el proyecto activo, verás el nombre de la referencia y su GUID; en caso contrario se mostrará un mensaje indicando que todas las referencias son válidas.

Si deseamos comprobar si una referencia específica es válida, inserta un módulo nuevo en el proyecto VBA y escribe una función como esta:

```
Function EstaRota(strRef As String) As Boolean
    Dim objVBProj As VBProject
    Dim objRef As Reference

    Set objVBProj = ThisWorkbook.VBProject
    For Each objRef In objVBProj.References
        If strRef = objRef.Name And objRef.IsBroken Then
            IsBrokenRef = True
            Exit Function
        End If
    Next
    EstaRota = False
End Function
```

Para probar la función, introduce las siguientes instrucciones en la ventana **Inmediato**:

```
ref = EstaRota("OLE Automation")
?ref
```

Si el resultado es Verdadero, la referencia no es válida; si es Falso, es válida.

10 Trabajar con las ventanas

Como ya sabemos, el editor de VBA contiene numerosas ventanas (ventana principal, **Explorador de proyectos**, ventana **Propiedades**, ventanas **Inmediato** e **Inspecciones**, ventana

Código…). Cada una de ellas está representada por el objeto `Window`. Cada objeto `Window` es miembro de la colección `VBIDE.Windows`. Usamos la propiedad `Type` del objeto `Window` para determinar el tipo de ventana. Los tipos de ventana se muestran en la siguiente tabla:

Ventana	Constante	Valor
Código	`vbext_wt_CodeWindow`	0
Diseño	`vbext_wt_Designer`	1
Explorador de objetos	`vbext_wt_Browser`	2
Inspecciones	`vbext_wt_Watch`	3
Locales	`vbext_wt_Locals`	4
Inmediato	`vbext_wt_Immediate`	5
Explorador de proyectos	`vbext_wt_ProjectWindow`	6
Propiedades	`vbext_wt_PropertyWindow`	7
Buscar	`vbext_wt_Find`	8
Buscar y reemplazar	`vbext_wt_FindReplace`	9
Cuadro de herramientas	`vbext_wt_Toolbox`	10
Principal	`vbext_wt_MainWindow`	12

El siguiente procedimiento recorre todas las ventanas del editor de VBA, cierra y la ventana **Inmediato** y muestra un cuadro de diálogo con los nombres de las ventanas abiertas:

1. Inserta un módulo nuevo en el libro **Capítulo 24 – Editor VBA.xlsm**.
2. En la ventana Código introduce el siguiente procedimiento:

```
Sub CierraInmediato()
    Dim objWin As VBIDE.Window
    Dim strOpenWindows As String

    strOpenWindows = _
    "Se encuentran abiertas las siguientes ventanas:" & _
    vbCrLf & vbCrLf
    For Each objWin In Application.VBE.Windows
        Select Case objWin.Type
            Case vbext_wt_Immediate
                MsgBox objWin.Caption & " fue cerrada."
```

```
                objWin.Close
            Case Else
                strOpenWindows = strOpenWindows & _
                objWin.Caption & vbCrLf
        End Select
    Next
    MsgBox strOpenWindows
    Set objWin = Nothing
End Sub
```

3. Ejecuta el procedimiento.

11 Trabajar con los menús y barras de herramientas del editor

En el Capítulo 19 aprendiste a escribir código VBA para crear o modificar los menús contextuales. Usando el mismo objeto `CommandBar` con el que ya te has familiarizado puedes personalizar los menús y las barras de herramientas en el editor de VBA. Para trabajar con la colección `CommandBars`, tienes que asegurarte de que la referencia a la biblioteca de **Microsoft Office 16.0** está activa en el cuadro de diálogo **Referencias**. Si no se encuentra activa en este momento, Excel mostrará un error de "tipo no definido" cuando el código intenta acceder a la colección `CommandBars`.

11.1 Generar una lista de barras de herramientas y controles del editor

El siguiente procedimiento enumera todas las barras de herramientas y menús contextuales que se pueden encontrar en el editor de VBA. Cada barra de herramientas o menú contextual es un objeto `CommandBar`. Cada `CommandBar` tiene unos controles asignados a ella. El siguiente procedimiento enumera todos los controles para cada `CommandBar`, incluyendo los IDs de los controles.

1. Inserta un módulo nuevo en el proyecto VBA.
2. Introduce el siguiente procedimiento:

```
Sub EnumeraControles()
    Dim objCmdBar As CommandBar
    Dim strCmdType As String
    Dim c As Variant

    Workbooks.Add
    Range("A1").Select
    With ActiveCell
        .Offset(0, 0) = "Nombre"
        .Offset(0, 1) = "Título del control"
        .Offset(0, 2) = "ID"
    End With
    For Each objCmdBar In Application.VBE.CommandBars
```

```
            Select Case objCmdBar.Type
                Case 0
                    strCmdType = "Barra de herramientas"
                Case 1
                    strCmdType = "Barra de menú"
                Case 2
                    strCmdType = "Menú contextual"
                End Select
            ActiveCell.Offset(1, 0) = objCmdBar.Name & _
            " (" & strCmdType & ")"
            For Each c In objCmdBar.Controls
                ActiveCell.Offset(1, 0).Select
                With ActiveCell
                    .Offset(0, 1) = c.Caption
                    .Offset(0, 2) = c.ID
                End With
            Next
        Next
        Columns("A:C").AutoFit
        Set objCmdBar = Nothing
    End Sub
```

3. Ejecuta el procedimiento.
   Se creará un libro nuevo en el que se escribe la información sobre las barras de herramientas, menús y controles que se encuentran en el editor de VBA (ver Imagen 11.1).

	A	B	C
1	Nombre	Título del control	ID
2	Barra de menús (Barra de menú)	&Archivo	30002
3		&Edición	30003
4		&Ver	30004
5		&Insertar	30005
6		&Formato	30006
7		&Depuración	30165
8		Ejecu&tar	30012
9		&Herramientas	30007
10		&Complementos	30038
11		Ve&ntana	30009
12		Ay&uda	30010
13		Ribbon Commander	1
14	Estándar (Barra de herramientas)	Microsoft Excel	106
15		Insertar objeto	32806
16		&Guardar Libro4	3
17		Cor&tar	21
18		&Copiar	19

Imagen 11.1 Podemos enumerar los comandos de los menús y barras de herramientas desde un procedimiento.

11.2 Agregar un botón a un menú del editor

El siguiente procedimiento agrega un nuevo **Botón de comando** al final de las opciones del menú **Herramientas**:

1. Introduce el siguiente procedimiento en el mismo módulo donde introdujiste el anterior:

```
Sub AgregaBoton()
    Dim objCmdBar As CommandBar
    Dim objCmdBtn As CommandBarButton

    ' Obtiene la referencia al menú Herramientas del editor
    Set objCmdBar = Application.VBE.CommandBars.FindControl _
    (ID:=30007).CommandBar
    ' Agrega el botón al menú Herramientas
    Set objCmdBtn = objCmdBar.Controls.Add(msoControlButton)
    ' Establece las propiedades del nuevo botón
    With objCmdBtn
        .Caption = "Mostrar menús y barras de herramientas"
        .OnAction = "EnumeraControles"
    End With
End Sub
```

   No ejecutes el procedimiento. Todavía no está finalizado. Para ejecutar un procedimiento asignado a cualquier elemento del menú del editor, es necesario crear el evento `Click` del botón de comando. Utiliza el objeto `CommandBarEvents` para activar el evento `Click` cuando se hace clic en un control del menú. Esto se hace en un módulo de clase.

2. En el menú del editor selecciona **Insertar – Módulo de clase**.
3. En la ventana **Propiedades**, cambia el nombre de `Clase1` por `clsEventosMenu`.
4. En la ventana **Código** del módulo de clase introduce el siguiente código:

```
Public WithEvents cmdBtnEvents As CommandBarButton
Private Sub cmdBtnEvents_Click( _
ByVal Ctrl As Office.CommandBarButton, _
CancelDefault As Boolean)
    On Error Resume Next
    MsgBox "dentro del módulo de clase"
    ' Ejecuta el procedimiento especificado en la propiedad
    ' onAction
    Application.Run Ctrl.OnAction
    ' Especifica que ya hemos gestionado el evento
    CancelDefault = True
```

```
End Sub
```

Observa que la primera declaración del módulo de clase utiliza la palabra **WithEvents** para declarar un objeto llamado **cmdBtnEvents** del tipo **CommandBarButton** cuyos eventos queremos manejar. A continuación, indicamos que este objeto recibirá el evento **Click** cuando se seleccione el elemento de menú. La primera línea del procedimiento **cmdBtnEvents_Click** impedirá que aparezca un mensaje de error en caso de que el procedimiento especificado en la propiedad **onAction** no exista. La siguiente línea ejecutará el procedimiento especificado en la propiedad **onAction** del control. Como la propiedad **onAction** no hace que se ejecute el código del procedimiento especificado, debes llamarlo con el método Run del objeto **Application**.

Ahora que le has dicho a VBA que te gustaría que se encargara del evento **Click** para el elemento de menú, necesitas conectar el módulo de clase con el módulo estándar que contiene el código del procedimiento que creaste en el paso 1.

5. Introduce la siguiente declaración al principio del módulo que contiene el procedimiento **AgregaBoton**:

```
Dim myClickEvent As clsEventosMenu
```

En la instrucción anterior, **myClickEvent** es una variable a nivel de módulo definida por la clase **clsEventosMenu**. Esta variable servirá de enlace entre el elemento de menú y el módulo de la clase **clsEventosMenu**.

El último paso requiere que añadas código al procedimiento **AgregaBoton** para que VBA sepa que necesita manejar el evento **Click** para el elemento del menú.

6. Introduce el siguiente código al final del procedimiento **AgregaBoton**:

```
    ' crea una instancia de la clase clsEventosMenu
    Set myClickEvent = New clsEventosMenu
    ' Conecta la instancia de la clase con el nuevo botón
    Set myClickEvent.cmdBtnEvents = objCmdBtn
    Set objCmdBtn = Nothing
    Set objCmdBar = Nothing
```

El procedimiento **AgregaBoton** debería verse como el siguiente:

```
Sub AgregaBoton()
    Dim objCmdBar As CommandBar
    Dim objCmdBtn As CommandBarButton

    ' Obtiene la referencia al menú Herramientas del editor
    Set objCmdBar = Application.VBE.CommandBars.FindControl _
    (ID:=30007).CommandBar
    ' Agrega el botón al menú Herramientas
    Set objCmdBtn = objCmdBar.Controls.Add(msoControlButton)
```

```
    ' Establece las propiedades del nuevo botón
    With objCmdBtn
        .Caption = "Mostrar menús y barras de herramientas"
        .OnAction = "EnumeraControles"
    End With
    ' crea una instancia de la clase clsEventosMenu
    Set myClickEvent = New clsEventosMenu
    ' Conecta la instancia de la clase con el nuevo botón
    Set myClickEvent.cmdBtnEvents = objCmdBtn
    Set objCmdBtn = Nothing
    Set objCmdBar = Nothing
End Sub
```

7. Ejecuta el procedimiento `AgregaBoton`.
   El procedimiento crea un nuevo botón en la parte inferior del menú **Herramientas** y lo conecta con el evento localizado en el módulo de clase.
8. Haz clic en el nuevo botón del menú **Herramientas**.
   VBA activa el evento `Click` del elemento seleccionado y ejecuta el código de procedimiento especificado en la propiedad `onAction`. Al activar la ventana de Excel, deberías ver un libro nuevo con un listado completo de las barras de herramientas y sus controles.

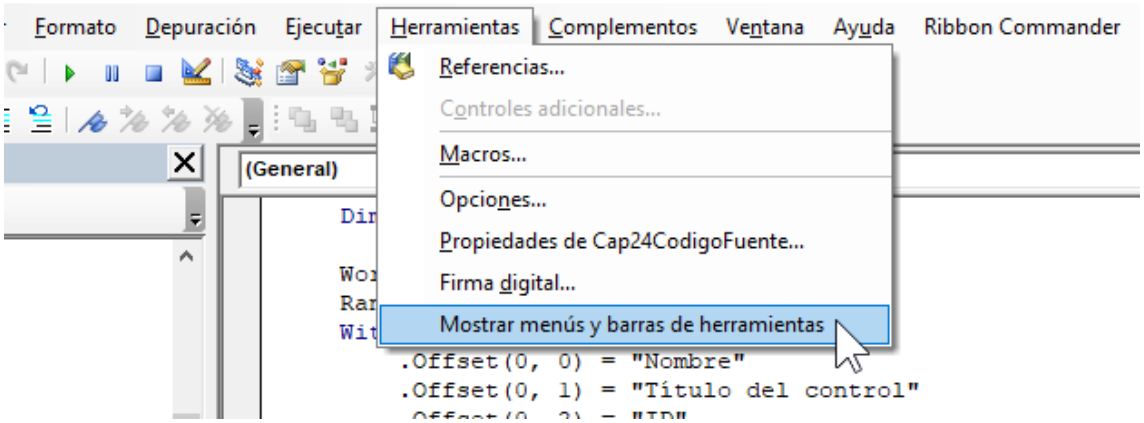


Imagen 11.2 Se ha agregado una nueva opción al menú Herramientas mediante un procedimiento.

11.3 Eliminar un botón de un menú o barra de herramientas del editor

El siguiente procedimiento elimina el botón personalizado que agregamos en la sección anterior:

1. En el mismo módulo donde introdujiste `AgregaBoton`, inserta el siguiente código:

```
Sub EliminaBoton()
    Dim objCmdBar As CommandBar
    Dim objCmdBarCtrl As CommandBarControl

    ' Obtiene la referencia del menú Herramientas
```

```
    Set objCmdBar = Application.VBE.CommandBars("Tools")
    ' Recorre todos los controles de las barras de herramientas
    ' y elimina el control que coincide con el título
    For Each objCmdBarCtrl In objCmdBar.Controls
        If objCmdBarCtrl.Caption = _
        "Mostrar menús y barras de herramientas" Then
            objCmdBarCtrl.Delete
        End If
    Next
    Set objCmdBarCtrl = Nothing
    Set objCmdBar = Nothing
End Sub
```

2. Ejecuta el procedimiento.
   Al ejecutarlo, el botón que agregaste en la sección anterior desaparecerá.

12 Resumen

En este capítulo has utilizado numerosos objetos, propiedades y métodos de la biblioteca **Microsoft Visual Basic for Applications Extesibility 5.3** para controlar el editor de VBA.

Si has llegado hasta aquí, has puesto en práctica todos los ejemplos y ejercicios del manual y has sentido curiosidad hasta el punto de buscar información por tu cuenta, he de darte la enhorabuena.

En el manual he tratado de resumir todos y cada uno de los aspectos fundamentales de la programación en VBA: metodología del lenguaje, ubicación de cada elemento dentro del modelo de objetos, ejemplos de uso de propiedades y métodos y ejercicios que puedes modificar para adaptarlos a tus necesidades en cada momento.

En caso de que te surja alguna duda con el contenido del manual, recuerda que puedes plantearla en el grupo “De 0 a 100 con macros y VBA que he habilitado en el foro de Ayuda Excel. Mi objetivo es que aprendas a crear tus propias macros para automatizar tus tareas.

Gracias.