USERS

INCLUYE VERSIÓN DIGITAL GRATIS

MACROS EN EXCEL 2013

PROGRAMACIÓN DE APLICACIONES CON VBA

EDITOR DE VISUAL BASIC: TRABAJO Y CONFIGURACIÓN

CREACIÓN Y MODIFICACIÓN DE MÓDULOS DE CÓDIGO

DOMINIO DEL MODELO DE OBJETOS DE EXCEL

PROPIEDADES, MÉTODOS Y EVENTOS DE OBJETOS

DOMINIO DE LAS ESTRUCTURAS DE CONTROL

CREACIÓN DE FUNCIONES PERSONALIZADAS

por VIVIANA ZANINI

AUTOMATICE SUS PLANILLAS Y OPTIMICE EL TRABAJO

RU RedUSERS

MACROS EN EXCEL 2013

DESARROLLO DE APLICACIONES CON VISUAL BASIC

por Viviana Zanini

RedUSERS

TÍTULO: Macros en Excel 2013
AUTOR: Viviana Zanini
COLECCIÓN: Manuales USERS
FORMATO: 24 x 17 cm
PÁGINAS: 320

ISBN 978-987-1857-99-9

Zanini, Viviana
Macros en Excel 2013. - 1a ed. - Buenos Aires : Fox Andina, 2013.
320 p. ; 24x17 cm. - (Manual users; 248)
ISBN 978-987-1857-99-9
1. Informática. I. Título
CDD 005.3

Viviana Zanini

Es Analista de Sistemas de Computación y profesora de Informática. Ha realizado diferentes cursos de especialización en el área de programación.

Se desempeña como profesora en institutos incorporados a la enseñanza oficial, en los niveles primario de adultos, secundario y terciario. Ha confeccionado distintas guías de estudio para las materias que imparte.

Ha realizado diferentes proyectos de forma independiente y también ha desarrollado su experiencia laboral en empresas.

Ha colaborado como autora en la colección de fascículos *Curso visual y práctico de Excel*, de esta misma editorial.

Agradecimientos

Quiero agradecer, en primer lugar, a mis padres, por haberme enseñado lo que hoy soy. A mi hermana Sandra, por apoyarme en la vida y en mis proyectos. A mis dos sobrinos, Víctor y Christian, a mi cuñado Hugo, a Haydeé, y a mis amigos que me han apoyado y alentado en esta aventura.

También quiero agradecer especialmente a mi amigo Luis por su invaluable ayuda.

Agradezco a María, mi editora, por darme esta oportunidad y por guiarme a través de los distintos capítulos del libro.

Prólogo

A lo largo de mi vida, he sentido una especial atracción hacia las ciencias exactas y las tecnologías. Aún recuerdo cuando tuve mi primera computadora; desde entonces, algo cambió en mí para siempre. De a poco, fui aprendiendo su funcionamiento e inicié con placer el derrotero de la informática hasta arribar a las herramientas computacionales de hoy en día. De esta manera, fui testigo de cómo estos recursos maravillosos pueden ayudar profundamente tanto a estudiantes como a profesionales en sus diversas áreas.

Por estas razones, cuando me propusieron la idea de realizar esta obra relacionada con la programación de planillas de cálculo, de inmediato me sentí muy atraída con la idea. Al escribir este libro, fue una importante fuente de inspiración el método con el que imparto mis clases, el esfuerzo por transmitir mi apasionamiento a mis alumnos y la forma en que ellos, a su vez, van recibiendo un legado tecnológico que les servirá para la vida misma.

Esta obra contiene todo lo necesario para introducirnos en el mundo de Visual Basic para Aplicaciones (VBA), así como también los conceptos básicos que servirán como punto de partida para quienes quieran comenzar a programar sus propias macros.

Es mi deseo que disfruten del resultado de este libro, así como yo he disfrutado del proceso de elaboración y de escritura. Solo entonces podré decir que he hecho un buen trabajo.

Para concluir, quiero dejarles una frase de Ada Byron:

"La máquina analítica no tiene la pretensión de crear nada. Puede realizar cualquier cosa siempre que conozcamos cómo llevarla a cabo. Puede seguir un análisis; pero es incapaz de descubrir relaciones analíticas o verdades. Su potencialidad es la de ayudarnos a hacer posible aquello sobre lo que tenemos un conocimiento previo".

Viviana Zanini

El libro de un vistazo

Este libro está destinado a todos aquellos que deseen aprender a utilizar Visual Basic para Aplicaciones y así poder automatizar las tareas que realizamos con Microsoft Excel. Además del contenido teórico, incluimos diferentes ejemplos prácticos que se pueden reproducir siguiendo las explicaciones paso a paso.

*01 INTRODUCCIÓN A LA AUTOMATIZACIÓN

En la programación de Microsoft Office, todo gira alrededor de los diferentes objetos y el lenguaje Visual Basic para Aplicaciones. Por esta razón, en este capítulo inicial, conoceremos en detalle los aspectos básicos que nos permitirán comprender el lenguaje VBA para Excel, y explicaremos el modelo de objetos de Excel y su relación jerárquica. Además, vamos a preparar el entorno de trabajo para poder utilizar el lenguaje de programación en los capítulos que desarrollaremos a continuación.

*02 EL EDITOR DE VISUAL BASIC PARA EXCEL

El Editor de Visual Basic para Aplicaciones es un programa cuya ventana principal nos permite escribir, analizar y modificar el código de programación de una macro. Antes de comenzar con el estudio de las instrucciones VBA, describiremos el entorno de programación, y conoceremos las ventanas, las barras de herramientas principales y su funcionamiento en general.

*03 ESCRIBIR SENTENCIAS CON VBA

Como todo lenguaje de programación, VBA posee reglas que debemos respetar para escribir las sentencias. En este capítulo, aprenderemos a escribir los procedimientos para introducir datos y visualizar resultados. También explicaremos cómo podemos exportar o importar un módulo.

*04 LOS DATOS EN VBA

En un sentido amplio, un tipo de dato define un conjunto de valores y las operaciones sobre estos valores. En este capítulo, describiremos los datos y los tipos de datos utilizados por el lenguaje VBA. Veremos el uso de variables y el ámbito de estas. También examinaremos los diferentes operadores para trabajar con ellos.

*05 FUNCIONES

Las funciones son uno de los elementos básicos de la programación. VBA incorpora funciones predefinidas, que no necesitan ser declaradas y codificadas. En este

capítulo, trataremos algunas de las funciones predefinidas básicas: matemáticas, de cadena, de comprobación, entre otras.

*06 ESTRUCTURAS DE PROGRAMACIÓN

En el desarrollo de algoritmos muchas veces se requiere que se realicen cálculos reiterativos, selección de resultados o también verificaciones. En este capítulo, explicaremos cómo representar decisiones en algoritmos cuando existen acciones con dos o más alternativas. También conoceremos las estructuras de control repetitivas disponibles en VBA.

*07 PRINCIPALES OBJETOS DE EXCEL

La programación orientada a objetos se basa en la idea de la existencia de un mundo lleno de objetos, de modo que la resolución del problema se realiza en términos de objetos. En este capítulo, explicaremos y trabajaremos con los principales objetos de Excel, conoceremos sus propiedades y métodos.

*08 FORMULARIOS

Los formularios son cuadros de diálogo que se programan por medio de controles, y que nos permiten crear una interfaz simple y amigable para introducir, modificar o visualizar datos. En este capítulo, explicaremos cómo crear y programar formularios.

SERVICIOS AL LECTOR

En esta ultima sección presentamos un índice temático que nos permitirá encontrar de manera fácil y rápida los contenidos principales de este libro.

INFORMACIÓN COMPLEMENTARIA

A lo largo de este manual, podrá encontrar una serie de recuadros que le brindarán información complementaria: curiosidades, trucos, ideas y consejos sobre los temas tratados. Para que pueda distinguirlos en forma más sencilla, cada recuadro está identificado con diferentes iconos:


CURIOSIDADES E IDEAS

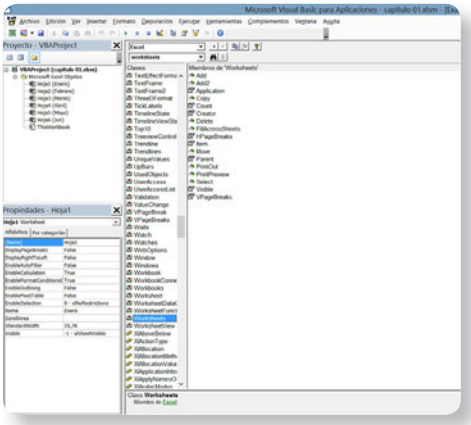

ATENCIÓN


DATOS ÚTILES Y NOVEDADES


SITIOS WEB

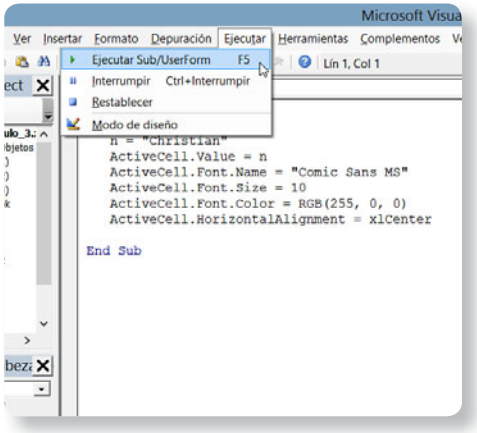
Contenido

*01 Introducción a la automatización



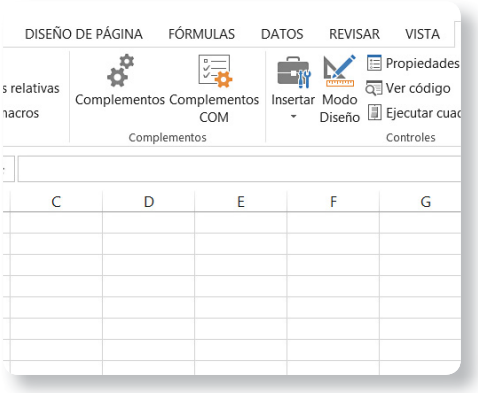
*02 El Editor de Visual Basic para Excel

*03 Escribir sentencias con VBA



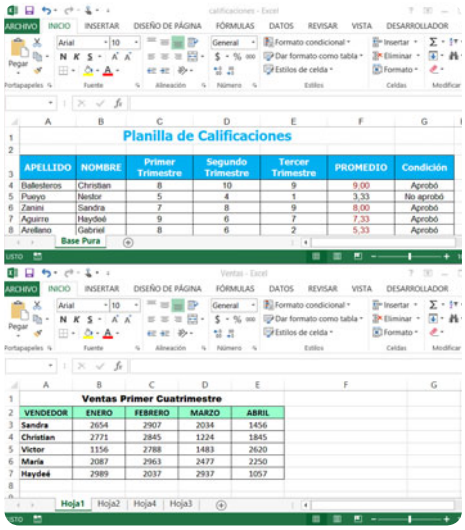
*04
Los datos en VBA

*05
Funciones



*06 Estructuras de programación

*07
Principales objetos de Excel



*08
Formularios

*
Servicios al lector

Introducción

Visual Basic para Aplicaciones es un lenguaje de programación común a todas las aplicaciones del paquete Microsoft Office, que permite escribir un conjunto de instrucciones (macros) para programar los distintos objetos de Excel, como, por ejemplo, una hoja de cálculo, un conjunto de celdas o gráficos. De esta manera, es posible automatizar las tareas que realizamos en forma repetitiva en Excel, y hasta crear aplicaciones.

Comenzaremos por el estudio del lenguaje Visual Basic para Aplicaciones. Daremos una visión general de la programación orientada a objetos, y explicaremos el modelo de objetos de Microsoft Excel y su relación jerárquica. Continuaremos con la descripción de cada una de las herramientas que forman el entorno de programación: el Editor de VBA. Estudiaremos las reglas que debemos seguir para escribir procedimientos para introducir datos y visualizar resultados. Identificaremos la importancia de las variables y veremos cómo aplicarlas de acuerdo con su valor para la resolución de operaciones.

Una vez que hayamos adquirido estos conocimientos, aprenderemos a utilizar las diferentes funciones que posee VBA para aplicarlas en nuestros procedimientos. Más adelante, estudiaremos las estructuras de decisión y las estructuras de control repetitivas que podemos utilizar en los algoritmos. Profundizaremos en la programación orientada a objetos y aprenderemos a trabajar con los principales objetos de Excel, sus propiedades y métodos.

Para finalizar, entraremos al mundo de los formularios y sus diferentes controles, que nos permitirán crear una interfaz amigable para que el usuario pueda introducir, modificar y visualizar datos.

Todo esto lo haremos a partir de ejemplos que nos permitirán aprender de forma práctica este lenguaje. El código utilizado a lo largo de esta obra se puede descargar del sitio **http://www.redusers.com/premium/notas_contenidos/macrosexcel2013/** para que cada uno pueda probarlo y estudiarlo en profundidad.

*01

Introducción a la automatización

En este capítulo, conoceremos los conceptos básicos de la programación orientada a objetos para trabajar en Excel, creando macros para optimizar procesos que realizamos con frecuencia o hacer cálculos complejos que no podríamos realizar con una simple fórmula.

El lenguaje Visual Basic para Aplicaciones

Como seguramente sabemos, **Excel** es una potente aplicación incluida en el paquete **Microsoft Office** que nos proporciona herramientas y funciones destinadas a analizar, compartir y administrar datos. Además, Excel nos ofrece amplias posibilidades para automatizar tareas que realizamos de manera cotidiana, y para crear aplicaciones basadas en el procesamiento y el análisis de datos numéricos por medio de la creación de macros.

Visual Basic para Aplicaciones (*Visual Basic for Aplications*), también conocido por su abreviatura **VBA**, es un lenguaje de programación común a todas las aplicaciones del paquete Microsoft Office, como así también en otros programas como Corel Draw o Autocad.

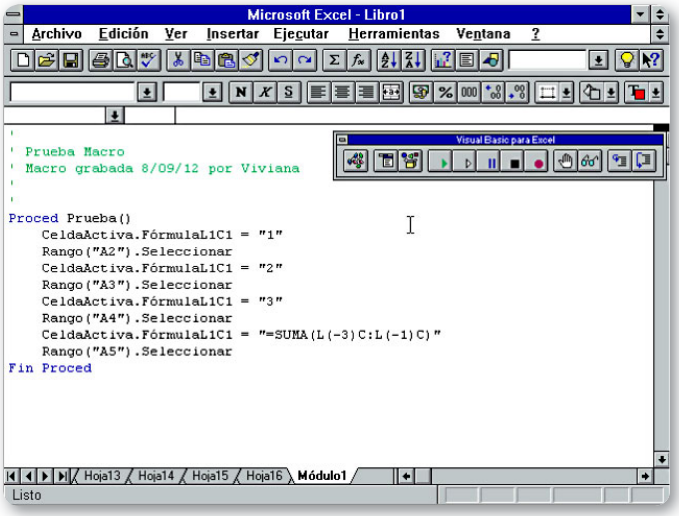


Figura 1. En esta imagen vemos como, en Excel 5.0, el módulo de VBA se inserta a la derecha de la última hoja del libro.

En sus orígenes, las macros se escribían en una hoja de macros separada de la hoja de cálculo y se guardaban en un archivo con la extensión .XLM. En la versión 5.0 del año 1993, Microsoft incorporó Visual Basic para Aplicaciones, y el código de programación de las

macros se escribía en módulos, que son hojas de un libro de Excel al igual que las hojas de cálculo o las hojas de gráficos.

En la versión 97, se implementaron grandes cambios, y los módulos dejaron de ser visibles en la ventana de aplicación de Excel; a partir de ahí, los módulos se pueden escribir y editar en la ventana del **Editor de Visual Basic**. Otro de los cambios que se introdujeron fueron los **módulos de clases**, con los que podemos crear nuevas funcionalidades en Excel.

> VBA ES UN LENGUAJE DE PROGRAMACIÓN COMÚN A TODAS LAS APLICACIONES DE MICROSOFT OFFICE

A partir de la versión 2007, los libros que contienen código Visual Basic para Aplicaciones se guardan con un formato de archivo diferente (.XLSM) al del archivo estándar (.XLSX), y los conceptos de seguridad se rediseñaron; apareció, entonces, el **Centro de confianza**, que permite ejecutar macros sin necesidad de **certificados digitales**. En la versión de Excel 2013, desde la perspectiva de la programación, no se produjeron cambios significativos.

VBA y Visual Basic (VB)

Visual Basic es un lenguaje de programación orientado a objetos que permite crear aplicaciones. Visual Basic para Aplicaciones es una versión de Visual Basic que se encuentra embebido en las aplicaciones de Microsoft Office y que permite escribir un conjunto de instrucciones (**macros**) para programar los distintos objetos de Excel, como por ejemplo, una hoja de cálculo, un conjunto de celdas o de gráficos. De esta manera, es posible automatizar las tareas que realizamos en forma repetitiva en Excel, y hasta podemos crear aplicaciones.

CERTIFICADO DIGITAL

Llamado también **Certificado de clave pública** o **Certificado de integridad**, es un documento digital que nos permite identificarnos, firmar digitalmente un documento o efectuar transacciones de tipo comercial con total seguridad y apoyo legal. En síntesis, es la herramienta que nos permite tomar las medidas de seguridad adecuadas para mantener la confidencialidad e integridad de la información.

La diferencia entre Visual Basic y Visual Basic para Aplicaciones radica en que el primero nos permite, entre otras cosas, realizar ejecutables que se pueden instalar en cualquier computadora bajo el entorno Windows. En cambio, VBA solo permite escribir instrucciones que se ejecutarán dentro del entorno Excel.

VBA es un lenguaje de fácil aprendizaje. El código en VB y en VBA es parecido, por lo que, si tenemos algún conocimiento de programación en VB, podremos utilizarlo para comprender el lenguaje de macros.

Las macros

Venimos mencionando el término *macro,* pero ¿qué es una macro? Podemos decir que una **macro** es una secuencia de instrucciones escritas en lenguaje VBA que se almacenan en un módulo. Cuando invocamos una macro, ya sea presionando un botón o una combinación de teclas, se desencadenarán las instrucciones almacenadas en ella.

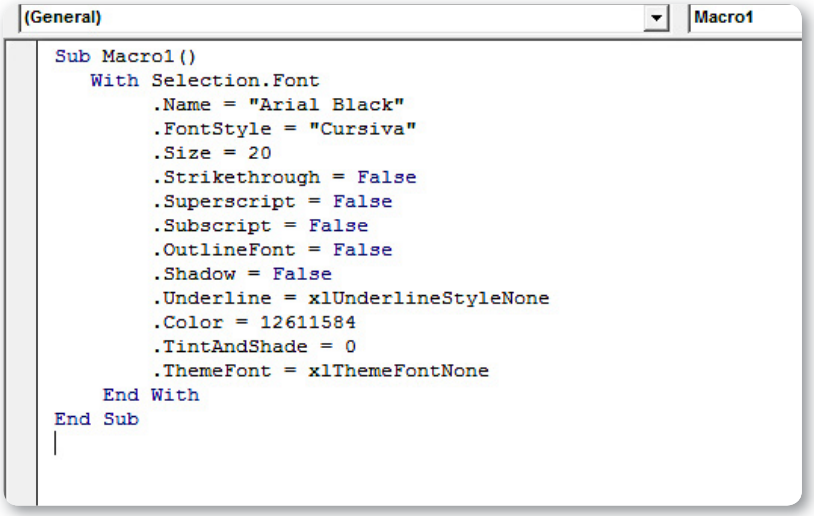


Figura 2. En esta imagen, podemos ver un conjunto de instrucciones escritas en VBA.

Con las macros no solamente podemos agilizar las tareas que realizamos con frecuencia, sino que también es posible ampliar la

funcionalidad de Excel, creando nuevas funciones para resolver cálculos que no podemos realizar con las funciones estándares del programa.

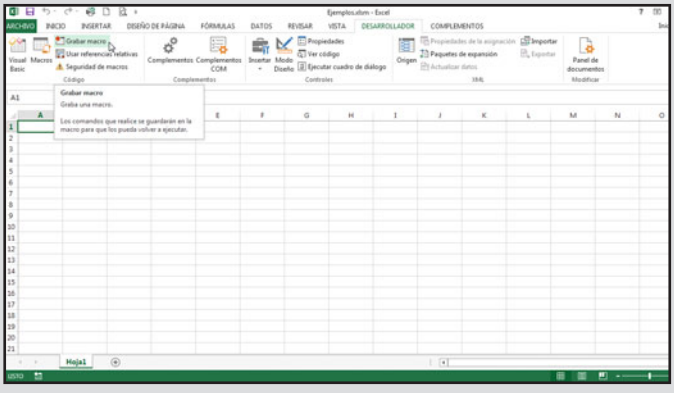
Tenemos dos maneras de crear una macro. Una es empleando la herramienta **Grabar macros**, y la otra es escribiendo las instrucciones en el **Editor de Visual Basic** que se encuentra embebido en Excel.

La manera más rápida y sencilla de crear una macro es utilizando la **grabadora de macros**, ya que no necesitamos tener ningún conocimiento previo de programación en VBA. Cuando empleamos la herramienta **Grabar macros**, lo que hacemos es ir grabando todas las operaciones (paso a paso) que vamos realizando en nuestro libro hasta que detenemos la grabación. A medida que realizamos las operaciones, la grabadora de macros las va convirtiendo al lenguaje VBA. Si bien emplear este método es más fácil, nos veremos limitados ya que solo podremos crear macros simples para llevar a cabo una tarea particular.

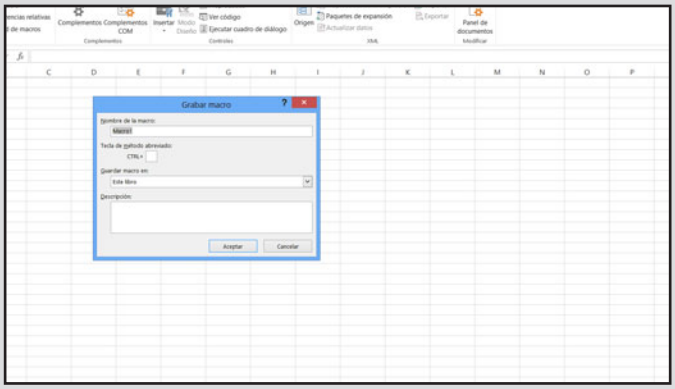
Si, en cambio, queremos crear macros más complejas, tendremos que hacerlo utilizando el lenguaje de programación. Sin detenernos mucho, en el siguiente paso a paso veremos cómo crear una macro que cambie el formato de texto de una celda (fuente, tamaño de la letra, color de la fuente, negrita) empleando la grabadora de macros.

PXP: CREAR UNA MACRO CON LA GRABADORA

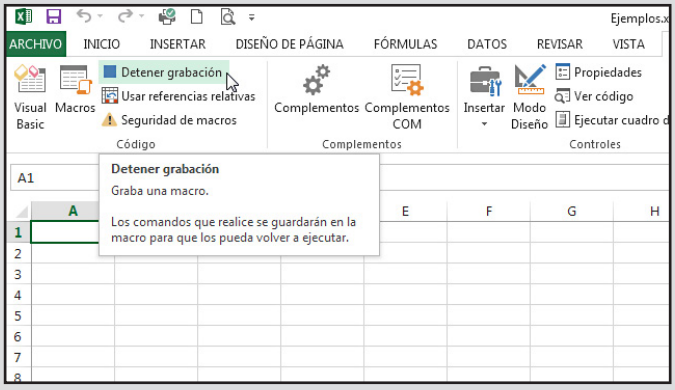
01 Haga clic en la ficha `Desarrollador` y presione el botón `Grabar macro`.



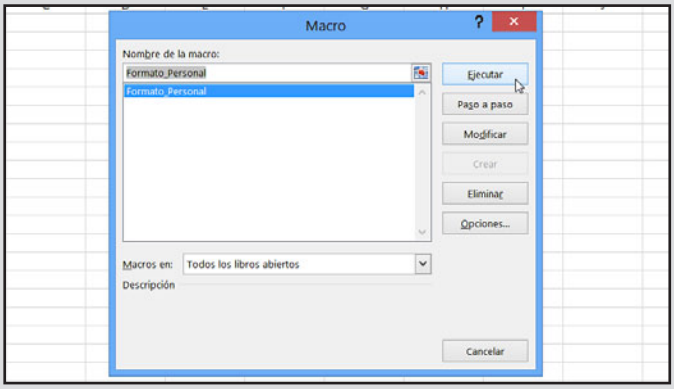
02 En Grabar macro, asigne un nombre a la macro; en este caso, Formato_personal. Establezca una combinación de teclas para la macro ingresando, en el cuadro de edición, una letra; en este caso, la A mayúscula. Finalmente, grabe la macro en el libro activo seleccionando la opción Este libro de la lista Guardar macro en. Por último, presione Aceptar para iniciar la grabación.



03 Use las opciones Fuente, Tamaño de letra, Color de fuente, Negrita del grupo lógico Fuente de la ficha Inicio, para darle formato a la tipografía. Luego de hacer este cambio, presione el botón Detener grabación.



04 Para probar la macro, sitúese en una celda, presione el botón `Macros` de la ficha `Desarrollador`, y, en el cuadro de diálogo `Macro`, seleccione la macro `Formato_Personal`. Luego, haga clic en el botón `Ejecutar` para ver cómo se modifica la letra de la celda.



Programación orientada a objetos

Visual Basic para Aplicaciones es un **lenguaje de programación orientada a objetos** (**POO**). Este concepto de programación se basa en la existencia de un mundo lleno de objetos. Excel, al igual que el resto de

NOMBRES DE MACROS

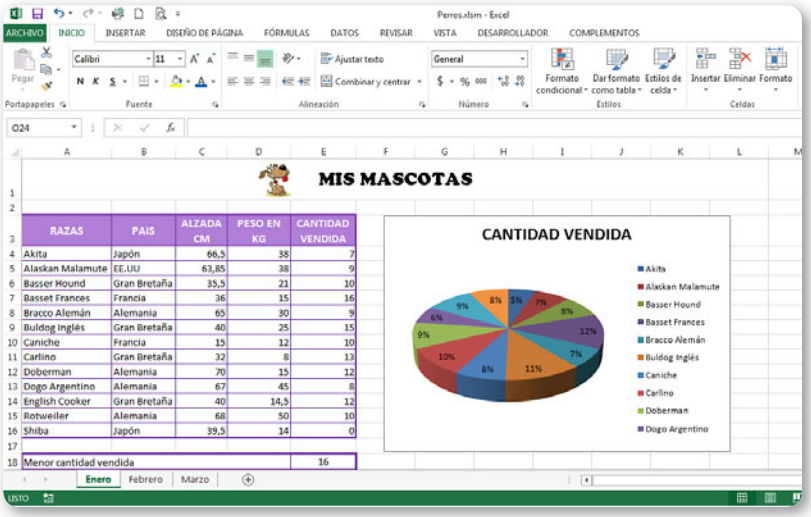
El nombre de una macro puede tener una longitud máxima de 255 caracteres. Debe comenzar siempre con una letra y solamente puede contener los siguientes caracteres: letras, números y el guion bajo. No puede contener caracteres especiales, como signos de exclamación (!) o signos de interrogación (?); y tampoco puede contener espacios en blanco.

las aplicaciones de Microsoft Office, nos ofrece un modelo de objetos que podremos manipular con VBA para cambiar sus propiedades. Por ejemplo, podemos cambiar los atributos de la fuente (nombre, tamaño, color) o, mediante métodos como **Seleccionar**, copiar un conjunto de datos.

Antes de involucrarnos en profundidad en el tema de la programación, veremos algunos conceptos importantes que nos servirán de base para crear nuestros algoritmos.

Objetos

Podemos decir que un **objeto** es algo tangible que se identifica por sus propiedades y sus métodos, por ejemplo, una persona, un avión, una mesa, un teléfono. Cada elemento de Excel es un objeto: un libro (**Workbook**), las hojas de cálculo (**Worksheet**), los rangos (**Range**), los gráficos (**Charts**) o una tabla dinámica (**PivotTable**) son algunos ejemplos de objetos del modelo de objetos de Excel.



RAZAS	PAIS	ALZADA CM	PESO EN KG	CANTIDAD VENDIDA
Akita	Japón	66,5	38	7
Alaskan Malamute	EE.UU	63,85	38	9
Basset Hound	Gran Bretaña	35,5	21	10
Basset Frances	Francia	36	15	16
Bracco Alemán	Alemania	65	30	9
Buldog Inglés	Gran Bretaña	40	25	15
Caniche	Francia	15	12	10
Carlino	Gran Bretaña	32	8	13
Doberman	Alemania	70	15	12
Dogo Argentino	Alemania	67	45	8
English Cooker	Gran Bretaña	40	14,5	12
Rotweiler	Alemania	68	50	10
Shiba	Japón	39,5	14	0
Menor cantidad vendida				16

Figura 3. En esta imagen, podemos ver algunos objetos de Excel: las hojas de cálculo, los rangos, un gráfico, una imagen.

Los objetos se agrupan en categorías o **colecciones**. Así, la clase **aviones**, por ejemplo, sería la representante de todos los aviones del mundo, mientras que el objeto **avión** será un avión en concreto. En el caso de Microsoft Excel, por ejemplo, la clase **Workbook** es la

representante de todos los libros de Excel, mientras que el objeto **Workbook** será un libro de trabajo en concreto.

Jerarquía de objetos

El modelo de objetos de Excel está compuesto por más de 192 objetos diferentes, algunos de ellos están ocultos, y otros se usan muy poco o se dejaron de utilizar.

Los objetos de Excel pueden contener otros objetos, y, a su vez, estos pueden contener otros objetos. Esto significa que tenemos un cierto orden o jerarquía..

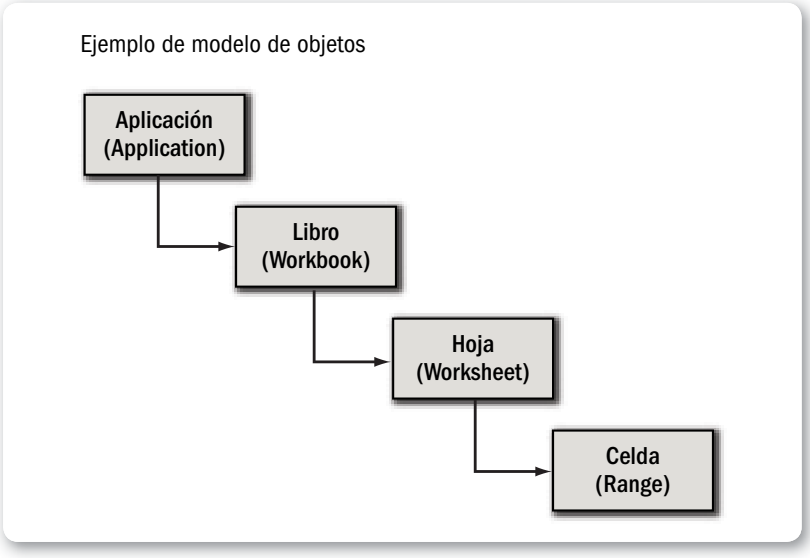


Figura 4. En este diagrama, vemos las colecciones de objetos más relevantes y su relación jerárquica.

ATAJOS POR DEFECTO

Debemos ser cuidadosos al asignar una combinación de teclas para ejecutar una macro, ya que podríamos suplantar el método abreviado asignado a otra macro o, aún peor, una combinación de acceso rápido que traiga Excel por defecto. Por ejemplo, si asignáramos la combinación **CTRL + C** a una macro, mientras el libro que la contiene estuviera abierto, el atajo no funcionará para copiar elementos al portapapeles.

Por ejemplo, cuando abrimos Excel, estamos abriendo una aplicación VBA representada por el objeto **Application**, y, al mismo tiempo, estamos abriendo un libro representado por el objeto **Workbook**, que contiene una hoja activa representada por el objeto **Woksheet**, y a su vez se activa una celda representada por el objeto **Range**

EL MODELO DE OBJETOS DE EXCEL CONTIENE MÁS DE 192 ELEMENTOS DIFERENTES

La disposición jerárquica de estos objetos se llama **modelo de objetos de Excel**, que está relacionado con la interfaz de usuario. Excel 2013 presenta una interfaz de documento único (SDI), que implica que cada libro tiene su propia ventana de aplicación de nivel superior con su correspondiente cinta de opciones.

- **Application** (aplicación): es el primer objeto en la jerarquía. Representa a la aplicación Excel. Este objeto actúa como contenedor de los demás objetos de Excel, por ejemplo: el libro, las hojas de cálculo, las celdas, los gráficos y muchos otros más. Por medio de este objeto, podemos acceder a las opciones de Excel como la de elegir si el cálculo es manual o automático, entre otras.
- **Workbook** (libro de trabajo): representa a un libro abierto dentro de la aplicación Excel. Este objeto se encuentra contenido en el objeto **Application** y mediante él podemos, por ejemplo, abrir un libro de trabajo, guardarlo, protegerlo o enviarlo a imprimir. El objeto **Workbook** es un miembro de la colección Workbooks.
- **Worksheet** (las hojas de cálculo): siguiendo el orden jerárquico, en tercer lugar tenemos el objeto **Worksheet**, que representa las hojas del libro con el que estamos trabajando. Con este objeto podemos, por ejemplo, cambiar el nombre de las hojas, insertar o eliminar hojas, protegerlas, etcétera.

ALGORITMO

Un **algoritmo** es un **método** para resolver un problema a través de una secuencia de pasos que nos llevará a cumplir un objetivo o solución, y esto se puede pasar a un lenguaje de programación. En programación, los algoritmos se implementan en forma de **instrucciones**. La forma de escribir los algoritmos va a depender del lenguaje de programación que utilicemos.

- **Range** (celda): este objeto es el que representa todas las celdas de la hoja, así como todas sus columnas y filas, los rangos, la selección de las celdas. Con este objeto podemos manipular todo lo relacionado con una celda o grupo de celdas, introducir valores, realizar operaciones, insertar celdas, entre otras cosas.

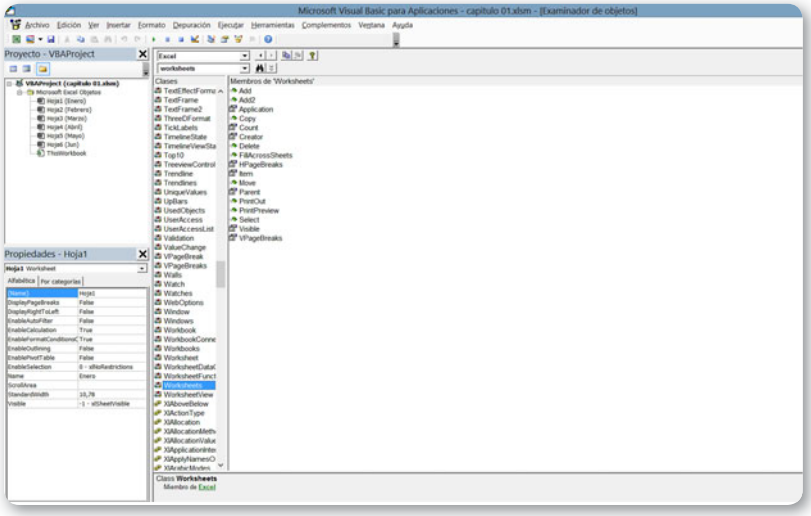


Figura 5. En esta imagen, podemos observar el conjunto de objetos **`Worksheets`**.

Referencia a objetos

Hacemos referencia a los objetos de Excel según la posición jerárquica que estos ocupan en el modelo de objetos y empleamos el punto para separar el objeto contenedor de uno o varios objetos. Es decir, el punto nos servirá para navegar por la jerarquía de objetos. Por ejemplo, para hacer referencia a un libro de Excel llamado **Planilla.xlsx**, escribimos la siguiente sentencia:

```
Application.Workbooks("planilla.xlsm")
```

Si, por ejemplo, quisiéramos hacer referencia a la celda **A1** de la hoja **Datos** del libro **Planilla.xlsx**, debemos indicar toda la ruta de la jerarquía del modelo de objetos:

```
Application.Workbooks("planilla.xlsm").Worksheets("datos").Range("A1").
Select
```

Para simplificar este procedimiento, podemos omitir la referencia **Application** en la mayoría de los casos. Esto es posible debido a que es el primer objeto de la estructura jerárquica y es el que contiene al resto de los objetos. Por lo tanto, también podemos hacer referencia a la celda **A1**, de una manera más sencilla, por medio del código que presentamos a continuación:

```
Workbooks("planilla.xlsm").Worksheets("datos").Range("A1").Select
```

Además, podemos omitir la referencia específica a un objeto si ese objeto se encuentra activo. Es decir, si solo tenemos un libro abierto, podemos omitir la referencia **Workbooks**. Siguiendo el ejemplo anterior, si el único libro abierto es **Planilla.xlsx**, podemos hacer referencia a la celda **A1** por medio del siguiente código:

```
Worksheets("datos").Range("A1").Select
```

Y algo más simple aún, si sabemos que la hoja activa es **Datos**, también es posible omitir el objeto **Worksheets**:

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

ORIGEN POO

El concepto de la Programación orientada a Objetos o POO (OOP, *Objects Oriented Programming*) no es nuevo. Tiene su origen en **Simula 67**, un lenguaje diseñado para hacer simulaciones que fue creado por **Ole-Johan Dahl** y **Kristen Nygaard** del Centro Noruego, en Oslo. POO es un paradigma de programación que usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos.

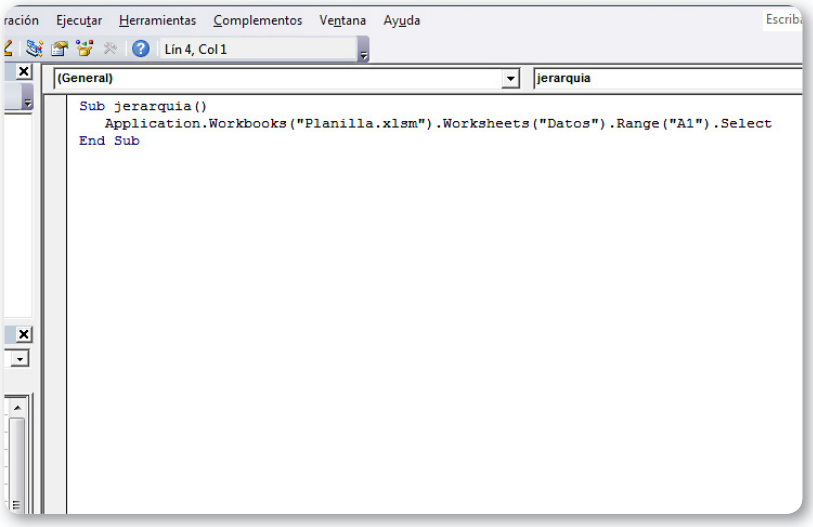


Figura 6. En esta imagen, podemos observar cómo hacer referencia a la celda según el modelo jerárquico.

Colecciones

Podemos decir que un conjunto de objetos del mismo tipo forma una colección. La **colección de objetos** nos permite trabajar con un grupo de objetos en lugar de hacerlo simplemente con un solo objeto. Por lo general, el nombre del objeto de la colección es el plural del nombre de los objetos contenidos dentro de la colección.

Por ejemplo, la colección denominada **Worksheets** es una colección de los objetos **Worksheet** que están contenidos dentro de un objeto **Workbook**. Podemos decir que la colección **Worksheets** es una colección dinámica, ya que irá variando a medida que vayamos agregando o eliminando hojas de cálculo a nuestro libro.

VARIABLES

En programación, las variables se utilizan para guardar un dato en la memoria. Este dato puede ser, por ejemplo, una cadena de caracteres (letras, números, símbolos), números y fechas, entre otros tipos. Más adelante, en el **capítulo 4** de este libro, conoceremos los tipos de datos y variables utilizados en el lenguaje Visual Basic para Aplicaciones.

Podemos hacer referencia a un objeto específico de una colección si colocamos el **número del índice** del objeto o el **nombre del objeto** de la colección. Para hacer referencia a un objeto de una colección, entonces, podemos usar alguna de las siguientes sintaxis:

```
Coleccion!Objeto
Coleccion![Objeto]
Coleccion("Objeto")
Coleccion(var)
Coleccion(index)
```

Donde **Coleccion** es el nombre de la colección, **Objeto** es el nombre del objeto, **Var** es una variable del tipo **String** que contiene el nombre del objeto, e **Index** representa el número del índice del objeto de la colección.

Por ejemplo, si queremos hacer referencia a la primera hoja que se llama **datos** del libro **planilla.xlsm**, podemos escribir algunas de las sentencias que presentamos a continuación:

```
Workbooks("planilla.xlsm").Worksheets("datos")
```

O bien:

```
Workbooks(1).Worksheets(1)
```

Si el libro activo es **planilla.xlsm**, la referencia anterior la podemos simplificar de la siguiente manera:

```
Worksheets!Datos
```

O bien:

```
Worksheets(1)
```

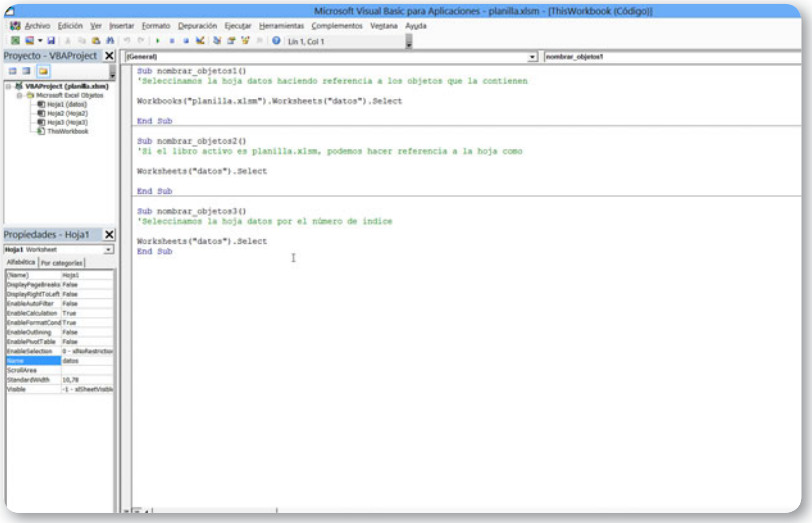


Figura 7. En esta imagen, vemos cómo podemos hacer referencia a la hoja **datos**.

Propiedades, métodos y eventos

Los objetos tienen propiedades, métodos y eventos que definen las características y propiedades del objeto.

Las **propiedades** son un conjunto de características propias del objeto que hacen que se distinga de otro objeto, tales como su nombre, tamaño, color, localización en pantalla.

Por ejemplo, si una persona fuera un objeto de Excel, sus propiedades serían su altura, su peso y su color de cabello, mientras que, si un avión fuera un objeto de Excel, sus propiedades serían el tipo de motor, su tamaño, su color.

> LAS PROPIEDADES, MÉTODOS Y EVENTOS DEFINEN LAS CARACTERÍSTICAS DE LOS OBJETOS

De la misma manera, los objetos de Excel tienen propiedades. Por ejemplo, el objeto **Range** tiene las propiedades **Name** (nombre), **Value** (valor) y **Column** (columna), entre muchas otras.

El uso de las diferentes propiedades nos va a permitir ver o cambiar las características del objeto. Por ejemplo, podemos utilizar la propiedad **Value** para modificar el valor de una celda. También es posible utilizar las propiedades para cambiar el aspecto de un objeto,

por ejemplo, podemos utilizar la propiedad **Hidden** para ocultar o mostrar un objeto determinado.

Algunas propiedades pueden ser también objetos. Por ejemplo, si queremos cambiarle la fuente al objeto **Range**, utilizaremos la propiedad **Font** (fuente). Como las fuentes tienen distintos nombres (Arial, Times New Roman, Comic Sans Ms), diferentes tamaños (10, 12, 14) y distintos estilos (negrita, cursiva, subrayado), estas son las propiedades de la fuente. Por consiguiente, si la fuente tiene diferentes propiedades, entonces la fuente es también un objeto.

Para hacer referencia a las propiedades de cualquier objeto, empleamos la siguiente sintaxis:

```
Objeto.Propiedad = valor
```

Donde **Objeto** es el nombre del objeto, **Propiedad** es el nombre de la propiedad que cambiamos y **Valor** se refiere al valor que se le asigna a la propiedad. Por ejemplo, para asignarle el valor **27** a la celda **A2**, escribimos la siguiente sintaxis:

```
Range("A2").Value = 27
```

Otra opción sería cambiar el tamaño de la fuente de la celda **A2**, empleando la siguiente sintaxis:

```
Range("A2").Font.Size = 25
```

Un **método** es un conjunto de comportamientos o acciones que puede realizarse en el objeto. Podríamos decir que son las órdenes que le damos al objeto para que haga algo sobre sí mismo.

Siguiendo con el ejemplo de una persona, sus métodos serían hablar, caminar, comer o dormir. En el caso del avión, los métodos serían carretear, girar, despegar. El objeto **Range** (celda), por ejemplo, tiene los métodos **Activate** (activar) y **Clear** (borrar), entre otros.

Para implementar los métodos de un objeto, utilizamos la sintaxis que presentamos a continuación:

```
Objeto.Método
```

Donde **Objeto** es el nombre del objeto y **Método** es el método que queremos ejecutar. Por ejemplo, para activar la celda **A2** de la hoja activa, escribimos la siguiente sentencia:

```
Range("A2").Select
```

Algunos métodos tienen **argumentos** que a veces son necesarios y otras, opcionales. Los argumentos nos permiten especificar en forma más amplia las opciones para la acción que vamos a ejecutar. Por ejemplo, si queremos guardar el libro activo con el nombre **Empleado.xlsm**, escribimos la siguiente sentencia:

```
ThisWorkbook.SaveAs Filename:="Empleados.xlsm"
```

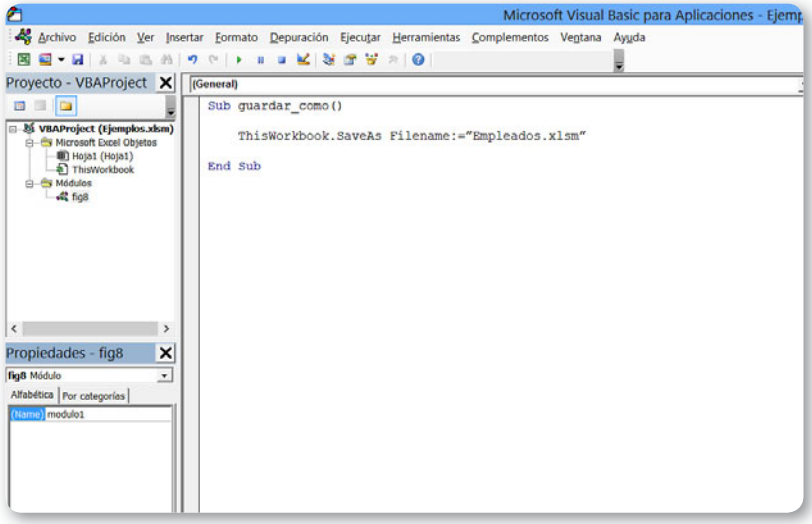


Figura 8. En esta imagen, vemos el código VBA para guardar un archivo con otro nombre.

Además de las propiedades y los métodos, cada objeto tiene sus eventos. Podemos definir **evento** como la acción que puede

ser reconocida por un objeto. Por medio de los eventos, es posible controlar el momento exacto en el que deseamos ejecutar un conjunto de instrucciones (procedimientos). Ejemplos de eventos son **abrir un libro, imprimir, cambiar el contenido de una celda, hacer clic.**

LOS OBJETOS DE EXCEL TIENEN UN CONJUNTO DE EVENTOS QUE LES PUEDEN OCURRIR

En Visual Basic para Aplicaciones, los diferentes objetos de Microsoft **Excel** tienen un conjunto de eventos que les pueden ocurrir.

Por ejemplo, el evento más típico de un botón es el **Click** que se produce cuando lo presionamos. Otro ejemplo de un evento frecuente para una hoja de cálculo es **Change**, que se produce cada vez que cambiamos de celda.

Que suceda algo como respuesta a un evento dependerá de que hayamos programado una acción en el procedimiento de dicho evento.

Por ejemplo, si queremos que cada vez que cerramos el libro de trabajo se muestre un mensaje de despedida del sistema, tendríamos que escribir, en el evento **BeforeClose** del objeto **Workbook**, el código que presentamos a continuación:

```
Private Sub Workbook_BeforeClose(Cancel As Boolean)
    Dim Mensaje As String

    Mensaje = "Muchas gracias por usar el Sistema de Facturación"

    MsgBox Mensaje, vbInformation, "Hasta Pronto"
End Sub
```

MACROS DISPONIBLES

Si empleamos los métodos abreviados de teclado, podremos acceder al cuadro de diálogo **Macros** de la ficha **Desarrollador**, donde encontraremos un listado de las diferentes macros que se encuentran disponibles y las opciones para ejecutarlas, modificarlas o eliminarlas. Para acceder a esta lista, debemos presionar la combinación de teclas **Alt + F8**.

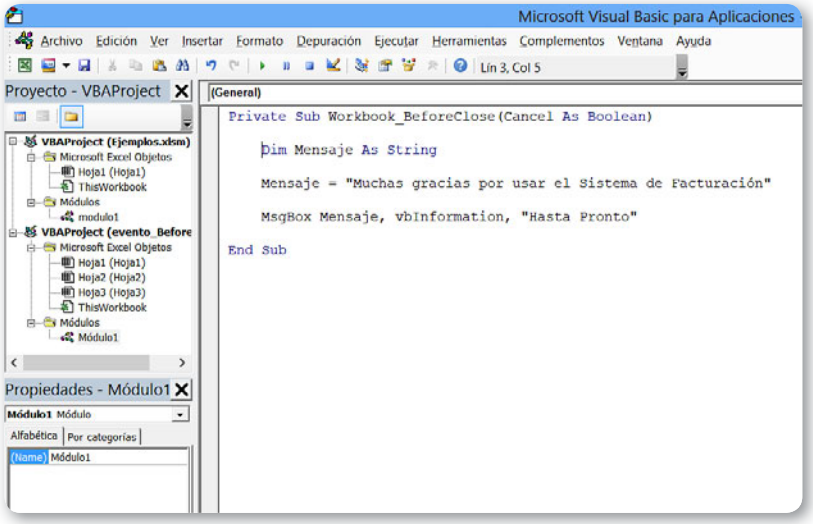


Figura 9. En esta imagen, vemos el código VBA que se ejecutará cuando cerremos el archivo.

Private Sub Workbook_BeforeClose(Cancel As Boolean) es un procedimiento que se ejecuta cada vez que cerramos el libro de trabajo, y la acción, o evento, es dar el mensaje.

Aunque este tema, al principio, parece algo muy complejo de entender y de realizar por nosotros mismos, en los próximos capítulos veremos más en detalle las diferentes propiedades, los métodos y los eventos que poseen los principales objetos de Excel, y también los llevaremos a la práctica. De esta manera aprenderemos a manejarlos.

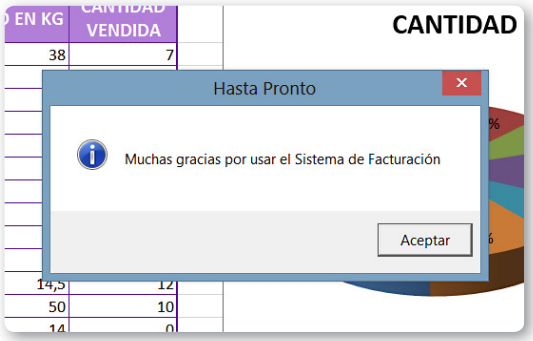


Figura 10. Al cerrar el libro de trabajo, aparecerá la ventana con el mensaje que hemos programado en el evento **BeforeClose** del objeto **Workbook**.

La ficha Desarrollador

Desarrollador es una de las fichas de la cinta de opciones de la versión 2013 de Office, que en las versiones 2007 y 2010 se denominaba **Programador**. En ella, encontraremos los comandos que emplearemos para crear, editar y ejecutar macros, así como lo necesario para gestionar **complementos** y crear **formularios**.

> LA FICHA DESARROLLADOR NOS PERMITE TRABAJAR CON LAS MACROS

Por defecto, la ficha **Desarrollador** no se encuentra visible, por lo tanto, lo primero que haremos para trabajar con macros es activarla. Para activarla en Excel 2013, vamos a la ficha **Archivo** y seleccionamos **Opciones**. En la parte izquierda del cuadro de diálogo **Opciones de Excel**, seleccionamos **Personalizar cinta de opciones** y, en la sección **Pestañas Principales**, marcamos la casilla de verificación de la ficha **Desarrollador** para hacerla visible y, por último, pulsamos **Aceptar**.

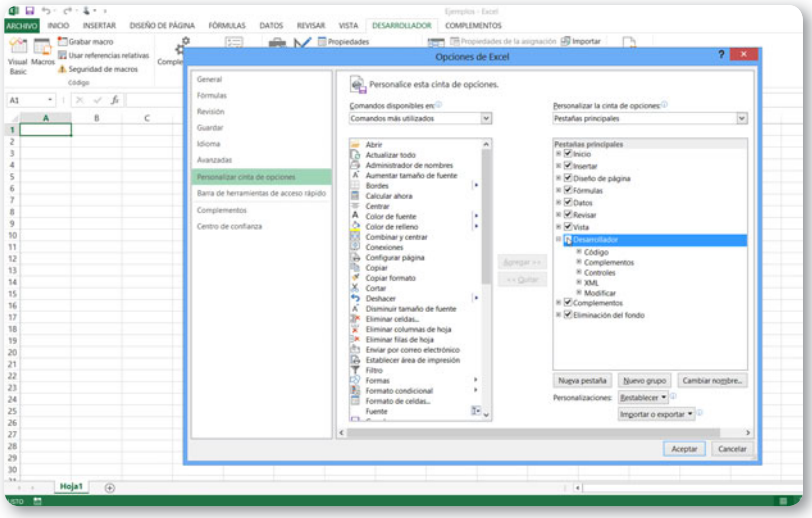
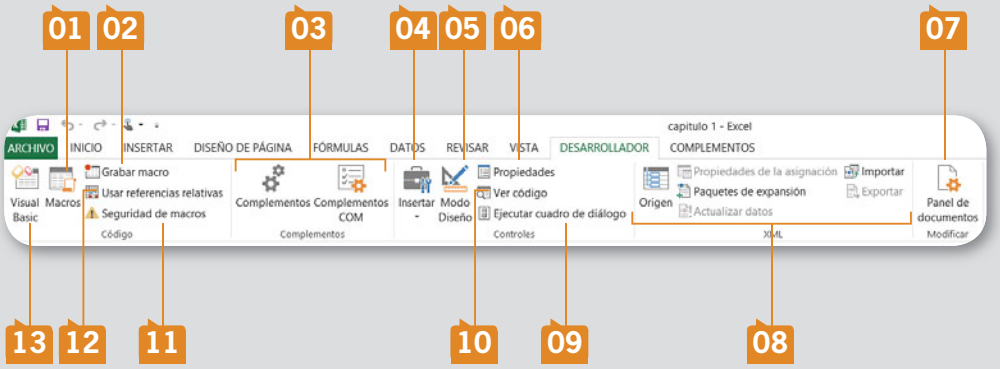


Figura 11. Para poder trabajar con macros, tenemos que modificar las opciones de Excel para activar la ficha **Desarrollador**.

En la ficha **Desarrollador**, podemos distinguir los grupos lógicos **Código**, **Complementos**, **Controles**, **XML** y **Modificar**, que contienen los comandos que veremos en la siguiente guía visual.

GV: GRUPOS DE LA FICHA DESARROLLADOR



01 **Macros:** abre el cuadro de diálogo **Macro**, donde veremos y podremos ejecutar todas las macros incluidas en los libros abiertos. También podremos modificarlas, eliminarlas o cambiarles la combinación de teclas si pulsamos el botón **Opciones...**.

02 **Grabar macro:** con este botón iniciamos el proceso de grabación de una macro.

03 **Complementos:** este grupo nos permite administrar y habilitar los complementos disponibles para usar con el archivo, como **Solver**, **Herramientas para análisis-VBA**, etcétera.

04 **Insertar:** este botón nos permite insertar los distintos controles que nos provee Excel 2013 para generar interfaces amigables: **Controles de Formulario** y **Controles ActiveX**. Los primeros son más simples de usar, pueden ser directamente asignados a una macro o pueden ser utilizados para manipular listas, textos, barras de desplazamiento, casillas de verificación y selección, etcétera. Los segundos son más flexibles y ofrecen muchas posibilidades, pero requieren ser programados con VBA.

05 **Modo Diseño:** con este botón, activamos o desactivamos el modo diseño del proyecto. El modo diseño es el tiempo durante el cual no se ejecuta el código del proyecto ni los eventos de la aplicación principal.

06 **Propiedades:** con este botón, podemos cambiar las propiedades de los controles ActiveX o las opciones de la hoja seleccionada.

07 **Modificar:** este grupo contiene el comando **Panel de documentos**, donde podremos modificar las propiedades del panel de información del documento, tales como título, nombre del autor, asunto y palabras clave que identifican el tema.

08 **XML:** dentro de este grupo, encontramos las herramientas que nos permiten importar datos de un archivo XML o exportar los datos como un archivo XML. XML (*Extensible Markup Language*) es un metalenguaje que sirve para definir lenguajes de marcado o de etiquetas.

09 **Ejecutar cuadro de diálogo:** esta opción nos permite ejecutar un cuadro de diálogo personalizado o Userform (formulario) que diseñamos en VBA

10 **Ver código:** al presionar este botón, se abrirá el editor de Visual Basic, donde veremos el código asociado al control que hemos seleccionado.

11 **Seguridad de macros:** con este botón, podemos personalizar la seguridad de las macro, es decir, desde aquí es posible habilitar o deshabilitar la ejecución de las macros.

12 **Usar referencias relativas:** este botón nos permite especificar si deseamos que las referencias de las celdas se graben en forma absoluta o relativa. Podemos cambiar de una forma a otra en cualquier momento, incluso, durante la grabación de la macro.

13 **Visual Basic:** con este icono, accedemos al Editor de Visual Basic. Esta es la herramienta que vamos a emplear para realizar todo el trabajo de programación. También podemos acceder a ella presionando la combinación de teclas **Alt + F11**.

Archivos y seguridad

La funcionalidad de automatización proporcionada por las macros hizo que Excel fuera propenso a sufrir por virus informáticos programados para macros. Por tal motivo, Microsoft tomó medidas para prevenir el uso indebido de Excel, estableciendo un formato de archivo especial para guardar los libros que contienen macros. Además, incluyó varios niveles de seguridad.

ACCEDER CON EL TECLADO

Empleando los métodos abreviados de teclado, podemos acceder las diferentes opciones de la ficha **Desarrollador**. Presionamos primero la combinación de teclas **Alt + G** y, seguidamente, la letra correspondiente a la opción deseada. Por ejemplo **Alt + G + MA** para abrir el cuadro de diálogo **Macro**.

Formato de archivos

La extensión de un archivo nos brinda información importante sobre un conjunto de características que definen el tipo de archivo, el formato, la clase de datos que contiene y el programa que requiere la computadora para visualizarlo o poder editarlo.

Al igual que en la versión 2007 y 2010, los archivos creados en Excel 2013 se guardan en el formato **Office Open XML**. Por defecto, Excel 2013 asigna a sus archivos la extensión .XLSX. Este es un formato de archivo seguro, que no admite la grabación de macros ni controles ActiveX. Para guardar un archivo que contiene macros, debemos utilizar el formato .XLSM.

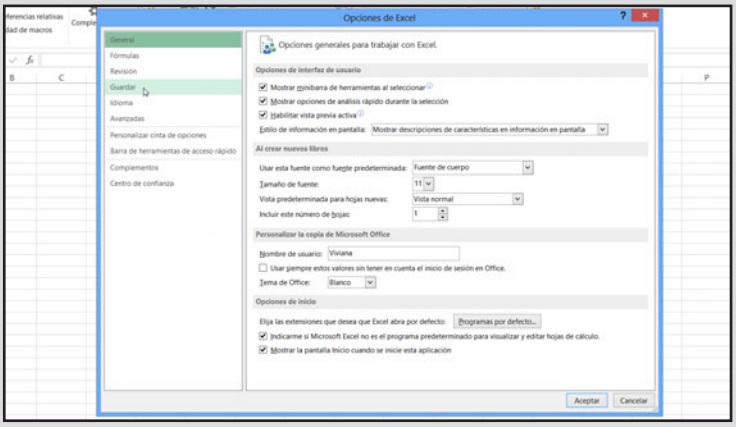
> PARA GUARDAR UN ARCHIVO QUE CONTIENE MACROS UTILIZAMOS EL FORMATO .XLSM

Para evitar tener que seleccionar el tipo de archivo habilitado para macros cada vez que guardamos un libro con código VBA, podemos predeterminar el formato .XLSM siguiendo estos pasos:

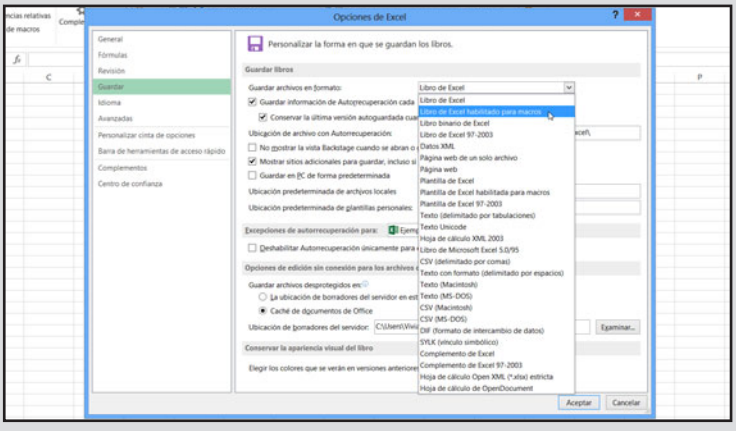
PXP: CAMBIAR EL FORMATO PREDETERMINADO DE ARCHIVO

01 Haga clic en la ficha `Archivo` de la cinta de opcionesy en el panel de la izquierda seleccione `Opciones`.

02 A continuación, se abrirá el cuadro de diálogo denominado `Opciones de Excel`, diríjase al panel que se encuentra en el sector izquierdo de la ventana y seleccione la tercera categoría de la lista: `Guardar`.



03 Dentro de la sección `Guardar libros`, en el menú desplegable `Guardar archivos en formato`, seleccione `Libro de Excel habilitado para macros (*.xlsm)` y para finalizar haga clic en `Aceptar`.



La principal razón de diferenciación de los archivos .XLSX con los archivos .XLSM es debido a razones de seguridad. En las versiones anteriores a Excel 2007, la existencia de macros en una hoja de cálculo era imperceptible a menos que tuviéramos habilitado los niveles de seguridad referentes a las macros.

Ahora, con solo mirar la extensión del archivo, podemos identificar que el libro de Excel contiene alguna macro y, así, decidir si queremos abrir o no el archivo, en función de la confianza que nos genera la persona que lo ha creado.

La seguridad

Siempre que abramos un archivo que contenga código VBA, en forma predeterminada Excel nos mostrará un mensaje justo debajo de la cinta de opciones, en el que nos dice que ha deshabilitado parte del contenido activo.

Esto se debe a que hace algunos años se pusieron de moda los **virus de macro**. Estos virus se aprovechaban de los archivos aparentemente inofensivos, como los documentos de Word o las planillas de Excel, para realizar diversos ataques que incluían borrado de archivos de los discos, envío de archivos por e-mail sin que el usuario se diera cuenta, envío de documentos para imprimir en forma inesperada, entre otras acciones. Por ese motivo, Microsoft incluyó protección para ayudarnos a protegernos de este tipo de virus.

Para poder trabajar con un libro que contiene macros, debemos configurar las restricciones de seguridad de Excel, ya que así podremos controlar lo que sucede al abrir un archivo que contiene código VBA. En las próximas páginas, veremos algunas opciones y posibilidades que tenemos para que Excel 2013 no deshabilite el código de las planillas.

MICROSOFT OFFICE OPEN XML

Este formato, utilizado en las suites de Office 2007, 2010 y 2013, presenta las siguientes ventajas: reduce el tamaño de los archivos respecto al formato de las versiones anteriores, mejora la recuperación de datos en archivos dañados, proporciona mayor seguridad al diferenciar los archivos que contienen macros y facilita el compartir datos entre programas y sistemas operativos.

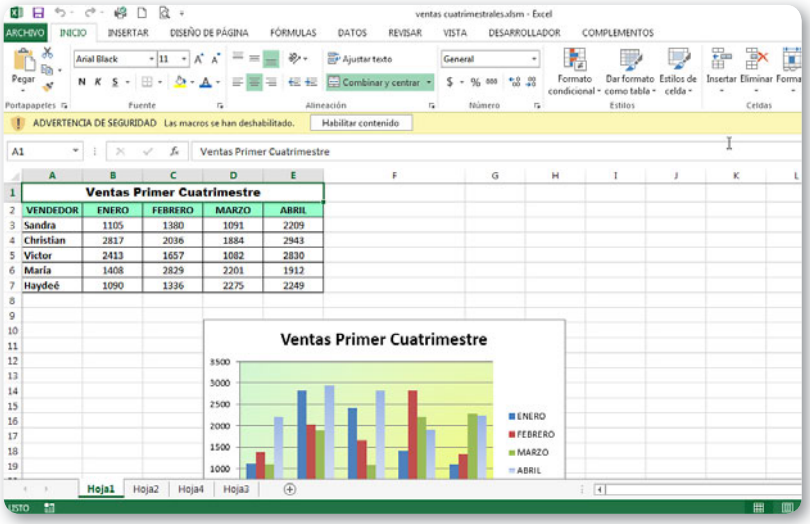


Figura 12. Para protegernos, Excel nos muestra una barra con un mensaje de advertencia de seguridad que nos avisa que existe contenido que podría ser inseguro.

Ubicaciones de confianza

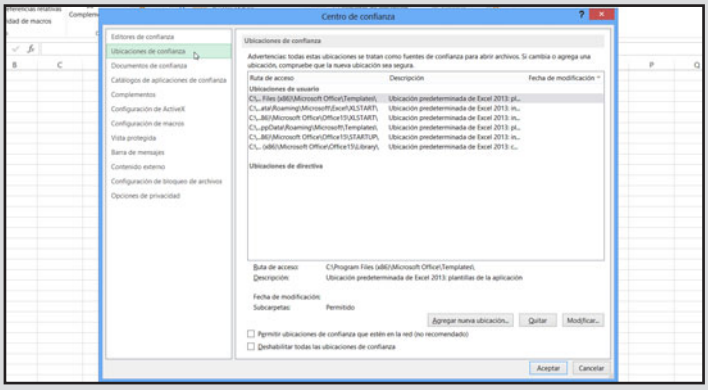
Como mencionamos con anterioridad, cada vez que abrimos un libro que contiene macros, nos aparece un cuadro de diálogo con una advertencia de seguridad que nos indica que las macros se han deshabilitado. Esto se debe a que Excel considera que el archivo que estamos abriendo puede ser poco seguro porque no se encuentra guardado en una de las ubicaciones que la aplicación califica que son de confianza.

EXCEL CONSIDERA QUE LOS ARCHIVOS QUE CONTIENEN MACROS PUEDEN NO SER DE CONFIANZA

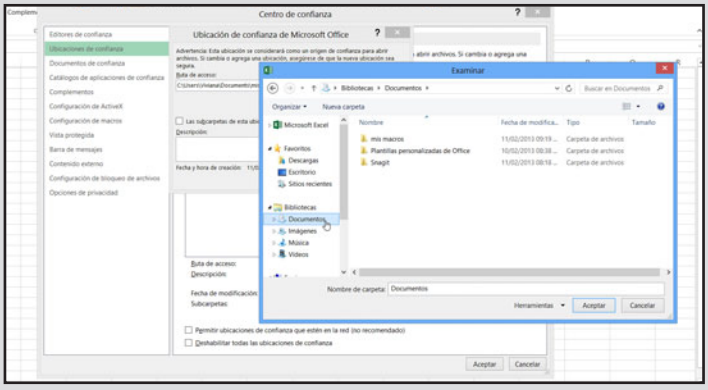
Una **ubicación de confianza** es una carpeta cualquiera de nuestro disco, cuyo contenido consideramos que está libre de amenazas. Podemos tener tantas carpetas de confianza como sea necesario para guardar en ellas nuestros archivos. De esta manera, cualquier archivo que contenga código VBA y que se encuentre guardado en esa carpeta se abrirá con las macros habilitadas automáticamente, sin mostrar ningún mensaje de advertencia. A continuación, veamos paso a paso cómo podemos crear una ubicación de confianza.

PXP: CREAR UNA UBICACIÓN DE CONFIANZA

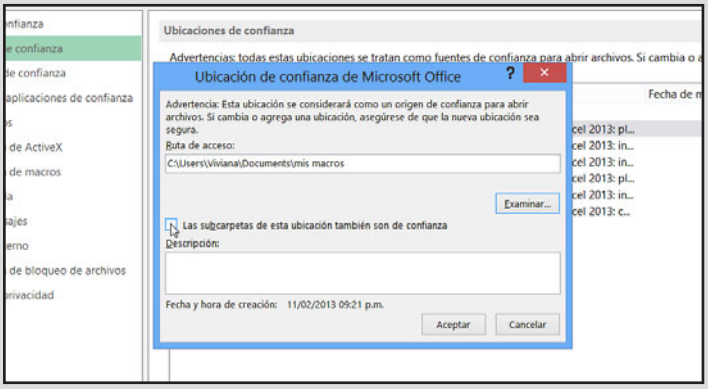
01 Haga clic en el botón Seguridad de macros de la ficha Desarrollador. En el cuadro de diálogo Centro de confianza, seleccione la opción Ubicaciones de confianza.



02 Haga clic en el botón de opción Agregar nueva ubicación... y, en la ventana Ubicación de confianza de Microsoft Office que se abre, presione el botón Examinar.... En el cuadro de diálogo Examinar, seleccione una carpeta del disco. Luego de ubicarla, haga clic en Aceptar.



03 Para incluir las subcarpetas de la carpeta de confianza, marque la opción `Las subcarpetas de esta ubicación también son de confianza` y presione `Aceptar`.



La seguridad: centro de confianza

Cuando no guardamos los archivos en una ubicación de confianza, podemos controlar lo que ocurre cuando abrimos un archivo que contiene macros si cambiamos la configuración de seguridad de las macros.

Podemos ver la configuración de seguridad de macros de Excel 2013 si hacemos clic en el botón **Seguridad de macros** del grupo **Código** de la ficha **Desarrollador**. En el cuadro de diálogo **Configuración de macros**, encontramos las siguientes opciones de configuración:

SEGURIDAD

Las modificaciones de configuración de seguridad de macros que hagamos en Excel no afectarán a las otras aplicaciones de Microsoft Office. Estas modificaciones se aplicarán a todos los libros de Excel, exceptuando los libros que ya tenemos abiertos. Para que las modificaciones afecten a esos libros, debemos cerrarlos y volverlos a abrir.

- **Deshabilitar todas las macros sin notificación**: cuando abrimos un archivo con macros, esta opción deshabilita su ejecución sin mostrar ningún mensaje de advertencia. Únicamente se ejecutarán las macros que se han guardado en una ubicación de confianza.
- **Deshabilitar todas las macros con notificación**: todas las macros serán deshabilitadas pero, al abrir un archivo que contiene macros, se mostrará una advertencia de seguridad que da la posibilidad de habilitar o no las macros para ese archivo en particular. Si optamos por habilitar las macros, habilitamos el código para siempre. Es decir, la próxima vez que abramos ese archivo, el código se habilitará en forma automática.

> PODEMOS CAMBIAR LA SEGURIDAD DE LAS MACROS DESDE LA FICHA DESARROLLADOR

- **Deshabilitar todas las macros excepto las firmadas digitalmente**: habilita, de manera automática, las macros que llevan una firma digital de un origen aprobado e impide la ejecución de las restantes macros, mostrándonos un mensaje de notificación.
- **Habilitar todas las macros (no recomendado; puede ejecutarse código posiblemente peligroso)**: esta opción habilita la ejecución de todas las macros. No es recomendable esta opción, ya que quedaríamos expuestos a un ataque de virus de macros.

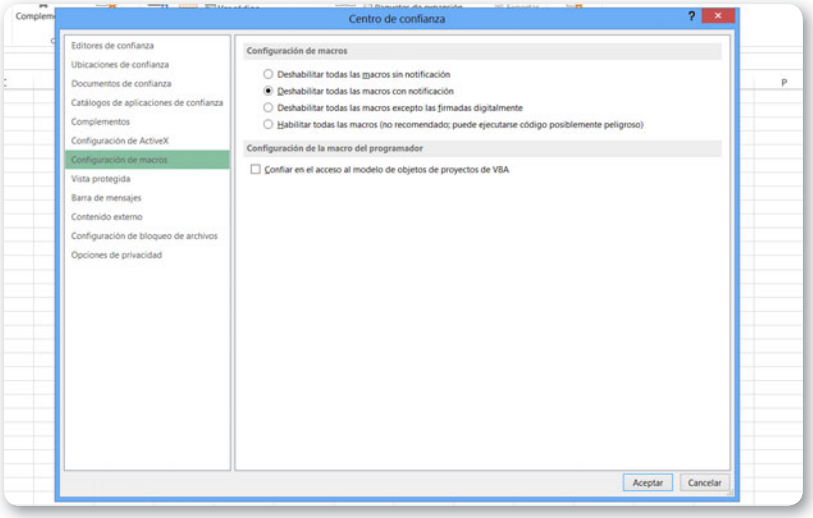


Figura 13. En el centro de confianza, podemos configurar las opciones de advertencia cuando se abre un libro con macros.

Quitar archivos de la lista de documentos confiables

Cada vez que presionamos el botón **Habilitar contenido** en el mensaje de advertencia de Excel 2013, habilitamos el código de la macro para siempre. De esta manera, el archivo queda habilitado ya que pasa a la lista de documentos confiables.

Si queremos quitar un archivo de la lista de documentos confiables, presionamos el botón **Seguridad de macros** y, en el cuadro de diálogo **Centro de confianza**, seleccionamos la opción **Ubicaciones de confianza**, a continuación, presionamos el botón **Quitar**. De esta manera, no solo quitaremos el archivo seleccionado, sino también todos los archivos que se encuentran incluidos en la lista.

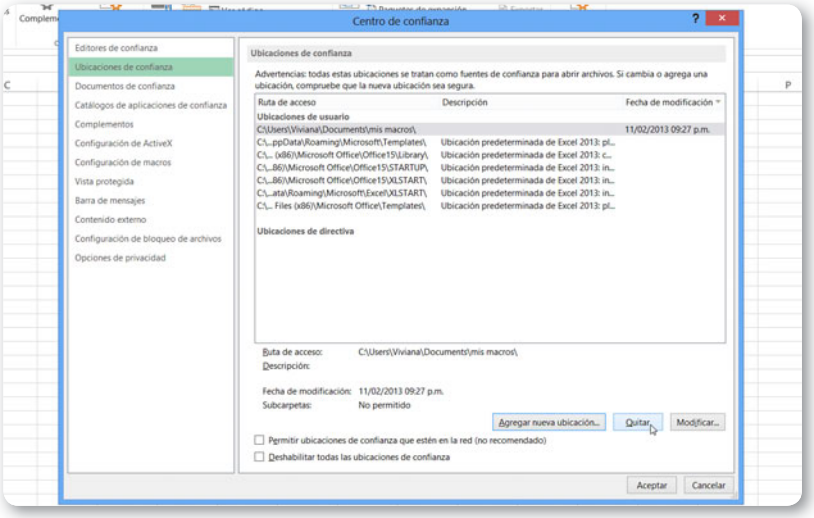


Figura 14. Desde el cuadro de diálogo **Ubicaciones de confianza**, podremos quitar una carpeta de confianza.

MOSTRAR EXTENSIONES DE ARCHIVOS

Por defecto, en Windows 8 las extensiones de los archivos están ocultas. Para visualizarlas, abrimos el **Explorador de Windows 8** y vamos a **Vista/Opciones/Cambiar opciones de carpeta y búsqueda**. En la ficha **Ver**, desmarcamos **Ocultar las extensiones de archivo para tipos de archivo conocidos**.

Certificados digitales

Como mencionamos antes, podemos habilitar automáticamente las macros que llevan una firma digital. Para ello, debemos tener un certificado digital.

Un **certificado digital** es un archivo que funciona como credencial de identidad en el universo digital. Cada certificado es único para cada persona y, con él, podemos firmar digitalmente documentos, certificando su autenticidad. Esta firma no tiene valor formal si no está legalizada por una entidad de certificación acreditada, como la **Jefatura de Gabinete de Ministros** en la República Argentina (**https://pki.jgm.gov.ar/app/**).

Para firmar digitalmente el código VBA de un libro de Excel, debemos hacer clic en el botón **Visual Basic** de la ficha **Desarrollador**. Luego, hacemos clic en el menú **Herramientas**, seleccionamos **Firma digital...** y presionamos el botón **Elegir...** para seleccionar nuestro certificado.

RESUMEN

Microsoft Excel 2013 es una herramienta muy eficaz que se puede usar para manipular, analizar y presentar datos. Al igual que otras aplicaciones de Office, Excel incluye el lenguaje de programación Visual Basic para Aplicaciones. En este capítulo, vimos las características del lenguaje, explicamos las relaciones entre objetos, qué son las propiedades, los métodos y los eventos. Estos conceptos nos servirán de base para poder crear y manipular nuestras propias macros empleando VBA. También vimos cómo podemos configurar la seguridad para ejecutar los archivos que contienen macros.

Actividades

TEST DE AUTOEVALUACIÓN

1. Explique que es VBA.
2. ¿Cómo se pueden crear las macros?
3. ¿A qué se denomina modelo de objetos de Excel?
4. Explique qué son las propiedades, los métodos y los eventos de un objeto.
5. ¿Cuál es el formato de archivo para guardar un libro con macros?

EJERCICIOS PRÁCTICOS

1. Active la ficha Desarrollador en la cinta de opciones.
2. Cree una carpeta y configúrela como ubicación de confianza.
3. Cambie las opciones de seguridad de macros desde el Centro de Confianza.
4. Predetermine el formato habilitado para macros en el cuadro de diálogo Guardar como.
5. Busque información en Internet sobre editores y ubicaciones de confianza.

*02

El Editor de Visual Basic para Excel

Luego de haber recorrido los principales conceptos de la programación orientada a objetos, en este capítulo conoceremos el entorno de programación de Excel: el Editor de Visual Basic. Describiremos sus componentes fundamentales y las principales herramientas para escribir, editar o eliminar macros y, además, crear aplicaciones.

↗↗↗

¿Qué es el Editor de Visual Basic?

El **Editor de Visual Basic**, también llamado **VBE** (*Visual Basic Editor*), es la herramienta que nos servirá para desarrollar, probar y modificar las macros que utilizaremos junto con los libros de trabajo.

En este capítulo nos dedicaremos a conocer los principales componentes de su interfaz y sus herramientas más importantes.

Se ejecuta en su propia ventana, separada de la de Excel, y podemos acceder a él de diferentes maneras:

- Haciendo clic en el botón **Visual Basic** de la ficha **Desarrollador**.
- Cuando ya tenemos una macro creada, pulsando el botón **Modificar** del cuadro de diálogo **Macros**, que aparece al presionar el botón **Macros** de la ficha **Desarrollador**.
- Presionando la combinación de teclas **Alt + F11**.

En esta ventana, encontramos las clásicas barra de menú y barra de herramientas ubicadas en la parte superior de la ventana.

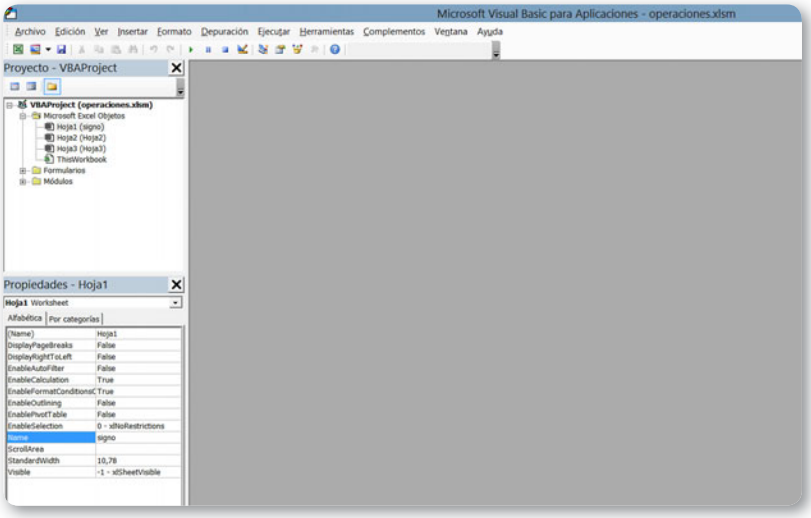


Figura 1. La ventana del Editor de Visual Basic mantiene la interfaz de usuario de las versiones anteriores a Microsoft Office 2007 a 2013.

El entorno de VBE

A continuación, conoceremos en detalle algunas de las herramientas principales que integran la ventana del Editor de Visual Basic.

La barra de menú

A través de la barra de menú accedemos a la mayoría de las funciones de VBE para desarrollar, comprobar y guardar las macros. En ella, encontraremos las siguientes opciones:

- **Archivo**: agrupa los comandos que permiten administrar los archivos con código VBA, como **Guardar Libro**, **Imprimir...**, **Importar archivo...** o **Exportar archivo...**.
- **Edición**: contiene los comandos que nos ayudarán a trabajar con la ventana donde se incluye el código VBA, como los clásicos **Copiar**, **Cortar**, **Pegar**, **Buscar** y **Reemplazar.** También incluye un conjunto de comandos que nos proporcionan información sobre el objeto con el cual estamos trabajando (**IntelliSense**) en la ventana de **Código**:

  - Lista de propiedades y métodos: muestra un menú con las propiedades y los métodos de un objeto después de que escribimos el punto.

  - Lista de constantes: presenta un menú emergente con todas las constantes disponibles para una propiedad o método.

  - Información rápida: para acceder a la información rápida de un elemento. Cuando escribimos una palabra reservada, seguida de un espacio o de un paréntesis, nos muestra información sobre la sintaxis de ese elemento y resalta el parámetro actual.

  - Información de parámetros: brinda información sobre los parámetros que se pueden utilizar en una función.

  - Palabra completa: usamos esta herramienta para completar una palabra clave que hemos comenzado a escribir. Es decir, si la cantidad de letras que hemos escrito son suficientes para definir una palabra clave única, **IntelliSence** completa el resto de la palabra.

  - Marcadores: muestra un menú que nos permite desplazarnos a través de los marcadores.

> PARA ACCEDER AL EDITOR DE VISUAL BASIC, PODEMOS PRESIONAR LAS TECLAS ALT + F11

- **Ver**: nos permite mostrar u ocultar características del entorno del Editor de VBA. Aquí encontramos las opciones para activar y desactivar las diferentes ventanas del editor como, por ejemplo: **Código**, **Inmediato**, **Locales**, **Inspección**, **Propiedades**, **Explorador de Proyectos**. Más adelante, explicaremos con más detalle cada una de ellas. Con la opción **Barra de herramientas**, podremos seleccionar las barras de herramientas que queremos mantener visibles.
- **Insertar**: desde aquí podemos insertar procedimientos, formularios, módulos y módulos de clase.
- **Formato**: este menú tiene varias opciones que nos permiten cambiar el tamaño y el aspecto de un grupo de controles de un formulario: botones, etiquetas, cuadro de texto, entre otros. Por ejemplo, el submenú **Igualar tamaño** nos permite hacer que un grupo de controles tengan el mismo ancho, alto o ambos.
- **Depuración**: después de escribir una macro necesitaremos probarla para ver si tiene errores. Este menú reúne las herramientas que nos permitirán ejecutar el código y alertarnos de algún error, como, por ejemplo, **Paso a paso por instrucciones**, **Paso a paso por procedimientos**.
- **Ejecutar**: contiene las herramientas para ejecutar, interrumpir y restablecer un procedimiento mientras estamos en modo de desarrollo, es decir, desde la ventana de código.
- **Herramientas**: desde este menú, podremos modificar las propiedades de los proyectos VBA, como así también las opciones generales del VBE, por ejemplo, controlar la apariencia del código (fuente utilizada, color, tamaño), controlar qué ventanas son acoplables, entre otras. Más adelante, describiremos estas herramientas.
- **Complementos**: permite el acceso al **Administrador de complementos**. Los **complementos** de Excel son archivos que tienen la extensión .XLA o .XLAM y guardan información de código creado en Visual Basic para Aplicaciones, como funciones y los

DEPURAR

La depuración de un programa es el proceso de corrección o la modificación del código para comprobar su funcionamiento. VBA incluye una amplia variedad de herramientas que nos van a ayudar en la tarea de la búsqueda de errores en el código, que veremos en los próximos capítulos.

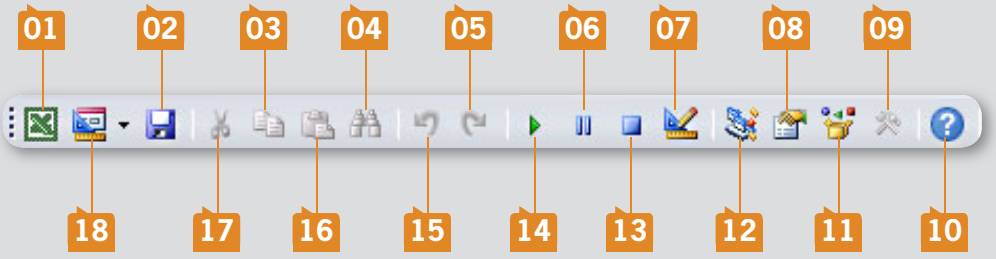
procedimientos, que utilizamos de manera frecuente. Los complementos se pueden usar en cualquier libro.

- **Ventana**: contiene los comandos que nos permiten organizar las ventanas del Editor Visual Basic.
- **Ayuda**: desde esta opción accedemos a la ayuda del Visual Basic.

La barra de herramientas Estándar

De manera predeterminada, debajo de la Barra de menú, se encuentra la barra de herramientas **Estándar**, que al igual que cualquier otra barra de herramientas contiene los botones con los comandos comúnmente más usados. En la siguiente **Guía visual**, explicaremos cada uno de sus elementos.

GV: BARRA DE HERRAMIENTAS ESTÁNDAR



01 **Ver Microsoft Excel:** permite ir de la ventana del Editor de Visual Basic a la hoja de trabajo de Microsoft Excel.

02 **Guardar:** guarda el libro, incluidos el proyecto y todos sus componentes.

03 **Copiar:** copia el texto u objeto seleccionado y lo coloca en el Portapapeles.

04 **Buscar:** abre el cuadro de diálogo **Buscar**, para rastrear un texto específico.

05 **Rehacer:** a la inversa del botón Deshacer, este botón rehace las últimas acciones de edición.

06 **Interrumpir:** detiene la ejecución de un procedimiento y cambia al modo de interrupción.

07 **Modo de Diseño:** permite activar o desactivar el modo **Diseño**.

08 **Ventana de Propiedades:** abre la ventana **Propiedades** donde visualizaremos las diferentes propiedades de cada uno de los objetos de VBA que seleccionemos.

09 **Cuadro de herramientas:** este botón está disponible cuando un UserForm está activo. Permite mostrar u ocultar el cuadro de herramientas que contiene todos los controles y objetos que se pueden insertar en un formulario.

10 **Ayuda de Microsoft Visual Basic para Aplicaciones:** brinda acceso a la ayuda de Microsoft Visual Basic.

11 **Examinador de objetos:** permite abrir la ventana del **Examinador de objetos** donde visualizaremos una lista con los objetos, sus propiedades, métodos y constantes.

12 **Explorador de proyectos:** abre una ventana que muestra los proyectos abiertos (archivos de Excel) y sus objetos (hojas de cálculo, formularios, módulos y módulos de clase).

13 **Restablecer:** restablece el proyecto interrumpido.

14 **Ejecutar macro:** permite ejecutar un procedimiento o un UserForm dependiendo de dónde se encuentre el cursor. Si las ventanas de **Código** o **UserForm** no están activas, entonces ejecutará una macro.

15 **Deshacer:** deshace la última acción de edición.

16 **Pegar:** inserta el contenido del Portapapeles en el lugar donde se encuentra el cursor.

17 **Cortar:** permite quitar el texto u objeto seleccionado y colocarlo en el Portapapeles.

18 **Insertar UserForm:** si hacemos clic directamente sobre el icono, se insertará un formulario (**UserForm**). Si desplegamos la flecha, veremos los comandos del menú **Insertar**: **UserForm**, **Módulo**, **Módulo de clase** y **Procedimiento**. Un formulario es una ventana que puede contener distintos controles, como botones de comandos, etiquetas, cuadros de texto, entre otros. En el **Capítulo 8**, trabajaremos con formularios

La barra de herramientas Edición

Una barra de herramienta que es de gran utilidad cuando estamos escribiendo un código es la barra de herramientas **Edición**. Para activarla, debemos ingresar al menú **Ver**, hacer un clic sobre la opción **Barra de herramientas** y, luego, seleccionar la opción **Edición**. Esta aparecerá como una barra flotante, pero si hacemos un clic sostenido con el mouse sobre el sector del título y la arrastramos, podremos ubicarla donde nos resulte más cómodo para trabajar.

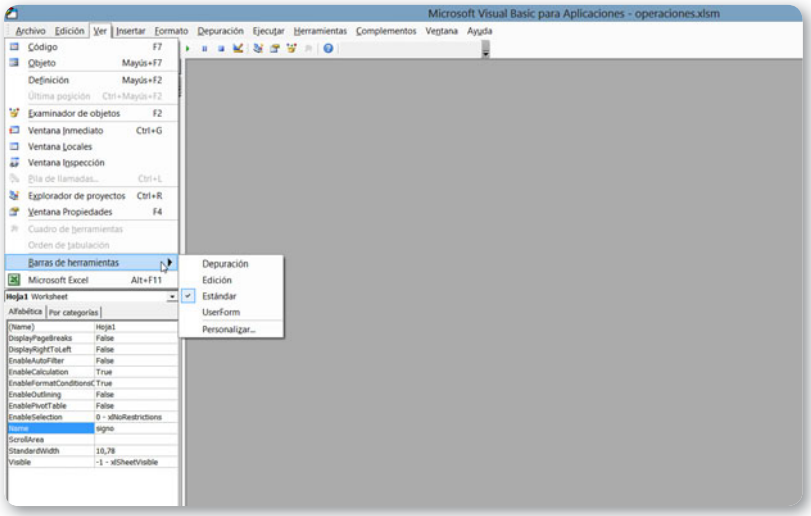


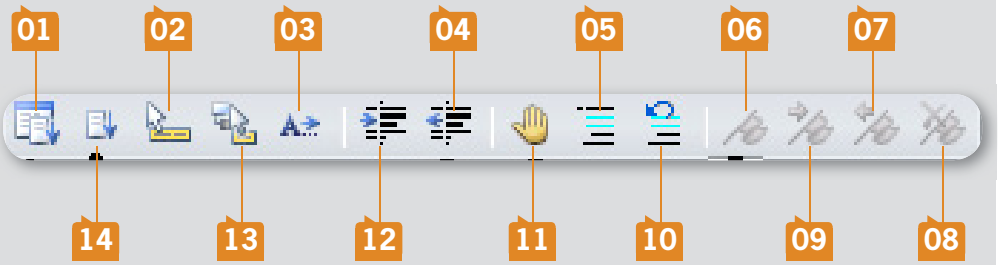
Figura 2. Desde el menú **Ver**, podemos activar y personalizar las diferentes barras de herramientas.

En la siguiente **Guía visual**, explicaremos para qué sirven las herramientas de esta barra.

TOOLTIPS

Los **tooltips** son aquellos mensajes emergentes que aparecen al pasar el puntero del mouse sobre determinados elementos y comandos de una aplicación. Son una herramienta que se encuentra presente en la mayoría de las interfaces gráficas, y se emplean para proporcionarle al usuario información adicional acerca de la función del elemento sobre el cual se encuentra el puntero. De esta manera, facilitan el trabajo de los usuarios y permiten un manejo más intuitivo de los programas.

GV: BARRA DE HERRAMIENTAS DE EDICIÓN



01 **Lista de propiedades y métodos:** muestra una lista de las propiedades y los métodos que podemos aplicar al objeto seleccionado.

02 **Información rápida:** muestra un cuadro que proporciona información de sintaxis del elemento que hemos seleccionado.

03 **Palabra completa:** termina de escribir la palabra clave que hemos comenzado a escribir en la ventana **Código**, una vez que cuenta con los caracteres suficientes para identificarla.

04 **Sangría izquierda:** desplaza a la izquierda el texto seleccionado. Equivale a presionar la combinación de teclas **SHIFT** + **TAB**.

05 **Bloque de comentarios:** convierte un texto seleccionado en un comentario. Los comentarios son líneas dentro de nuestro código que no serán tomadas en cuenta al momento de realizar la ejecución del procedimiento.

06 **Alternar marcador:** lo empleamos para activar o desactivar marcadores, que nos permiten desplazarnos de forma sencilla a través del código.

07 **Alternar anterior:** se desplaza al marcador anterior.

08 **Borrar todos los marcadores:** elimina todos los marcadores que se encuentren activados. Cuando cerramos el editor, los marcadores se borran.

09 **Marcador siguiente:** se desplaza al marcador siguiente.

10 **Bloque sin comentarios:** transforma el texto seleccionado en código.

11 **Alternar punto de interrupción:** coloca un punto de interrupción en la línea de código seleccionada. Lo utilizamos para depurar código.

12 **Sangría derecha:** desplaza hacia la derecha el texto seleccionado. Equivale a pulsar la tecla **TAB**.

13 **Información de parámetros:** muestra información sobre los parámetros de la instrucción que hemos seleccionado.

14 **Lista de constantes:** despliega las constantes del sistema que se aplican al argumento actual. Una **constante** es un objeto de datos que tiene un valor fijo que no puede ser alterado. VBA posee un gran número de constantes intrínsecas que podemos utilizar para verificar aspectos tales como los tipos de datos, o podemos utilizar como argumentos fijos en funciones y expresiones.

Debajo de las diferentes barras, veremos que la interfaz del VBE se divide de manera predeterminada en tres sectores: la ventana **Explorador de Proyectos**, la ventana **Código** y la ventana **Propiedades**. Explicaremos cada una de ellas en detalle.

El Explorador de Proyectos

El **Explorador de Proyectos** se ubica en el sector superior izquierdo de la ventana de VBA. Si, por alguna razón, no estuviera visible, lo podemos activar de las maneras que detallamos a continuación:

- Seleccionando la opción **Explorador de Proyectos** del menú **Ver**.
- Presionando la combinación de teclas **CTRL + R**.
- Haciendo un clic sobre el botón **Explorador de Proyectos** que se encuentra en la **Barra de herramientas Estándar**.

BARRAS DE HERRAMIENTAS

Al igual que muchas de las aplicaciones para Windows, el Editor de Visual Basic tiene diferentes barras de herramientas disponibles. Estas pueden ser personalizadas, y, de esta manera, es posible crear barras de herramientas propias de acuerdo con nuestras necesidades.

Para comprender de manera clara la utilidad de esta ventana, primero es necesario definir qué es un proyecto.

Un **proyecto** es un libro de Excel que contiene un conjunto de objetos: hojas de trabajo, módulos y formularios. Entonces, a través de esta ventana, es posible gestionar los proyectos y movernos entre sus distintos componentes, como así también entre los diferentes proyectos que tengamos abiertos.

En la ventana vemos, en forma de estructura jerárquica, los proyectos que tenemos abiertos y los objetos que estos contienen. El nivel superior de la estructura es el proyecto actual, que, de manera predeterminada, recibe el nombre **VBAProyect(Libro1)**.

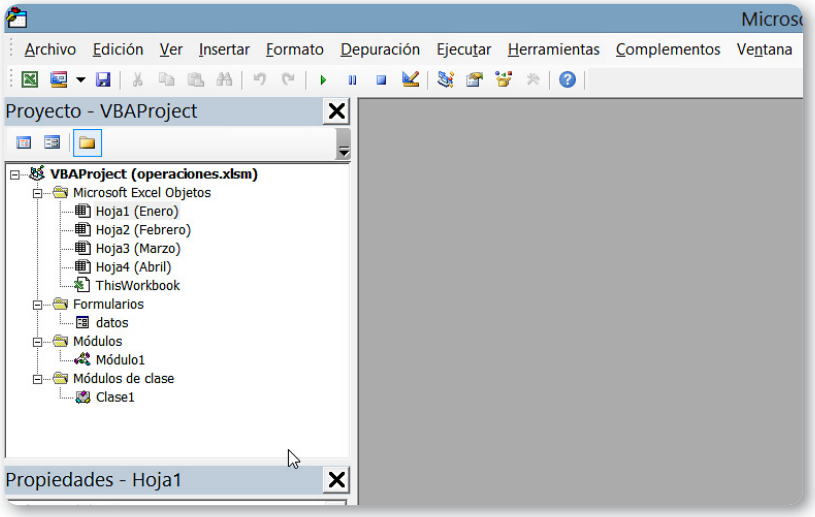


Figura 3. Desde el **Explorador de Proyectos**, es posible navegar por los objetos de un libro de Excel, que contienen o pueden contener código VBA.

En el nivel inmediatamente inferior al proyecto, aparece la carpeta:

- **Microsoft Excel Objetos**: esta carpeta reúne los objetos que forman parte del libro, como las hojas y las hojas de gráfico. El elemento llamado **ThisWorkbook** representa el libro completo. En cada uno de estos objetos, podemos escribir nuestro propio código (procedimientos). El código que incluyamos en el objeto **ThisWorkbook** afectará al resto de los objetos.

A medida que vamos insertado formularios, módulos o módulos de clase, se crearán las siguientes carpetas:

- **Formularios**: agrupa los objetos **Formularios**, llamados también **UserForms** o cuadros de diálogos personalizados. Los formularios contienen una serie de controles que nos van a permitir introducir, modificar o visualizar datos.
- **Módulos**: también se los denomina módulos estándar; en ellos escribiremos procedimientos. Un **procedimiento** es un conjunto de instrucciones VBA que sirven para realizar una tarea específica. Por ejemplo, podemos generar un procedimiento que seleccione un conjunto de datos, para luego crear con ellos un gráfico de barras o de otra clase. Un módulo puede contener varios procedimientos. A su vez, es posible tener tantos módulos como necesitemos dentro de un mismo libro. Estos pueden ser exportados o importados entre distintos proyectos. Se guardan bajo un nombre y poseen la extensión .BAS (que identifica a los archivos básicos).
- **Módulos de clase**: son módulos especiales que nos permiten crear nuestros propios objetos, dotándolos de diversas propiedades. Tienen la extensión .CLS (archivos de clase) y, al igual que los módulos estándares, pueden ser exportados o importados.

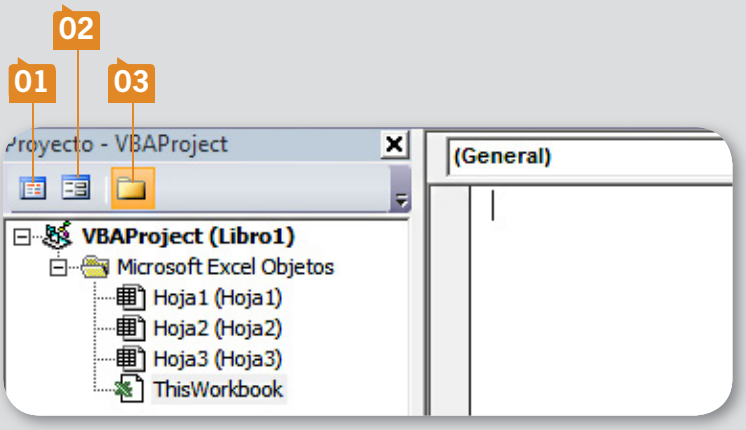
> UN PROYECTO ES UN LIBRO DE EXCEL QUE CONTIENE HOJAS DE CÁLCULO, MÓDULOS Y FORMULARIOS

En la parte superior de la ventana **Explorador de Proyectos**, hay una barra de herramientas que contiene tres botones que detallaremos en la **Guía visual** que presentamos a continuación.

SINTAXIS

Se denominan **reglas de sintaxis del lenguaje** a las reglas de gramática que debemos seguir en un lenguaje de programación. Las reglas de sintaxis determinan si la secuencia de caracteres que forman un programa puede ser traducida por el compilador a código de máquina. Cada lenguaje de programación tiene sus propias reglas de sintaxis.

GV: HERRAMIENTAS DEL EXPLORADOR DE PROYECTOS



01 **Ver código:** muestra la ventana de código para el módulo seleccionado.

02 **Ver objeto:** muestra la hoja o formulario que hemos seleccionado en la carpeta de objetos.

03 **Alternar carpetas:** permite ver todos los objetos en una sola carpeta, independientemente del tipo de objeto.

La ventana Código

La ventana **Código**, también llamada Editor de código, es la que emplearemos para escribir, mostrar y editar todo el código Visual Basic que va a gestionar una macro.

Cada elemento de un proyecto tiene asociado una ventana **Código**. Podemos tener abiertas tantas ventanas como necesitemos, pudiendo compartir código entre ellas mediante las acciones de cortar, copiar y pegar. Si no está visible, la activamos de las siguientes maneras:

- Haciendo un doble clic con el mouse sobre algún objeto de la ventana **Explorador de Proyectos**.

- Seleccionando del menú **Ver** la opción **Código**.
- Presionando la tecla **F7**.

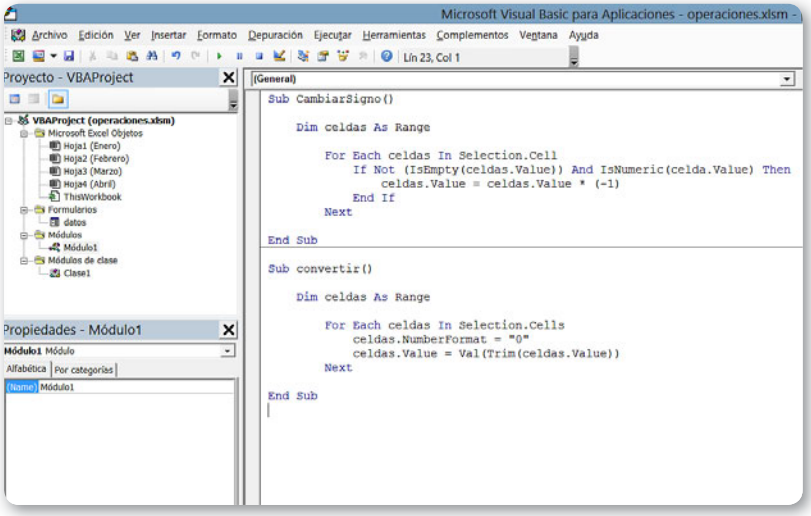


Figura 4. La ventana **Código** contiene diferentes elementos que nos facilitan localizar procedimientos y editar código VBA.

En función del objeto que deseemos codificar, esta ventana tendrá algunas particularidades determinadas. Si es un módulo estándar de código, tendrá solo funciones y procedimientos. Si es un módulo de código de un formulario, contendrá, además de los procedimientos normales, los métodos para los eventos del objeto **Formulario** y **Controles**. En cambio, si es un módulo de clase, tendrá procedimientos de propiedades, la definición de la clase y métodos, entre otros.

Debajo de la barra de título de la ventana, veremos dos listas desplegables (poseen dos flechas a su derecha) que nos van a permitir movernos de manera fácil y rápida dentro del código VBA.

CONTRAER O EXPANDIR

Los objetos de Visual Basic para Aplicaciones se guardan en carpetas que representan su función. Al igual que en el Explorador de Windows, podremos expandir o contraer las carpetas del Explorador de Proyectos haciendo simplemente un clic en el signo + (más) o – (menos).

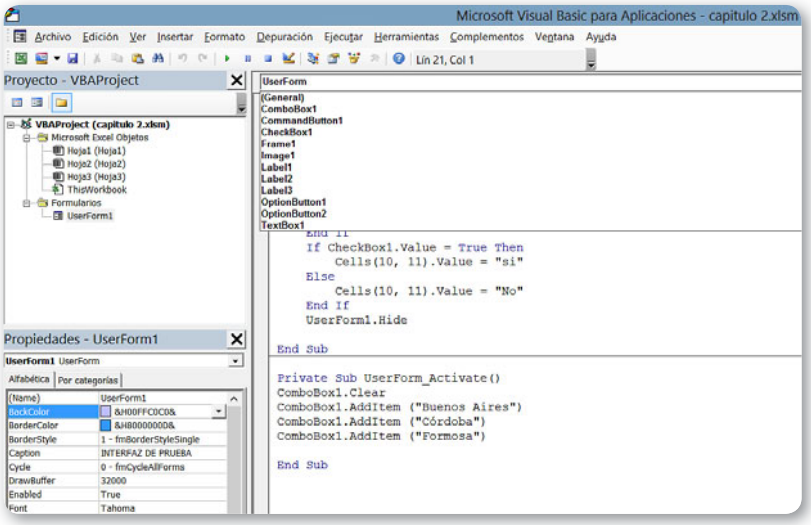


Figura 5. La ventana **Código** muestra la lista de objetos del formulario que hemos seleccionado.

- A la izquierda, tenemos el cuadro **Objetos**, que muestra el nombre del objeto que se encuentra seleccionado. Si el objeto es un módulo, solo mostrará la opción general. En cambio, si el objeto es un formulario, presentará una lista de todos los objetos asociados con él, como podemos observar en la **Figura 5**.
- A la derecha, se encuentra el cuadro **Procedimiento/Evento**. Si hemos seleccionado un módulo estándar de código, tendrá solo funciones y procedimientos. En cambio, si es un módulo de clase, tendrá procedimientos de propiedades, la definición de la clase y métodos, entre otros. Si es un módulo de código de un formulario, contendrá además de los procedimientos normales, los métodos para los eventos del objeto **Formulario** y **Controles**, como muestra la **Figura 6**.

DRAG AND DROP

Si tenemos abiertas varias ventanas **Código**, podemos copiar código entre ellas, utilizando el método **drag and drop**. Seleccionamos el código y lo arrastramos manteniendo presionada la tecla **CTRL** hasta el otro módulo. Si no mantenemos presionada esta tecla cuando arrastramos, en lugar de copiar el código, lo estaremos moviendo.

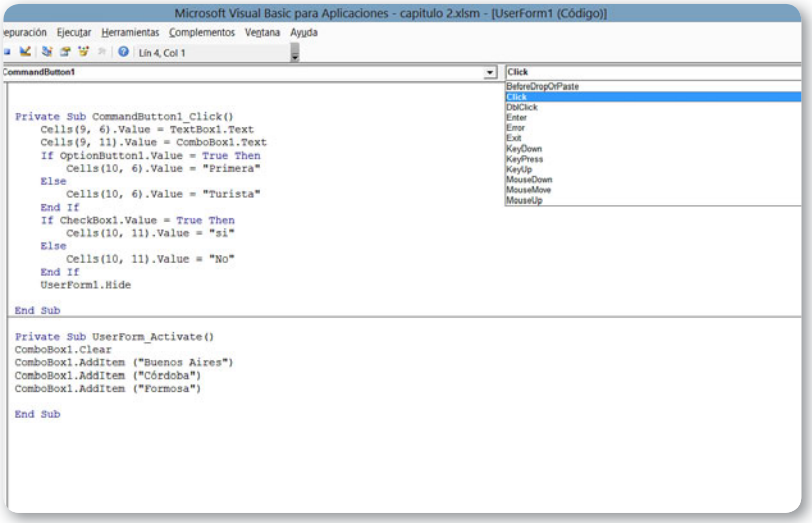


Figura 6. El cuadro **Procedimiento/Evento** presenta una relación de los eventos reconocidos por VBA para un formulario o control mostrado en el cuadro **Objeto**.

En la esquina inferior de la ventana **Código**, hay dos botones que nos permiten establecer cómo veremos los procedimientos:

- El botón **Ver módulo completo**, situado a la derecha, muestra todos los procedimientos incluidos en el módulo seleccionado, separados por una línea.
- El botón **Ver procedimiento**, ubicado a la izquierda, permite filtrar los procedimientos, mostrando únicamente el código del procedimiento sobre el que se encuentra el cursor.

> PODEMOS VER TODOS LOS PROCEDIMEINTOS DE UN MÓDULO O FILTRARLOS

DESPLAZARSE POR LOS PROCEDIMIENTOS

Todos los procedimientos de un módulo se muestran en una sola lista, por la cual nos podemos desplazar. Cuando seleccionamos un procedimiento utilizando los cuadros de lista de la ventana **Código**, el cursor se moverá a la primera línea del código en el procedimiento que hemos seleccionado.

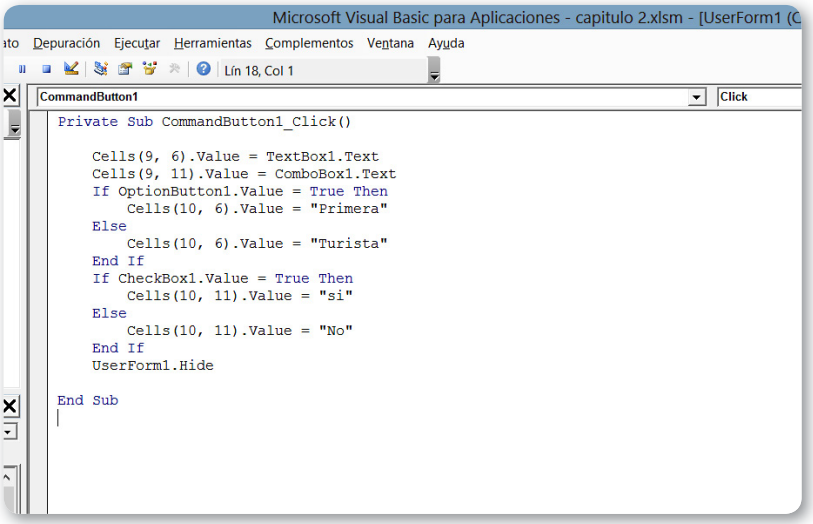


Figura 7. Todo el código del procedimiento se muestra igual que si se mirara la versión de un texto en un procesador de texto.

LA BARRA DE DIVISIÓN PERMITE SEPARAR LA VENTANA CÓDIGO EN DOS PANELES

Inmediatamente por encima de la barra de desplazamiento vertical, encontramos la **barra de división**. Si arrastramos esta barra hacia abajo, dividimos la ventana **Código**, en sentido vertical, en dos paneles, uno debajo del otro. De este modo, es posible desplazarnos de manera independiente por cada uno de los paneles a medida que lo necesitemos.

Los cuadros **Objeto** y **Procedimiento/Evento** mostrarán las opciones que corresponden al panel que se encuentre activo. Para cerrar el panel, simplemente debemos hacer doble clic sobre la barra de división.

BREAKPOINTS

La ventana **Código** presenta en la parte izquierda una barra indicadora. En esta, podremos situar marcas para señalar distintos puntos de interrupción, también denominados breakpoints, dentro de un determinado procedimiento o puntos de interrupción en diferentes procedimientos, que nos permiten examinar en detalle una línea de código específica.

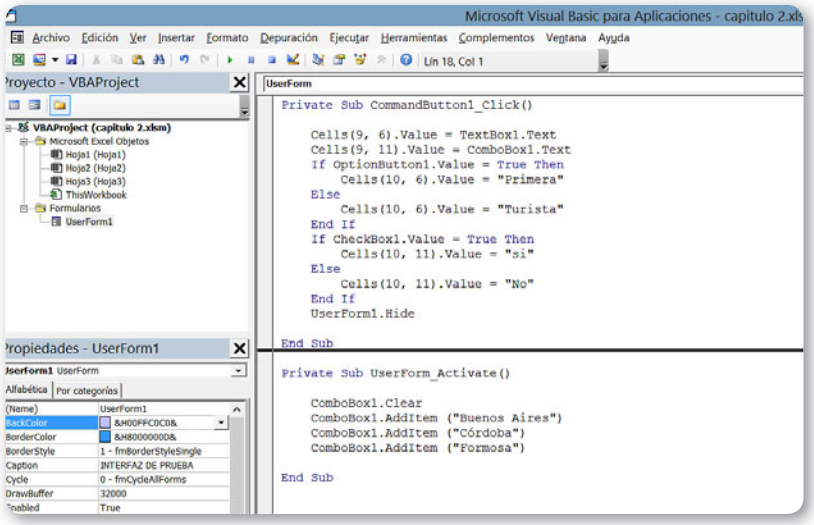


Figura 8. Si arrastramos la barra de división hasta la parte superior de la ventana, se cierra el panel.

La ventana Propiedades

La ventana **Propiedades** se ubica en el sector inferior izquierdo de la ventana de VBA. Muestra y permite cambiar las diferentes propiedades del objeto que hemos seleccionado en la ventana del **Explorador de proyectos**, mientras estamos en modo de diseño, es decir, cuando no estamos ejecutando un procedimiento o formulario.

Si esta ventana no está visible, podemos activarla mediante los siguientes procedimientos:

- Seleccionando del menú **Ver** la opción **Ventana Propiedades**.
- Presionando la tecla **F4**.

TRADUCTOR DE CÓDIGO

El traductor es un programa que convierte el código fuente a un código objeto y luego a un código ejecutable. Puede ser un **compilador** o un **intérprete**. El primero de ellos se encarga de revisar la sintaxis y, si no se detectan errores, hace la traducción del código fuente a un lenguaje de bajo nivel, mientras que el intérprete hace la revisión línea por línea.

- Haciendo clic sobre el botón **Ventana de Propiedades** que se encuentra en la barra de herramientas **Estándar**.

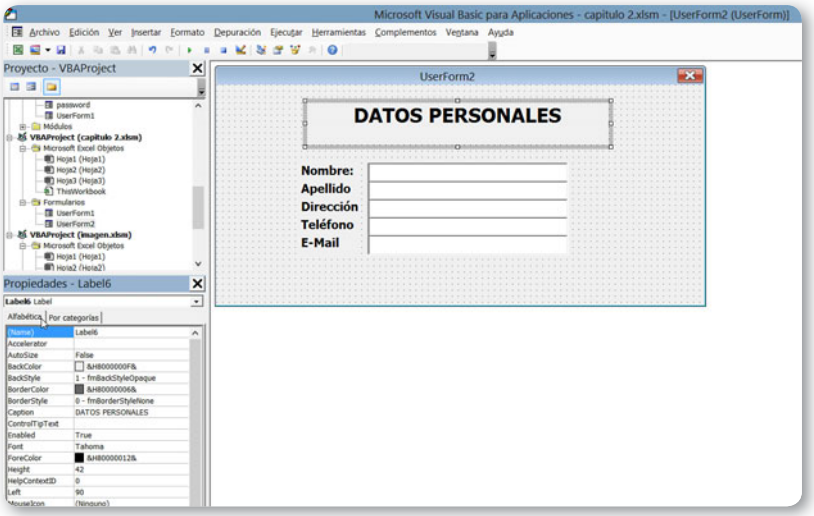


Figura 9. La ventana **Propiedades** muestra la configuración del objeto seleccionado.

Debajo de la barra de título de la ventana, visualizamos el nombre y el tipo del objeto que se encuentra actualmente seleccionado. En la parte inferior, vemos la lista de propiedades del objeto. Si seleccionamos un libro, una hoja o un formulario, tendremos una gran cantidad de propiedades. En cambio, cuando seleccionamos un módulo estándar, la única propiedad que veremos será el nombre del módulo (**Name**). En próximos capítulos, veremos cómo podemos cambiar las diferentes propiedades de los objetos.

INFORMACIÓN SOBRE LAS PROPIEDADES

Cada objeto posee propiedades específicas. Si queremos obtener información acerca de una propiedad en particular, podemos recurrir a la ayuda de Visual Basic para Aplicaciones. Para esto, debemos seleccionar la propiedad en cuestión y, luego, presionar la tecla **F1**. Inmediatamente se abrirá la ventana de ayuda de VBA con toda la información disponible sobre la propiedad.

Es posible visualizar las propiedades del objeto ordenadas en forma alfabética o agrupadas por categorías haciendo clic en las solapas correspondientes de esta ventana.

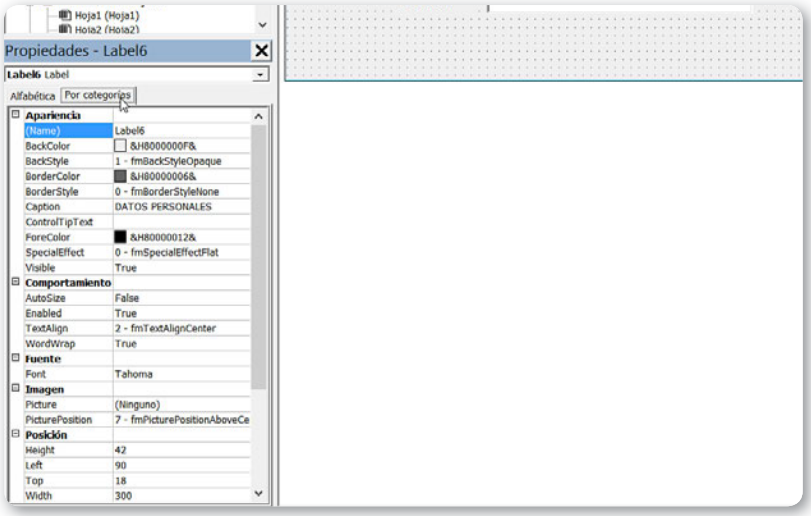


Figura 10. Luego de seleccionar la solapa **`Por categoría`**, vemos las propiedades del objeto **`Label`** agrupadas según este concepto.

Otras ventanas

Además de las tres ventanas que describimos antes, existen otras que nos ayudarán a escribir y probar el código, como la ventana **Inmediato**, la ventana **Locales** y la ventana **Inspección**. A continuación, veremos cada una de ellas en detalle.

La ventana Inmediato

La ventana **Inmediato**, también llamada ventana de depuración, permite probar una instrucción, un comando o una función del lenguaje VBA cuando estamos en modo de diseño. Podemos activarla de las siguientes maneras:

LA VENTANA INMEDIATO PERMITE PROBAR UNA INSTRUCCIÓN EN MODO DE DISEÑO

- Seleccionando del menú **Ver** la opción **Ventana Inmediato**.
- Presionando la combinación de teclas **CTRL + G**.

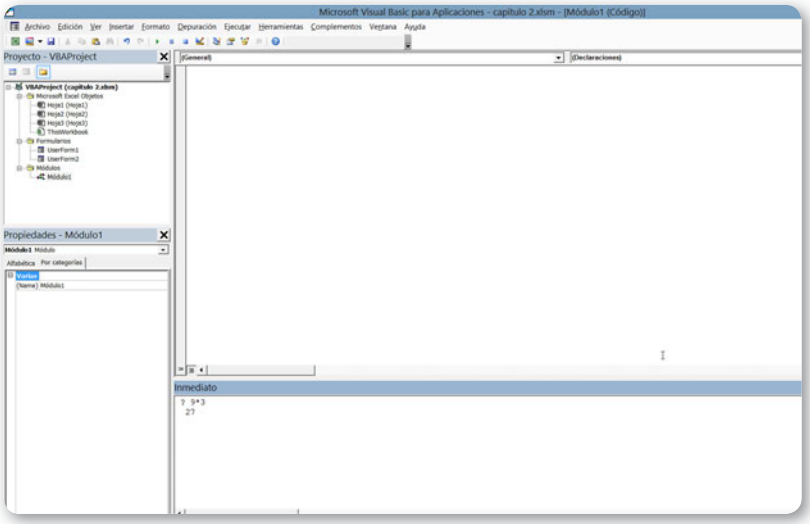


Figura 11. La ventana **Inmediato** es útil para ejecutar las instrucciones VBA de manera directa.

LAS INSTRUCCIONES DE LA VENTANA INMEDIATO SE EJECUTARÁN AL PULSAR ENTER

Esta herramienta muestra los resultados de las instrucciones que introducimos en esta ventana, sin tener que escribirlas en un procedimiento.

De esta manera, es posible comprobar los efectos de ciertos valores en un procedimiento o función de forma aislada.

Las instrucciones que escribamos en esta ventana se ejecutarán luego de presionar la tecla **ENTER**. Podemos volver a ejecutar cualquier instrucción que se encuentre en esta ventana, ubicando el cursor donde está la línea y presionando otra vez la tecla.

Excel recuerda todas las instrucciones que escribamos en la ventana **Inmediato**, incluso después de haberla cerrado. Su contenido se eliminará una vez que salgamos del programa.

La ventana Locales

La ventana **Locales** se utiliza para comprobar el valor de las variables y los objetos locales del módulo en el que nos encontremos durante la ejecución del código. Para activarla, vamos al menú **Ver** y seleccionamos la opción **Ventana Locales**.

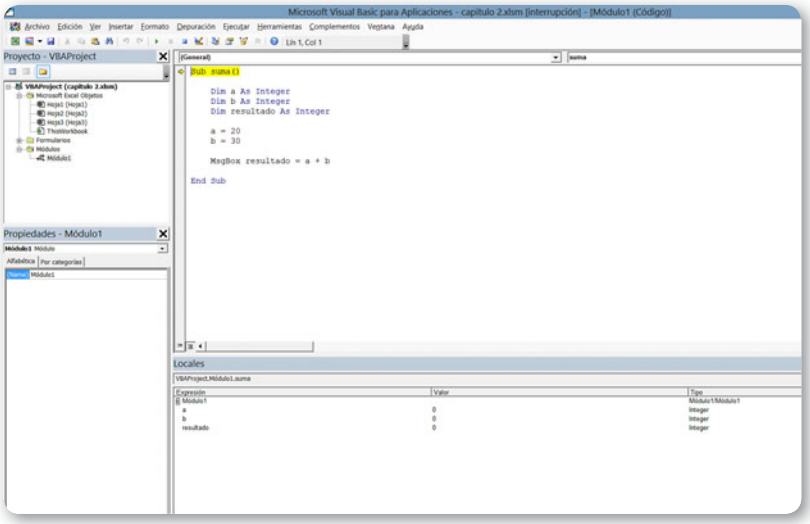


Figura 12. La ventana **Locales** simplemente muestra los nombres y los valores de las variables.

La ventana Inspección

La ventana **Inspección** permite agregar objetos para ver el valor actual de una variable o expresión, cuando estamos en modo de interrupción. Para activarla, debemos seleccionar la opción dentro del menú **Ver**.

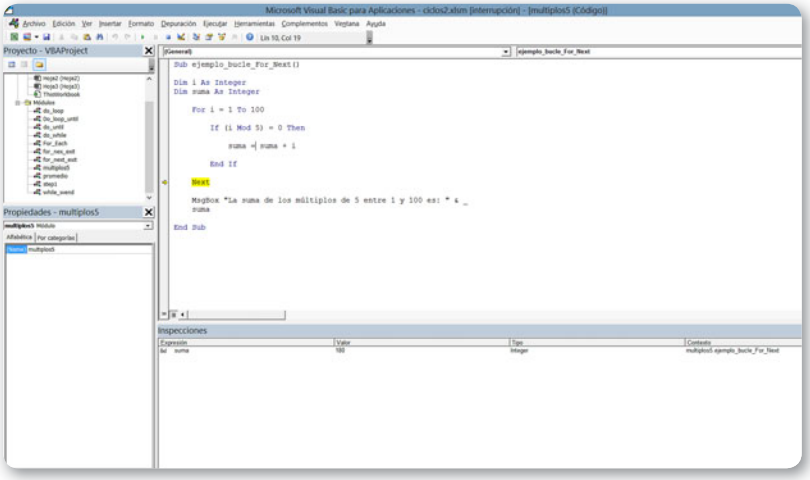


Figura 13. La ventana **Inspección** permite agregar objetos para tenerlos **vigilados**.

El Examinador de objetos

El **Examinador de objetos** nos provee de los medios básicos para encontrar los objetos, sus propiedades y métodos asociados, que utilizamos en los proyectos VBA. Para activarlo:

- Seleccionamos del menú **Ver** la opción **Examinador de objetos**.
- Presionamos la tecla **F2**.
- Hacemos clic en el botón **Examinador de objetos** de la barra de herramientas **Estándar**.

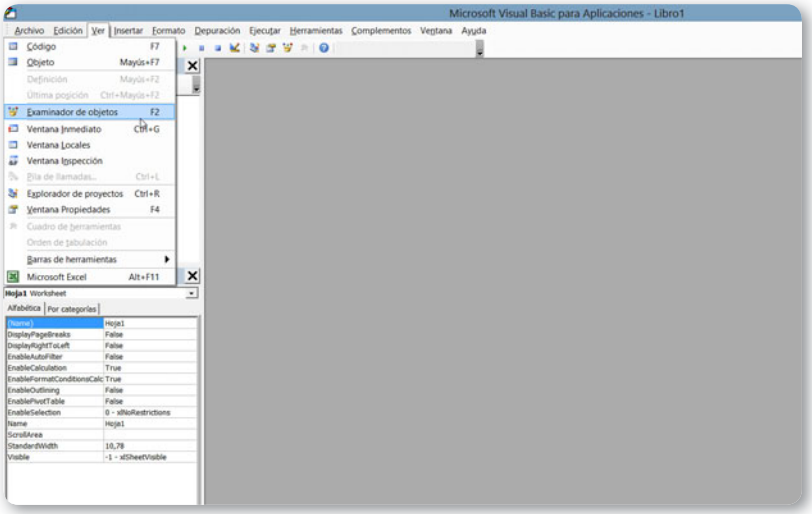


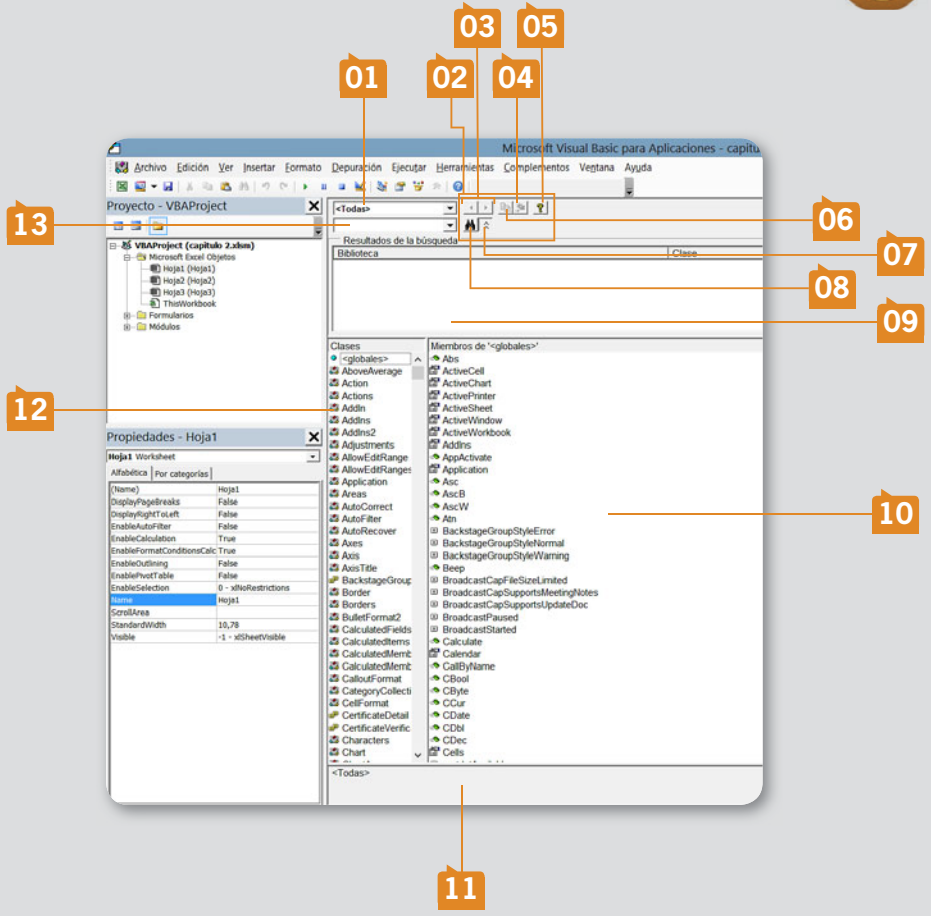
Figura 14. Disponemos de tres posibilidades para acceder al **`Examinador de objetos`**.

Una vez que abrimos la ventana, veremos una jerarquía de los objetos que aparecen en VBA, junto con sus métodos y propiedades. En la siguiente **Guía visual**, explicaremos en detalle cada una de sus partes.

TRABAJAR EN EQUIPO

Para compartir un archivo de Excel 2013 usando el servicio de **SkyDrive**, iniciamos sesión en Office con una cuenta de Microsoft haciendo un clic en el botón **Iniciar sesión**, de la esquina superior derecha de la ventana. Luego, con el libro abierto, hacemos clic en **Archivo/Guardar Como** y seleccionamos **SkyDrive**.

GV: EXAMINADOR DE OBJETOS



01 **Cuadro Proyecto/Biblioteca:** despliega una lista con los nombres de todas las bibliotecas y los proyectos que se encuentran disponibles en el proyecto activo.

02 **Barra de herramientas:** contiene los accesos rápidos para trabajar con el **Examinador de objetos**.

03 **El Botón Volver** nos sitúa en el último lugar donde estuvimos con el examinador y el **Botón Avanzar** nos sitúa en el último sitio donde estuvo el examinador antes de presionar **Volver.**

04 **Botón Ver definición:** si el miembro está definido en nuestro código, al presionar este botón se desplazará a esa parte del código.

05 **Botón Ayuda:** nos proporciona una ayuda en línea para el elemento que tengamos seleccionado.

06 **Botón Copiar al Portapeles:** copia la información que se muestra actualmente al portapapeles. Luego podemos usar la opción **Pegar** para incluir esta información en nuestro código.

07 **Botón Mostrar/Ocultar:** abre u oculta el cuadro **Resultados de búsqueda**.

08 **Botón Búsqueda:** permite buscar en las bibliotecas que hayamos seleccionado en el cuadro **Proyecto/Biblioteca** el texto que hayamos introducido en el cuadro de texto búsqueda.

09 **Resultados de la búsqueda:** muestra una lista con la biblioteca, la clase y el miembro que corresponde al texto que hemos escrito en el cuadro de texto de búsqueda

10 **Miembros:** proporciona una lista de los métodos y propiedades disponibles de la clase que seleccionamos en el cuadro **Clases**. Por defecto, los miembros aparecen ordenados en forma alfabética.

11 **Panel Detalles:** muestra una definición del miembro seleccionado.

12 **Clases:** muestra todas las clases de la biblioteca que hemos seleccionado del cuadro Proyecto/Biblioteca. Si seleccionamos, por ejemplo, la clase Excel, nos mostrará todos los objetos de Excel.

13 **Cuadro de texto de búsqueda:** permite encontrar rápidamente la información en una biblioteca en particular. Para ello, seleccionamos una biblioteca, escribimos en el cuadro de texto de búsqueda el texto por buscar y, luego, hacemos clic en el botón **Buscar**. Los resultados de la búsqueda se mostrarán en la sección **Resultado de la búsqueda**.

BIBLIOTECA

Una biblioteca consiste en un archivo que contiene información acerca de los objetos de una aplicación. Si en la ventana del **Examinador de objetos** seleccionamos la opción **<todas>**, como podemos esperar, se mostrarán los objetos de todas las bibliotecas instaladas en la computadora. Si, en cambio, por ejemplo, únicamente seleccionamos la biblioteca **Excel** o **VBA**, listará solo los nombres de los objetos que son exclusivos de Excel o Visual Basic para Aplicaciones.

Personalizar el Editor de VBA

Como en la mayoría de las aplicaciones, es posible personalizar la configuración del Editor de Visual Basic, ya sea modificando la disposición de las ventanas como la edición de código.

Para poder realizar estas modificaciones, debemos acceder al menú **Herramientas** y luego, selecionar **Opciones**. A continuación, conoceremos algunas de las características principales que contiene este cuadro de diálogo.

> PARA PERSONALIZAR EL VBE, DEBEMOS ACCEDER A HERRAMIENTAS/ OPCIONES

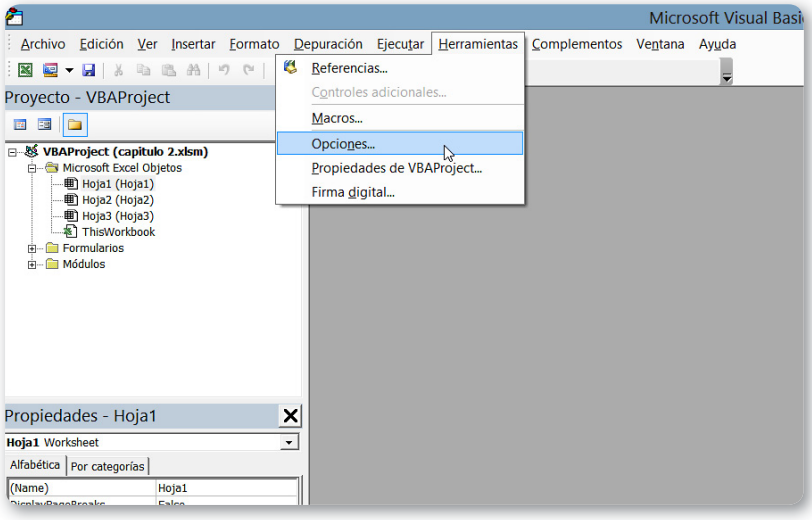


Figura 15. Otra forma de acceder a las opciones de configuración del Editor VBA es pulsar la combinación de teclas **ALT + H + N**.

ACCEDER AL MENÚ

Otra forma de acceder a los menús es mediante la tecla **ALT**. Al pulsarla, el menú **Archivo** aparece destacado y, con las flechas de dirección, podemos desplazarnos a través de los diferentes títulos. Para abrir un menú, presionamos la tecla **ENTER** y, para seleccionar una opción, pulsamos la tecla de la letra que se muestra subrayada en el nombre en cuestión (tecla de acceso).

Formato del editor: configurar la tipografía

Cuando escribimos código VBA en la ventana **Código**, de manera predeterminada, las palabras clave, las funciones y las instrucciones VBA aparecen en color azul; los objetos, los métodos y las propiedades se muestran en negro, y los comentarios se presentan en color verde. Es posible modificar estos colores de acuerdo con nuestra preferencia, mediante la pestaña **Formato del editor**.

- En el sector **Colores de código**, vemos una lista con todos los elementos de código a los que se les puede cambiar el color. Debajo hay listas desplegables para elegir los colores de primer plano, de fondo, del texto utilizado para los indicadores del margen y los marcadores.
- El cuadro **Fuente** despliega la lista con los tipos de fuentes disponibles en nuestra computadora. **Courier New** es la fuente que aparece en forma predeterminada.
- El cuadro **Tamaño** permite seleccionar o escribir un cuerpo de fuente.
- El cuadro **Ejemplo** presenta una vista previa de la fuente, el tamaño y los colores que fuimos definiendo.

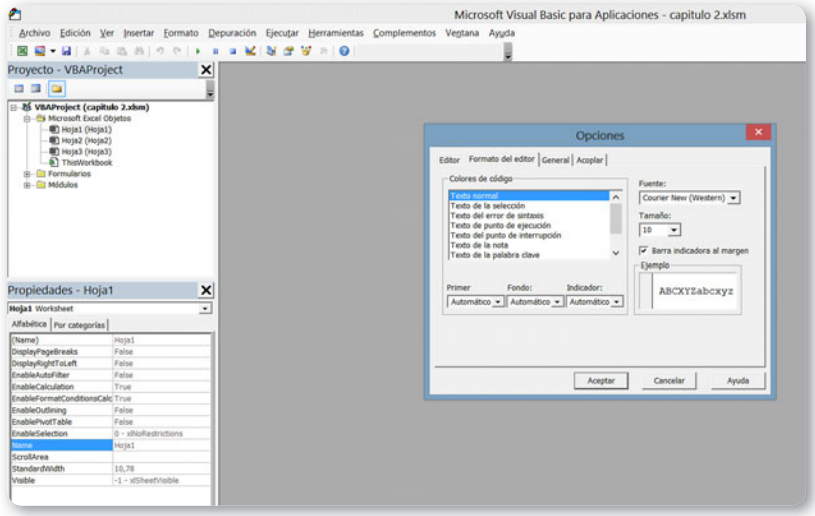


Figura 16. Desde la pestaña **Formato del editor**, es posible cambiar los colores del código, elegir la fuente y el tamaño de la letra.

La ficha Editor: configurar la introducción de código

Desde la pestaña **Editor**, configuramos el entorno de edición. Está dividida en dos secciones: **Opciones del código** y **Opciones de la ventana**.

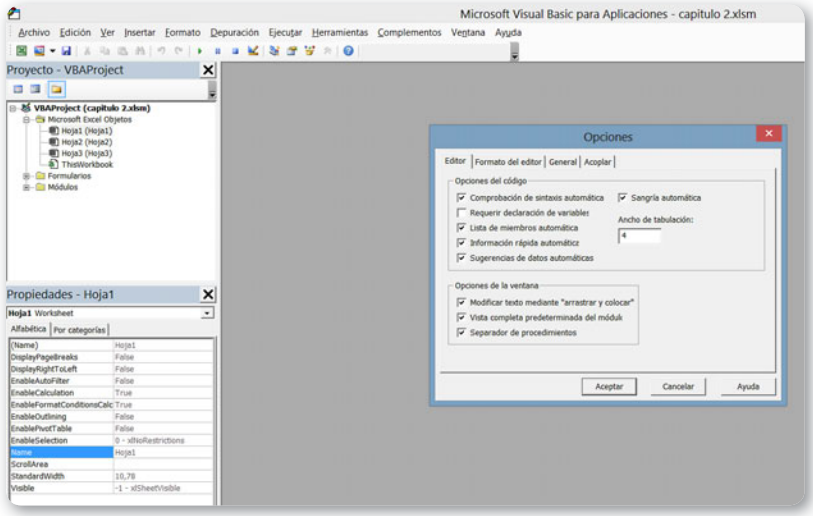


Figura 17. Para configurar las opciones de la pestaña **Editor**, debemos seleccionar las casillas de verificación y luego pulsar **Aceptar**.

En la primera sección encontramos las siguientes opciones:

- **Comprobación de sintaxis automática**: comprueba cada línea de código después de su introducción e informa la existencia de errores.
- **Requerir la declaración de variables**: agrega la instrucción **Option Explicit** al crear un nuevo módulo. Esta sentencia fuerza la declaración de variables. Detallaremos más sobre este tema en próximos capítulos.

UBICACIÓN DE LAS VENTANAS

Cuando seleccionamos una ventana desde el menú **Ver**, esta aparecerá en el lugar donde la ubicamos la última vez. Es decir, si colocamos la ventana **Inmediato** en la parte superior de la interfaz, la próxima vez que abramos la ventana de Microsoft Visual Basic para Aplicaciones, esta se encontrará en el mismo lugar.

- **Lista de miembros automática**: muestra una lista de miembros, constantes o valores disponibles para un parámetro de función, procedimiento, instrucción, entre otras.
- **Información rápida automática**: proporciona información sobre las funciones y sus parámetros a medida que se escriben.
- **Sugerencias de datos automáticas**: brinda información sobre variables y expresiones. Solo está disponible en modo de interrupción.
- **Sangría automática**: después de haber incluido una tabulación en una línea, permite que las líneas siguientes comiencen en esa posición.
- **Ancho de tabulación**: para establecer el valor de ancho del tabulador. El valor predeterminado es de 4 espacios, pero se puede definir con un intervalo de 1 a 32 espacios

En la segunda sección, disponemos de las siguientes opciones:

- **Modificar texto mediante "arrastrar y colocar"**: permite arrastrar y colocar código en las ventanas **Inmediato** e **Inspección**.
- **Vista completa predeterminada del módulo**: configura la ventana **Código** para mostrar los procedimientos como una lista única que se puede desplazar o cada procedimiento por separado.
- **Separador de procedimientos**: permite visualizar u ocultar las líneas de separación que aparecen al final de cada procedimiento en la ventana **Código**.

Acople de ventanas

Como vimos anteriormente, el Editor de Visual Basic contiene diferentes ventanas que utilizamos para el desarrollo de las macros. Podemos organizarlas, cambiarles el tamaño, moverlas, de la misma manera que lo hacemos en las aplicaciones del entorno Windows.

MODO DE INTERRUPCIÓN

El modo de interrupción es una parada momentánea mientras ejecutamos un programa desde el entorno de VBA. Cuando estamos en este modo, podemos ver paso a paso cómo se comporta una aplicación, y corregir los errores del código o cambiar su comportamiento de acuerdo con nuestra necesidad.

También es posible acoplar una ventana a un costado de la pantalla del editor, mediante la función arrastrar y soltar. Pero, antes de realizar este procedimiento, en la pestaña **Acoplar**, debemos especificar cuáles son las ventanas que deseamos acoplar.

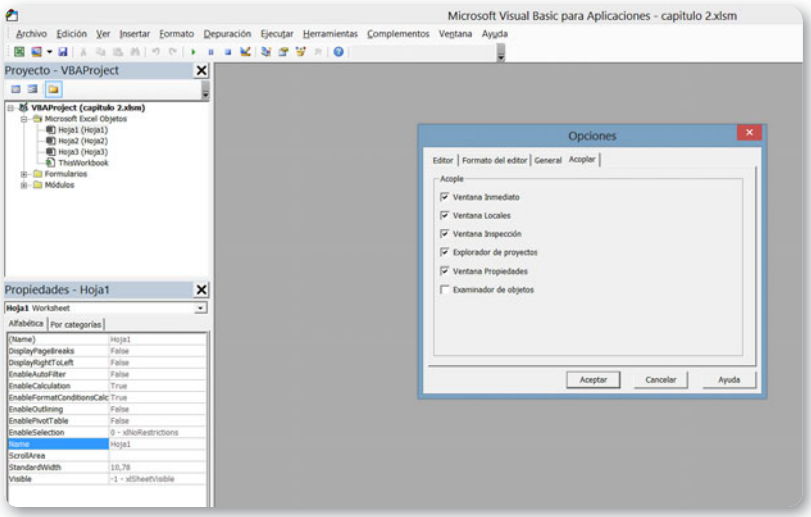


Figura 18. En la pestaña **Acoplar**, seleccionamos las casillas de las ventanas que queremos acoplar y, luego, podemos arrastrarlas.

La ficha General: gestión de errores

La ficha **General** nos permite establecer aquellas opciones que se aplicarán sobre el entorno de desarrollo. Entre las más importantes se encuentran las que detallamos a continuación:

- **Opciones de la cuadrícula**: los formularios, en modo de diseño, presentan una cuadrícula que nos sirve para posicionar los controles en forma alineada. Desde aquí, podemos mostrar u ocultar la cuadrícula, y configurar el ancho y alto. La opción **Forzar controles a cuadrícula** permite alinear los objetos a ella.
- **Interceptación de errores**: a pesar del cuidado que tengamos cuando estemos escribiendo el código de una macro, en algunas situaciones,

LA FICHA GENERAL NOS PERMITE ESTABLECER LAS OPCIONES PARA TODO EL ENTORNO

podemos cometer una serie de errores. Muchos de estos, a medida que vamos escribiendo las sentencias, son detectados en forma inmediata por el editor de VBA; en cambio, otros son descubiertos recién cuando ejecutamos la macro. Cuando se produce un error al ejecutar una macro, aparecerá un mensaje, y el proceso se detendrá o se comportará de manera imprevisible. Para evitar este inconveniente, podemos incluir código que intercepte el error e indique cómo controlarlo (**rutinas de control de errores**). El proceso de interceptar y controlar errores en tiempo de ejecución es lo que se denomina interceptación de errores. Si nuestro código incluye rutinas de control de errores, necesitamos seleccionar la opción **interrumpir en errores no controlados**, para que se tengan en cuenta estas instrucciones.

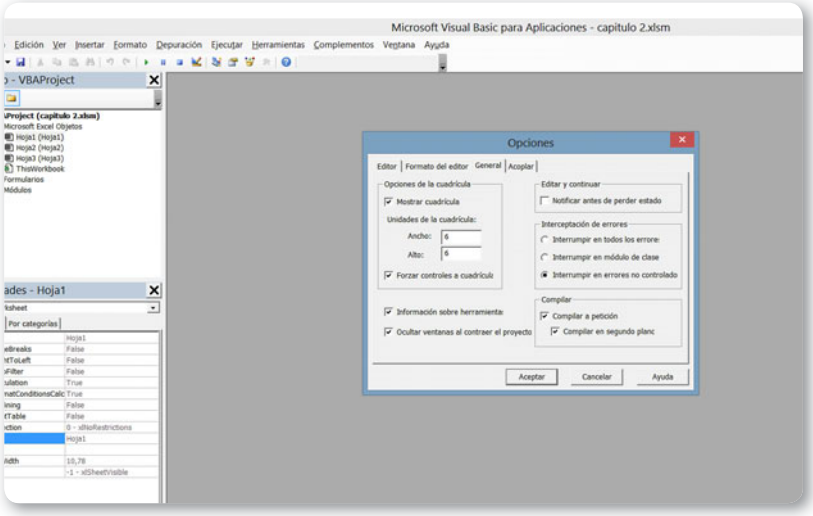


Figura 19. En la pestaña **General**, podemos seleccionar las diferentes opciones de interceptación de errores.

COMPILAR

Compilar se refiere al proceso por medio del cual un conjunto de instrucciones escritas o codificadas en un lenguaje de programación (lenguaje fuente) son convertidas a lenguaje máquina (código binario), de forma tal que estas instrucciones puedan ser ejecutadas por la computadora.

- **Compilar**: la opción **Compilar a petición** nos permite determinar si el proyecto debe ser compilado antes de iniciarse o si la compilación se realiza a medida que sea necesario. Si activamos **Compilar en segundo plano**, VBA utiliza el tiempo en el que el procesador está inactivo para compilar el código de nuestro procedimiento.

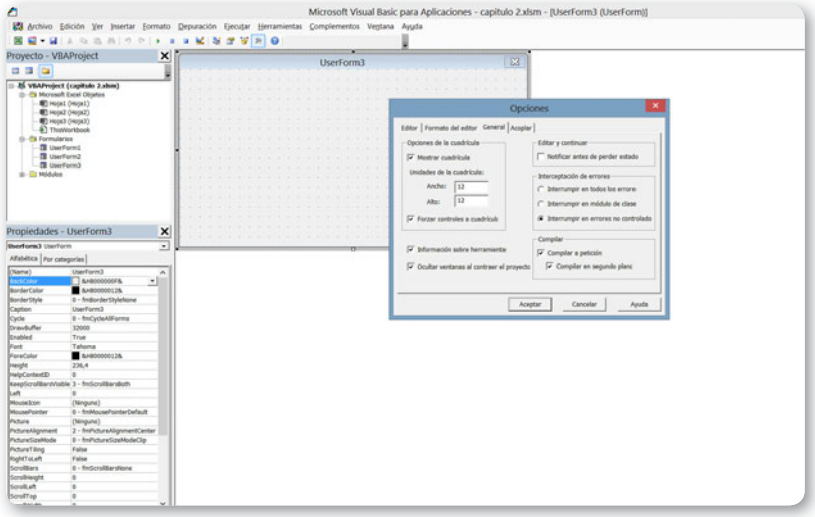


Figura 20. Podemos activar o desactivar la cuadrícula que se muestra en un **`Userform`**. Por defecto está activada.

RESUMEN

El Editor de Visual Basic se encuentra integrado en todas las aplicaciones del paquete Microsoft Office y permite programar desde una simple macro hasta el desarrollo de aplicaciones más complejas para automatizar diferentes tareas. En este capítulo, hicimos un recorrido por el entorno de desarrollo de Visual Basic (IDE, *Integrated Development Environment*), conocimos las diferentes barras de herramientas, las ventanas que lo componen, como así también aprendimos a configurar las opciones de esta herramienta.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuál es la función del Editor de VBA?
2. ¿Cuál es la combinación de teclas para acceder a VBA?
3. ¿Qué es un proyecto?
4. ¿Cuál es la función del **Explorador de Proyectos**?
5. ¿Qué provee el **Examinador de Objetos**?
6. ¿Cuál es la combinación de teclas para visualizar el **Explorador de Proyectos**?
7. ¿Para qué se emplea la ventana **Código**?
8. ¿Para qué se emplea la ventana **Propiedades**?
9. ¿Qué significa acoplar ventanas?
10. ¿Qué utilidad tiene la ventana **Inmediato**?

EJERCICIOS PRÁCTICOS

1. Acceda al editor de VBA.
2. Visualice la ventana **Código** haciendo doble clic en **ThisWorkbook**.
3. Acople la ventana **Código**.
4. Acceda a la ventana **Inmediato**.
5. A través de la ficha **Formato del editor**, modifique el tamaño de la fuente del elemento texto del error de sintaxis.

*03

Escribir sentencias con VBA

En este capítulo, comenzaremos a detallar la sintaxis del lenguaje de programación VBA y trabajaremos con algunos objetos de Excel. Aprenderemos a organizar el código de las macros en módulos, veremos qué son los procedimientos, y cómo escribirlos para introducir datos y visualizar resultados. También explicaremos cómo exportar o importar un módulo.

Procedimientos

Como mencionamos en el capítulo anterior, en la ventana del **Explorador de proyectos** vemos todos los objetos que pueden contener código VBA (los objetos **Hojas**, **ThisWorkbook**, **Módulos**, **UserFoms**). Dentro de estos objetos, vamos a organizar el código por medio de los procedimientos. En Excel, podemos distinguir tres tipos de procedimientos:

- Los procedimientos **Sub**, también llamados subrutinas.
- Los procedimientos **Function**, que son las funciones.
- Los procedimientos **Property** o procedimientos de propiedad.

Ámbito de los procedimentos

De forma predeterminada, los procedimientos son públicos (**Public**) en todos los módulos. Esto significa que se los puede llamar desde cualquier parte del proyecto, es decir, cuando un procedimiento es llamado para su ejecución, VBA lo busca en el módulo donde nos encontremos, si no lo encuentra, entonces continuará la búsqueda en el resto de los módulos del proyecto.

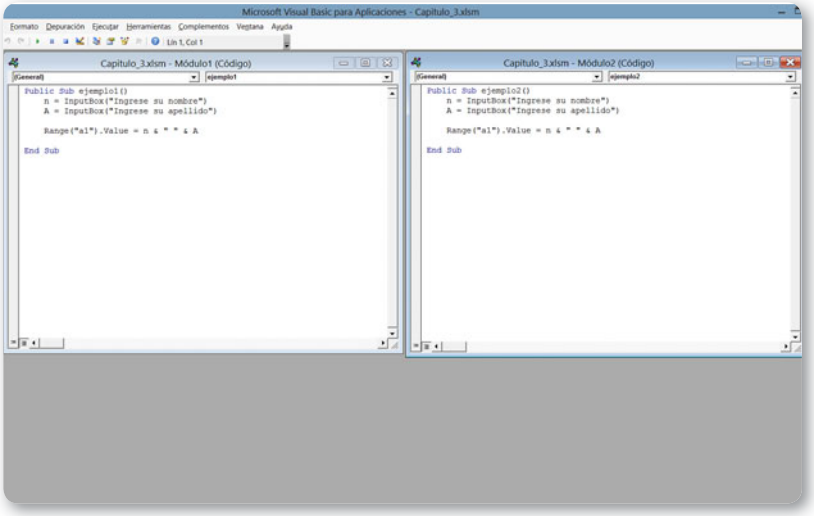


Figura 1. Un mismo procedimiento declarado como público (**`Public Sub`**) y como privado (**`Private Sub`**).

En cambio, un procedimiento privado (**Private**) solo puede ser llamado desde otros procedimientos que se encuentren en el mismo módulo.

Los procedimientos Sub

Podemos decir que los **procedimientos Sub** son el conjunto de códigos VBA contenido por las sentencias **Sub** y **End Sub**, que realizan una serie de acciones específicas. Pueden estar compuestos por todas las líneas de código que necesitemos, sin embargo, los procedimientos largos se vuelven más complejos. Por esto, es aconsejable dividirlos en procedimientos más pequeños, para que cada uno desarrolle una tarea. Así será más sencillo encontrar o modificar el código de una macro.

Es posible invocar o llamar a un procedimiento desde otro. Cuando un procedimiento llama a otro, el control se transfiere al segundo y, cuando finaliza la ejecución del código del segundo procedimiento, este devuelve el control al que lo llamó.

Distinguimos dos tipos de procedimientos **Sub**: los procedimientos generales y los procedimientos de eventos.

- **Procedimientos generales**: son los procedimientos declarados en un módulo. Una vez que se define este tipo de procedimiento, se lo debe llamar específicamente desde el código.

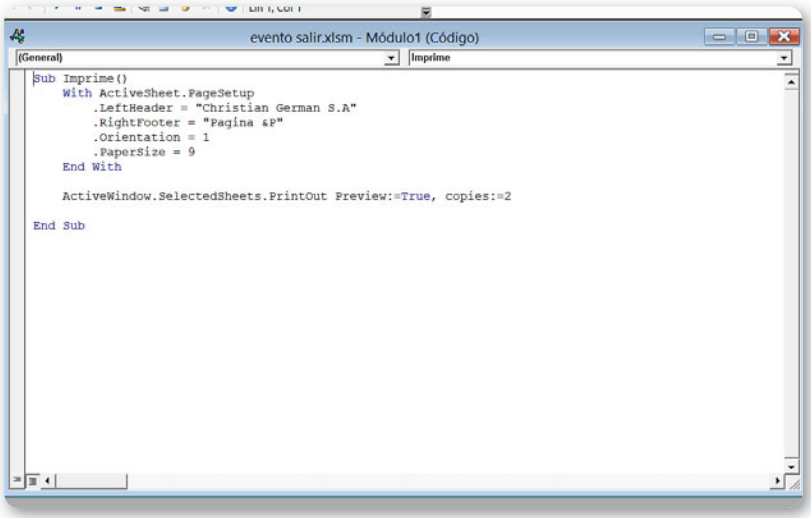


Figura 2. Un procedimiento general se debe llamar desde la aplicación.

- **Procedimientos de evento**: este tipo de procedimiento permanece inactivo hasta que se lo llama para responder a eventos provocados por el usuario o desencadenados por el sistema. Cuando un objeto en VBA reconoce que se ha producido un evento, llama automáticamente al procedimiento de evento utilizando el nombre correspondiente al evento. Más adelante, hablaremos sobre este tipo de procedimientos.

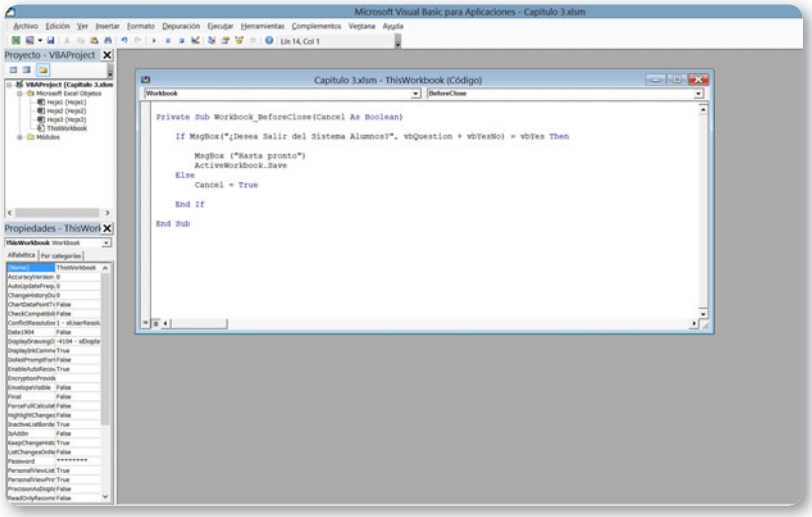


Figura 3. Este procedimiento muestra un mensaje antes de que se cierre el libro (evento **`BeforeClose`**).

Sintaxis de los procedimientos Sub

Todo procedimiento **Sub** obligatoriamente debe comenzar con la sentencia **Sub**, que indica el inicio, seguido del **nombre** del

PROGRAMA

Un **programa** es un conjunto de instrucciones, también llamadas **comandos**, que se procesan ordenadamente durante su ejecución. VBA es un leguaje de programación que ofrece una serie de comandos con los que podemos crear programas. En VBA, los programas forman parte de un procedimiento almacenado dentro de un módulo.

procedimiento y de la sentencia **End Sub**, que marca el fin. La sentencia **Sub** también puede ir precedida de otras expresiones, que tienen la función de delimitar el ámbito del procedimiento, y de una lista de argumentos, es decir, un conjunto de datos, para que el procedimiento pueda usarlos de manera interna.

La sintaxis que define a un procedimiento es la siguiente:

```
[Private | Public | Friend] [Static] Sub nombre [(lista de argumentos)]
    [Instrucciones]
    [Exit Sub]
    [Instrucciones]
End Sub
```

Donde:

- **Public**: indica que el procedimiento está disponible para todos los módulos del proyecto. Este dato es opcional.
- **Private**: determina que el procedimiento solo puede ser llamado desde otros procedimientos que estén en el mismo módulo. Este dato es opcional.
- **Friend**: se emplea únicamente en un módulo de clase. Podemos declarar como **Friend** a aquellos procedimientos de la clase que queremos poner a disposición de otras clases dentro del mismo procedimiento. Este dato es opcional.
- **Static**: indica que las variables del procedimiento se mantendrán entre una llamada y otra. Este dato es opcional.
- **nombre**: es el nombre del procedimiento **Sub**. Este dato es requerido.
- **lista de argumentos**: es una lista de variables separadas por comas que se pasan al procedimiento **Sub** cuando se lo invoca. Es opcional.

ELEMENTOS DE LA SINTAXIS

En la sintaxis para escribir un procedimiento, algunos elementos están encerrados entre corchetes ([]), y algunas palabras están separadas por una barra vertical (|). El corchete nos está indicando que los elementos son opcionales, y la barra vertical, que es posible seleccionar más de una palabra.

- **Instrucciones**: es el conjunto de sentencias que se ejecutan dentro del procedimiento **Sub**. Cada instrucción se escribe en una línea diferente. Este dato es opcional.
- **Exit Sub:** permite salir de un procedimiento. No es necesario a no ser que se necesite retornar a la sentencia situada inmediatamente a continuación de la que efectuó la llamada antes que el procedimiento finalice.
- **End Sub:** marca el cierre del procedimiento **Sub**.

Por ejemplo, la sintaxis de un procedimiento que nos permite ingresar el nombre y el apellido en la celda **A1** sería la siguiente:

```
Private Sub ejemplo()
   n = InputBox("Ingrese su nombre")
   A = InputBox("Ingrese su apellido")
   Range("a1").Value = n & " " & A
End Sub
```

Los procedimientos Function

Como sabemos, Microsoft Excel proporciona un gran conjunto de funciones predefinidas o internas, tales como **Suma**, **Promedio**, **Max**, **Min**, entre otras. Pero, en ocasiones, necesitamos realizar cálculos más complejos para los cuales no existe una función disponible.

Por medio de los procedimientos **Function**, podemos crear nuevas funciones ampliando, de esta manera, las incorporadas en Excel. A este tipo de funciones, se las conoce como funciones definidas por el usuario.

Al igual que los procedimientos **Sub**, un procedimiento **Function** puede tomar argumentos, realizar un conjunto de acciones específicas y

FÓRMULAS

Dentro de una hoja de cálculo, una fórmula siempre comienza con el signo igual (=), y el resultado se almacena dentro de la celda que la contiene. En un módulo de Visual Basic, los valores se guardan dentro de las variables, que no están asociadas con posiciones de una hoja de cálculo.

cambiar el valor de los argumentos. El código de estos procedimientos está encerrado entre las sentencias **Function** y **En Function**.

A diferencia de los procedimientos **Sub**, los procedimientos **Function** pueden devolver un valor o resultado; por esta razón, cuando declaramos una función, es necesario establecer qué tipo de variable será el valor que va a devolver el procedimiento.

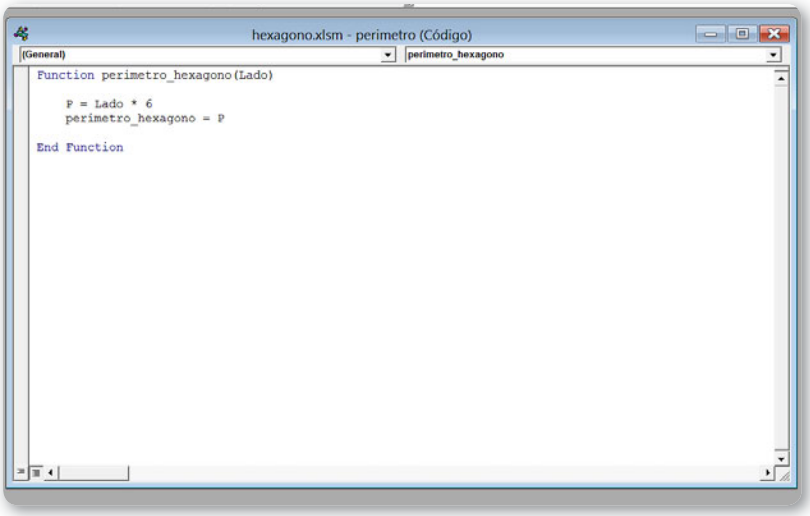


Figura 4. El código de esta función definida por el usuario devuelve el perímetro de un hexágono.

Sintaxis de los procedimientos Function

Todo procedimiento **Function** debe comenzar con la sentencia **Function** que indica el inicio, seguido del **nombre** de la función y la sentencia **End Function**, que marca el fin. La sintaxis que define a este tipo de procedimiento es la siguiente:

```
[Private | Public | Friend] [Static] Function nombre [(lista de argumentos)] [(As tipo)]
    [Instrucciones]
    [Exit Function]
    [Instrucciones]
End Function
```

Donde:

- **Public**: indica que el procedimiento **Function** está disponible para todos los módulos del proyecto. Este dato es opcional.
- **Private**: determina que el procedimiento **Function** solo puede ser llamado desde otros procedimientos que estén en el mismo módulo. Este dato es opcional.
- **Friend**: se emplea solamente en un módulo de clase. Podemos declarar como **Friend** a aquellos procedimientos de la clase que queremos poner a disposición de otras clases dentro del mismo procedimiento. Este dato es opcional.
- **Static**: indica que las variables del procedimiento **Function** se mantendrán entre una llamada y otra. Este dato es opcional.
- **nombre**: es el nombre del procedimiento **Function**. Este dato es requerido.
- **lista de argumentos**: es una lista de variables separadas por comas que se pasan al procedimiento cuando se lo invoca. Este dato es opcional.
- **As tipo**: determina el tipo de datos que devuelve la función. Este dato es opcional. En el **Capítulo 4**, veremos los tipos de datos.
- **Instrucciones**: es el conjunto de sentencias que se ejecutarán dentro del procedimiento. Cada instrucción se escribe en una línea diferente. Este dato es opcional.
- **Exit Function:** permite salir de una función. Solo se requiere si se necesita retornar a la sentencia situada inmediatamente a continuación de la que efectuó la llamada antes de que el procedimiento finalice.

Por ejemplo, la sintaxis de una función que nos permite calcular el área de un hexágono sería la siguiente:

ARGUMENTOS DE LOS PROCEDIMENTOS

Con frecuencia, el código de un procedimiento necesita cierta información para realizar su trabajo. Esta información se pasa a través de variables. Cuando una variable se pasa a un procedimiento, esta recibe el nombre de **argumento**. En el próximo capítulo analizaremos el uso de las variables y las constantes.

```
Function area_hexagono(Lado, Apotema)
    P = Lado * 6
    A = (P * Apotema) / 2
    area_hexagono = A
End Function
```

Los procedimientos Property

Los procedimientos **Property** se emplean para crear y personalizar las propiedades de los objetos de Microsoft Excel. Este tipo de procedimientos se declaran por defecto como públicos, aunque también es posible declararlos como privados. No entraremos en detalles, ya que el tema es muy amplio y no se abordará en este libro.

Trabajar con los procedimientos

Como ya mencionamos, podemos crear un procedimiento en los objetos **Hojas**, **ThisWorkbook**, **Módulos** y **UserFoms**. Sin embargo, es conveniente utilizar módulos para organizar mejor el código. Por ejemplo, en un módulo, podríamos tener todo el código que usamos para la edición de texto; en otro, todo lo relacionado con cálculos de porcentajes, y así sucesivamente. A continuación, veremos cómo podemos insertar módulos.

> AL TRABAJAR CON PROCEDIMIENTOS ES CONVENIENTE ORGANIZAR EL CÓDIGO EN MÓDULOS

Insertar módulos

Para agregar un modulo al proyecto, dentro del editor de VBA, vamos al menú **Insertar** y seleccionamos **Módulo**.

Al insertar el módulo, VBA le asigna un nombre por defecto, en este caso **Módulo1**, pero podemos modificarlo a través de la ventana **Propiedades**. Para esto, seleccionamos el **Módulo1** en el **Explorador de**

proyectos, vamos a la ventana **Propiedades**, hacemos clic en **Name**, y escribimos un nombre nuevo.

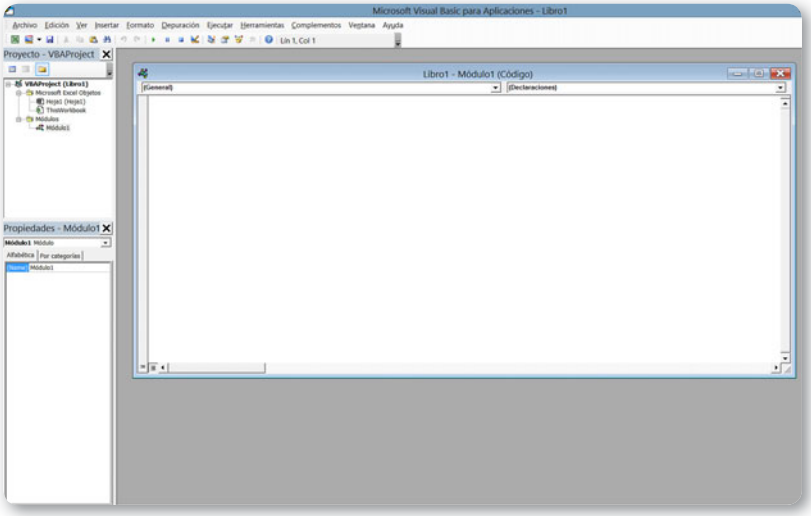


Figura 5. Insertamos el **Módulo1**, que permanecerá con el libro aún cuando esté cerrado y estará disponible cuando lo volvamos a abrir.

Eliminar un módulo

Si necesitamos eliminar un módulo del proyecto, debemos realizar un procedimiento muy sencillo. Primero, debemos dirigirnos al **Explorador de proyectos** y tenemos que seleccionar el módulo que queremos eliminar. Luego, hacemos un clic con el botón derecho del mouse y, en el menú que se abre, seleccionamos **Quitar Ejemplo...**.

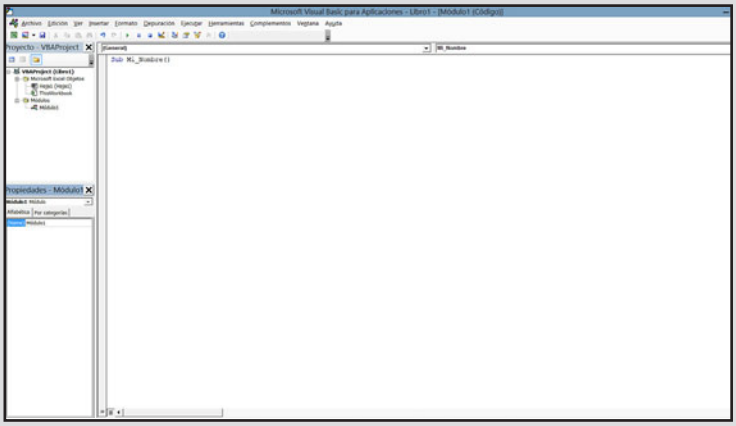
A continuación, aparecerá un cuadro de diálogo que nos preguntará si deseamos exportar el módulo antes de eliminarlo. Pulsamos **No**, y entonces el módulo se habrá eliminado.

Crear procedimientos Sub

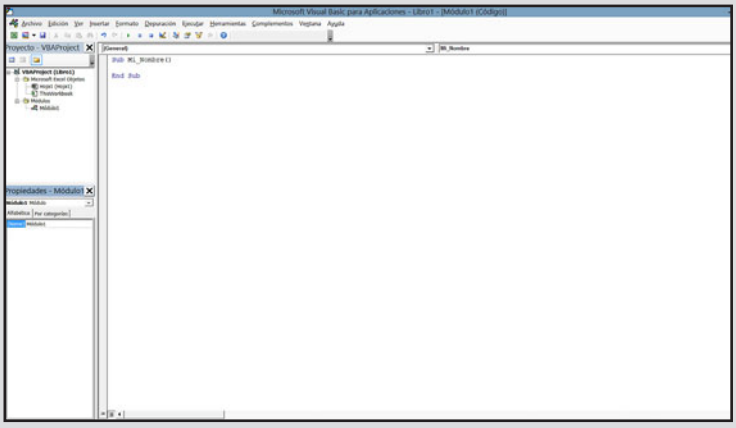
Para crear un procedimiento **Sub**, podemos hacerlo escribiendo directamente en la ventana **Código**. En el siguiente **Paso a paso**, veremos cómo generar un simple procedimiento que nos permite ingresar en la celda activa un nombre con fuente **Comic Sans Ms**, en tamaño **9**, color **Rojo** y alineación centrada.

PXP: CREAR UN PROCEDIMIENTO SUB

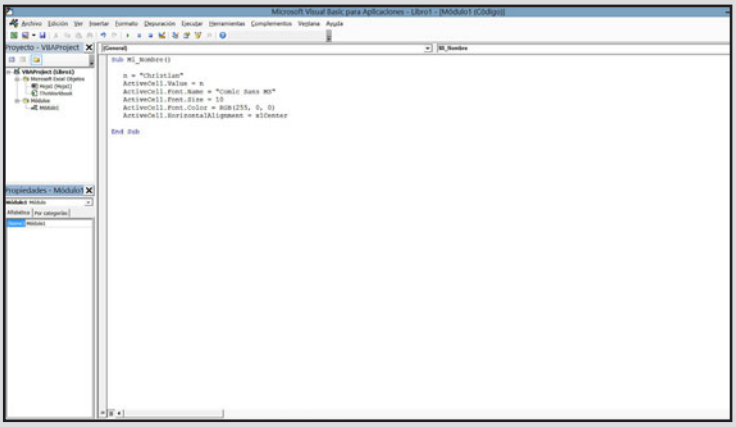
01 Abra un nuevo libro de Excel, ingrese al Editor de Visual Basic e inserte un nuevo módulo. En la ventana **Código**, escriba la palabra **Sub** y, a continuación, dejando un espacio, el nombre del procedimiento, por ejemplo: **Mi_Nombre**.



02 A continuación, presione ENTER. El editor de VBA añade después del nombre unos paréntesis y, luego, inserta una nueva línea con las instrucciones End Sub.



03 Escriba entre estas dos líneas el código que muestra la imagen, que le aplicará el formato al nombre.



También podemos crear un procedimiento **Sub** utilizando el cuadro de diálogo **Agregar procedimiento** del menú **Insertar/Procedimiento**. Se abrirá un cuadro de diálogo en el cual debemos seleccionar algunas opciones y, luego, presionamos **Aceptar**. Veremos que, en la ventana de código, aparece lo siguiente:

```
Public Sub Saludo()
End Sub
```

PALABRA COMPLETA

Si introducimos las primeras letras de una palabra clave en la ventana de código y presionamos la combinación de teclas **CTRL + Barra espaciadora**, VBA completará la palabra por nosotros. Si hay varias palabras clave que comienzan con las mismas letras, al pulsar **CTRL + Barra espaciadora**, VBA mostrará un menú con la lista de todas las palabras clave.

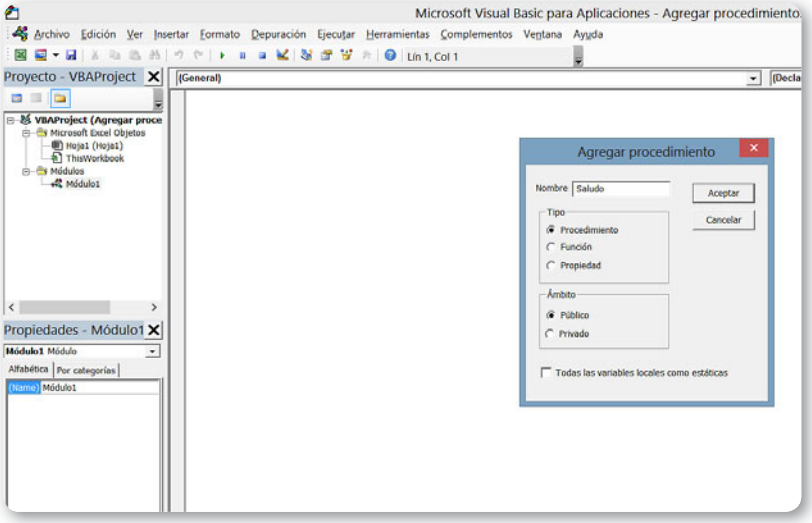


Figura 6. En el cuadro de diálogo **`Agregar procedimiento`**, especificamos el tipo, ámbito y el nombre del procedimiento.

Crear procedimentos Function

Al igual que como vimos en los procedimientos **Sub**, los procedimientos **Function** los podemos crear en cualquier objeto contenedor de código. También es algo muy sencillo de realizar, pero para entenderlo de manera clara y aprenderlo bien desde el principio, vamos a hacerlo través de un pequeño ejercicio.

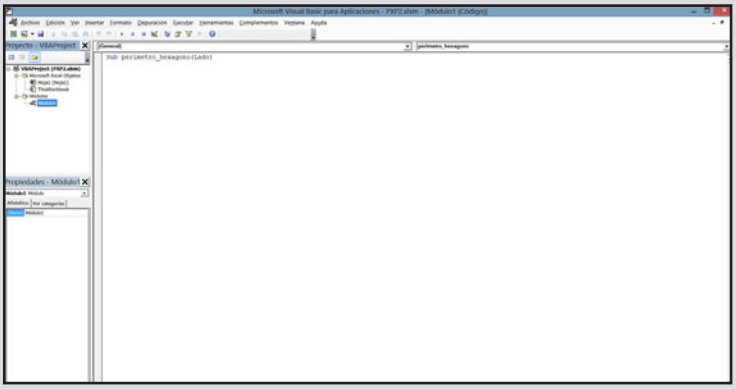
En el **Paso a paso**, que presentamos a continuación, describimos en detalle el procedimiento que debemos realizar para crear una función en un módulo que nos permitará calcular el perímetro de un hexágono. Solo tenemos que seguir cada uno de los pasos indicados.

AGREGAR COMENTARIOS

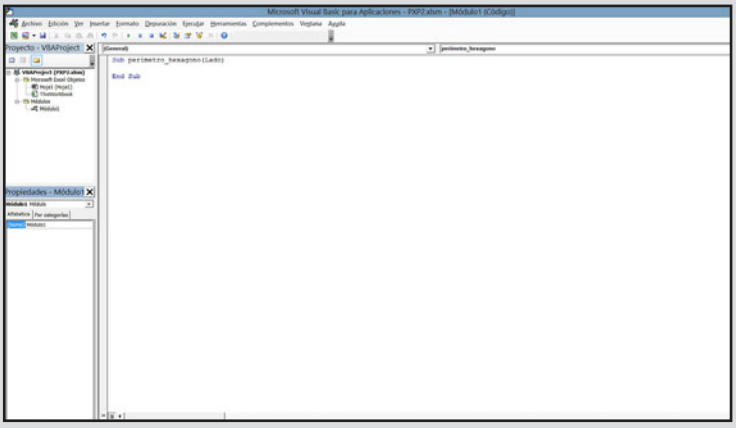
Debemos asegurarnos de que el código de los procedimientos VBA contenga todos los comentarios que consideremos indispensables. Los comentarios, por lo general, deben colocarse en el inicio de un procedimiento, y pueden contener diferentes tipos de detalles que, por ejemplo, incluyen la fecha en que se escribió el procedimiento, su versión, el nombre del programador y cualquier otro dato que pueda ayudar a entender el código contenido en el procedimiento.

PXP: CREAR UN PROCEDIMIENTO FUNCTION

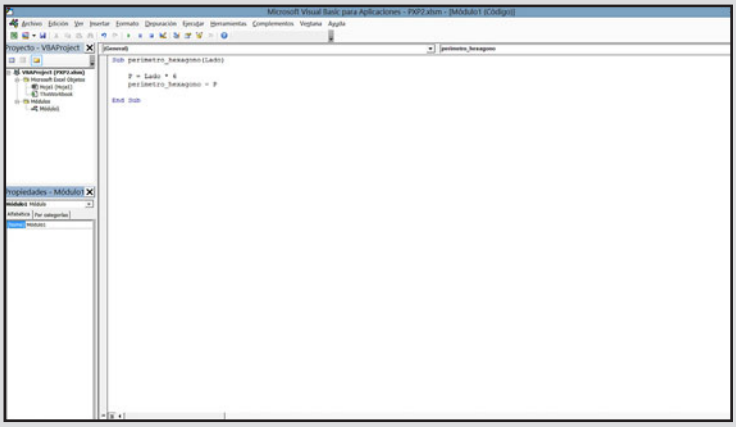
01 En un nuevo libro, ingrese al Editor de Visual Basic e inserte un módulo. En la ventana Código, escriba `Function` y, a continuación, dejando un espacio, el nombre del procedimiento, por ejemplo: `perimetro_hexagono (Lado)`.



02 Una vez ingresado el nombre, presione ENTER. El editor de VBA añade unos paréntesis después del nombre y, a continuación, inserta una nueva línea con las instrucciones `End Function`.



03 Entre estas dos líneas, escriba el código que se muestra en la imagen, que contiene la fórmula para calcular el perímetro.



Como vimos antes, podemos utilizar la sentencia **Exit Function** para salir de una función antes de que esta finalice. Por ejemplo, tenemos la función **calculo_perimetro** que espera recibir un valor positivo y mayor que cero para realizar el cálculo. Entonces, si el valor recibido es menor que cero, la función no podrá continuar, mostrará un mensaje de error y, como consecuencia, saldrá de la función de inmediato.

Por lo tanto, para detener la ejecución de la función cuando recibe un valor menor que cero, empleamos la sentencia **Exit Function** como muestra la siguiente sintaxis:

INTELLISENCE

IntelliSence es una interesante herramienta que nos permite ahorrar trabajo y tiempo en las búsquedas y en la escritura de código. Muestra, en la ventana de código, una lista de las propiedades y los métodos de un objeto después de que hayamos escrito el nombre de un objeto o de una colección conocida por Visual Basic para Aplicaciones seguida de un punto.

```
Function calculo_perimetro(Lado)
   If Lado <= 0 Then
       MsgBox "Ingresar un número mayor que 0"
      Exit Function
   Else
      P = Lado * 6
      calculo_perimetro = P
   End If
End Function
```

Conceptos básicos del código

Existen ciertas reglas que debemos respetar para la escritura de código, como las convenciones de asignación de nombres a los procedimientos, las variables y los argumentos; la combinación de líneas de código; las sangrías y el agregado de comentarios. A continuación, veremos en detalle todos estos requisitos que tenemos que cumplir.

Reglas de asignación de nombres

Debemos seguir algunas reglas para dar nombre a los procedimientos, variables y argumentos en un módulo de Visual Basic:

- El nombre no puede tener más de 255 caracteres de longitud, y el primer carácter debe ser una letra.
- No puede contener espacio ni caracteres especiales, tales como punto (**.**), signo de exclamación (**!**), arroba (**@**), ampersand (**&**), símbolo peso (**$**), símbolo numeral (**#**).
- No es posible usar nombres reservados de VBA (**palabras clave**) como **Private**, **If** o **Workbook**, entre otras.
- Las minúsculas y mayúsculas no se diferencian. El Editor de VBA ajusta automáticamente la escritura.

Dividir una instrucción en varias líneas

Aunque una línea de código de VBA puede tener hasta 1024 caracteres; como dijimos anteriormente, es una buena práctica dividirla en varias líneas. Para dividir una instrucción larga en diferentes líneas en la ventana de código, utilizamos el guion bajo (_) precedido de un espacio.

Podemos utilizar el carácter de continuación de línea en los siguientes lugares del código:

- Antes o después de los operadores, como **&**, **+**, **-**.
- Antes o después de una coma (**,**).
- Antes o después de un signo igual (**=**).

No es posible utilizarlo en los siguientes casos:

- Entre los dos puntos (**:**) y el signo igual (**=**).
- En el texto entre comillas.

> AUNQUE UNA LÍNEA DE CÓDIGO PUEDE TENER 1024 CARACTERES, ES MEJOR DIVIDIRLA

Sangrías

Aunque los procedimientos escritos en VBA no exigen reglas estrictas en su escritura, es una práctica habitual realizar el sangrado para las líneas de código. El uso de sangrías ayudará a que el código sea más fácil de leer y comprender. Es recomendable utilizar sangrías para entrar las líneas de código que toman decisiones o acciones repetidas. Para aplicarlas en una o más líneas, empleamos la tecla **TAB**.

El siguiente ejemplo ilustra el uso de sangrías:

```
Sub area_circulo()
   r = 3
   Pi = 3.1416
   MsgBox "El area del circulo de radio: " & r & _
   " es: " & r * r * Pi
End Sub
```

Agregar comentarios al código

Un **comentario** es un texto explicativo precedido por un **apóstrofo** (’), que situamos en el código. Este símbolo le indica a VBA que pase por alto las palabras que van a continuación de él. Los comentarios son muy útiles para describir la funcionalidad del código de los procedimientos. Además, sirven para probar y solucionar problemas en los procedimientos VBA. Por ejemplo, cuando ejecutamos un procedimiento, puede que este no funcione como esperamos. En lugar de borrar las líneas de código que sospechamos que sean las causantes del problema, lo que podemos hacer es comentarlas, de este modo, las omitimos momentáneamente y seguimos comprobando otras partes del procedimiento.

LOS COMENTARIOS PERMITEN PROBAR Y SOLUCIONAR PROBLEMAS DEL CÓDIGO

Ejecutar un procedimiento

Podemos ejecutar un procedimiento **Sub** de diferentes maneras; a continuación, veremos algunas de ellas.

Ejecutar un procedimiento desde otro procedimiento

Podemos llamar a un procedimiento **Sub** desde otro procedimiento escribiendo su nombre dentro del procedimiento que lo llama. Por ejemplo, para llamar al procedimiento **mi_nombre**, escribimos el código:

```
Sub llama_proc()
   mi_nombre
End Sub
```

Si dos o más módulos contienen un procedimiento con el mismo nombre, debemos incluir el nombre del módulo, además del nombre

del procedimiento. Por ejemplo, supongamos que tenemos un proyecto que se llama **Ejemplos2.xls** con tres módulos: **Módulo1**, **Módulo2** y **Módulo3**, y en los **Módulo1** y **Módulo3** tenemos dos procedimientos llamados **encabezado**. Para llamar al procedimiento **encabezado** que se encuentra en el **Módulo3**, deberemos escribir la siguiente sintaxis:

```
Sub llama_proc2()
   Módulo 3.encabezado
End Sub
```

También podemos llamar a un procedimiento empleando la cláusula **Call**. Veamos el siguiente ejemplo:

```
Sub llama_proc2()
   Call mi_nombre
End Sub
```

Aunque no debamos pasarle ningún argumento a un procedimiento cuando lo invocamos, es aconsejable que empleemos la sentencia **Call**. De esta forma, podemos identificar de manera rápida las llamadas que realizamos a otros procedimientos.

Ejecutar un procedimiento desde el Editor de VBA

Para ejecutar un procedimiento desde el Editor de Visual Basic, debemos ingresar al menú **Ejecutar** y seleccionar las opciones **Ejecutar**

SENTENCIA CALL

Tengamos en cuenta que, si usamos **Call**, debemos poner entre paréntesis los parámetros que se pasan al procedimiento llamado, independientemente del hecho de que no haya ningún valor de retorno del procedimiento. También podemos utilizar **Call** con una función, pero solo si el valor de retorno no se utiliza.

Sub/UserForm. También podemos presionar la tecla **F5** o el **botón Ejecutar** de la barra de herramientas **Estándar**. Esta opción solo es válida si el procedimiento no requiere argumentos.

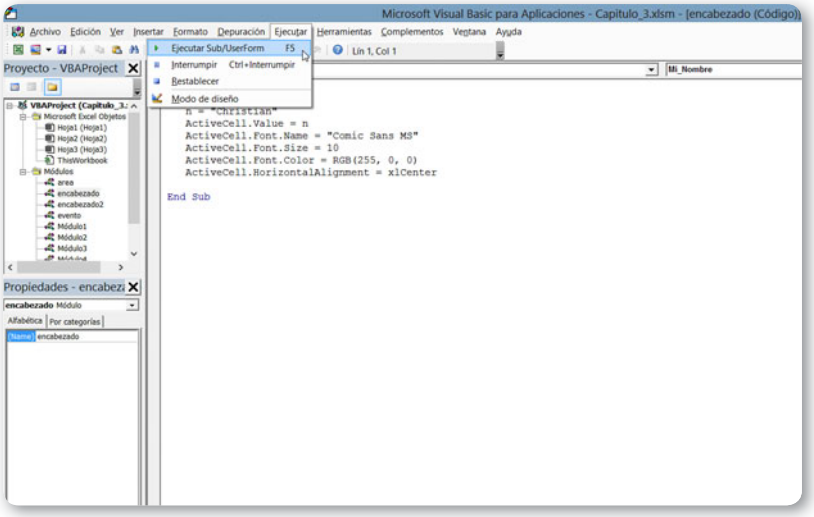


Figura 7. Desde el menú **Ejecutar**, podemos ejecutar un procedimiento en la ventana de VBE.

Ejecutar un procedimiento desde la ventana de Excel

También es posible ejecutar un procedimiento desde la ventana de Excel. Para realizar esto, debemos ir a la ficha **Desarrollador** y luego pulsar el botón **Macros**. En el cuadro de diálogo que se abre a continuación, seleccionamos el procedimiento que queremos ejecutar y, para finalizar, presionamos el botón **Ejecutar**.

NOMBRES DE PROCEDIMIENTOS

Si bien podemos asignar la palabra que queramos, cuando elegimos los nombres para los procedimientos, las variables e, incluso, las constantes es conveniente elegir denominaciones que sean significativas. Porque, de esta manera, lograremos una mayor claridad en la comprensión y el seguimiento de los procedimientos.

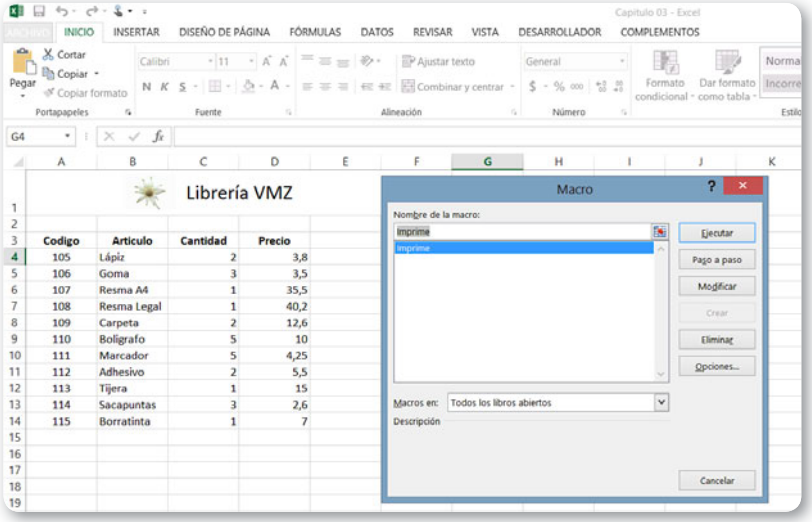


Figura 8. Podemos abrir el cuadro de diálogo **Macros** pulsando la combinación de teclas **ALT + F8**.

Ejecutar el procedimiento con una tecla de acceso directo

Otra manera de ejecutar un procedimiento es presionar una tecla de acceso directo asignada para este fin. Para esto, lo que primero debemos hacer es configurar qué tecla es la que vamos a utilizar. Desde la ventana de Excel, presionamos el botón **Macros** que se encuentra en la ficha **Desarrollador** y, en el cuadro de diálogo que se abre, seleccionamos **Opciones...**. A continuación, en el cuadro de diálogo denominado **Opciones de la macro**, tenemos que hacer un clic en **Tecla de método abreviado** y, luego, escribimos una letra en la casilla que se encuentra a continuación de **CTRL +**.

COMENTAR VARIAS LÍNEAS

Podemos comentar varias líneas de código, seleccionándolas y, luego, presionando el botón **Bloque con comentarios** de la barra de herramientas **Edición**. Para activar nuevamente el código comentado, seleccionamos las líneas y, luego, hacemos un clic en el botón **Bloque sin comentario** de la misma barra.

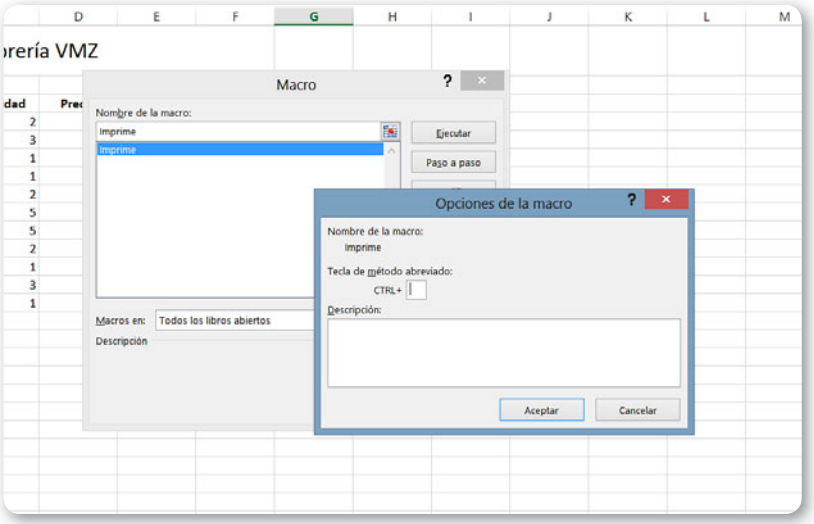


Figura 9. La letra introducida corresponde a la combinación de teclas para ejecutar la macro.

Ejecutar el procedimiento utilizando objetos

Otra manera de ejecutar un procedimiento es asignar la macro a un botón de control de formulario (esta herramienta la desarrollaremos en profundidad en el **Capítulo 8** dedicado a los formularios) o a cualquier otro objeto, como pueden ser las formas, las imágenes y los gráficos que se encuentran en la ventana de Excel.

Para asignar un procedimiento a un botón de formulario, vamos a la ficha **Desarrollador**, presionamos **Insertar** y, en el sector **Controles de formulario**, seleccionamos **Botón (control de formulario)**.

TENER EN CUENTA

Cuando le pasamos el control a una subrutina desde un procedimiento, esto hará que se detenga la ejecución del programa que la llamó, e iniciará la subrutina.

Esta subrutina continuará funcionando hasta que encuentre una declaración **End Sub** o una declaración de salida, y le pasará nuevamente el control al procedimiento que la llamó.

A continuación, en algún sector libre de la hoja de trabajo, hacemos un clic con el botón izquierdo del mouse y arrastramos el cursor para dibujar el botón. Al realizar esto, se abrirá el cuadro de diálogo **Asignar Macro**, en el que tenemos que seleccionar de la lista el nombre de la macro que necesitamos y luego presionamos **Aceptar**.

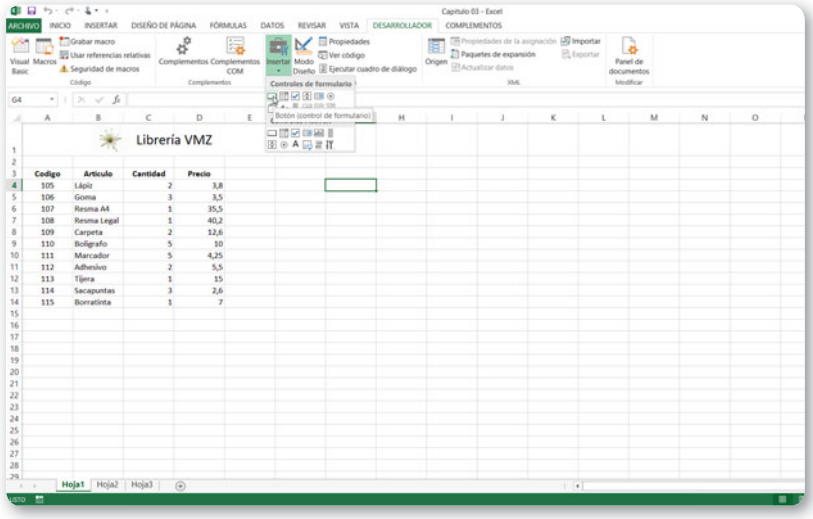


Figura 10. Los botones de formularios son los objetos ideales para asignar procedimientos.

Para asignar un procedimiento a un objeto, primero dibujamos alguna forma desde la ficha **Insertar**, seleccionamos el objeto con el botón derecho del mouse y, en el menú desplegable, elegimos la opción **Asignar Macro...**. En el cuadro de diálogo que se abre, seleccionamos el procedimiento que queremos asignarle al objeto.

EJECUTAR FUNCTION

Es importante tener en cuenta que no es posible ejecutar un procedimiento **Function** presionando la tecla **F5**, cuando el puntero del mouse se encuentra en la ventana de código del procedimiento de función en la ventana del Editor de Visual Basic. Tampoco podremos hacerlo desde la opción **Macro** que se encuentra en la ficha **Desarrollador** de Excel. Para habilitar estas dos opciones solo tenemos que cambiar la ubicación del puntero del mouse y luego realizar la operación.

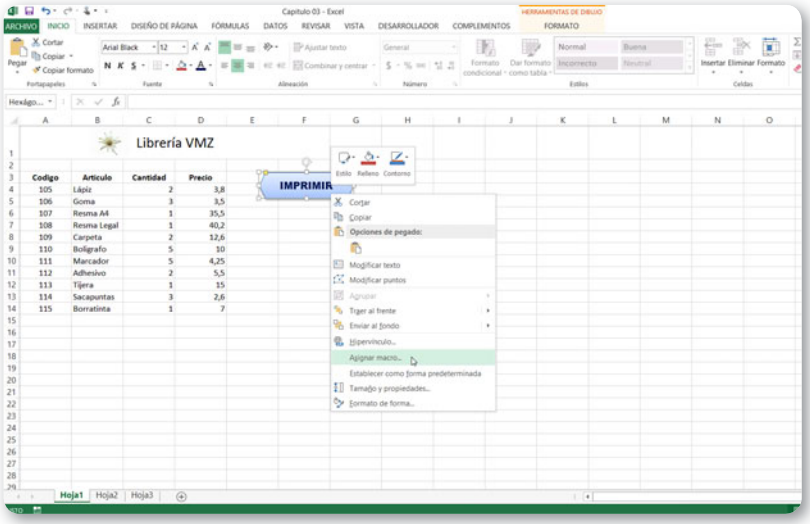


Figura 11. Al desplegar el menú contextual del objeto, encontramos la opción **`Asignar Macro`**....

Ejecutar funciones

A diferencia de los procedimientos **Sub**, los procedimientos **Function** se pueden ejecutar solamente de dos maneras:

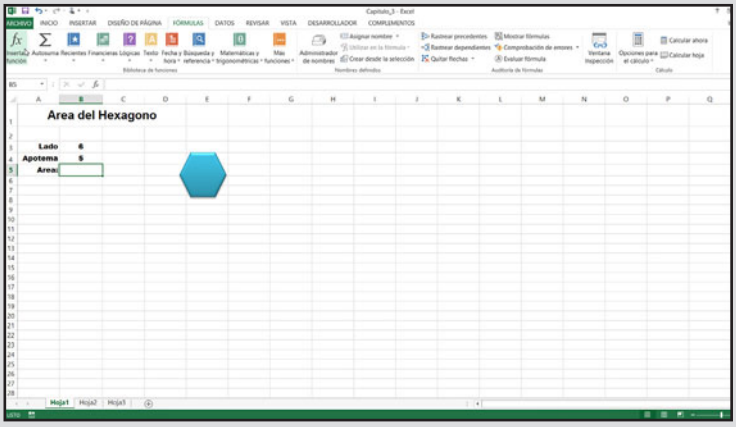
- Desde otro procedimiento **Sub** o **Function**.
- Dentro de una fórmula en una hoja de cálculo.

Ejecutar una función desde una hoja de cálculo

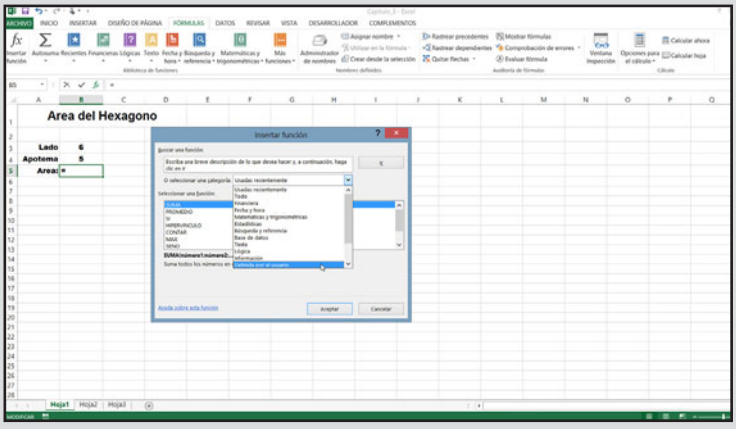
En este caso veremos cómo podemos ejecutar una función desde una hoja de cálculo. Tenemos, por ejemplo, la función personalizada **área_hexagono** que, como su nombre lo indica, nos permite calcular el área de un hexágono. Para utilizarla podemos proceder de dos maneras diferentes. La primera sería insertar la función desde la ficha **Fórmulas/ Insertar función** del grupo **Biblioteca de funciones**. En el siguiente **Paso a paso**, describimos este procedimiento.

PXP: LLAMAR UNA FUNCIÓN DESDE UNA HOJA

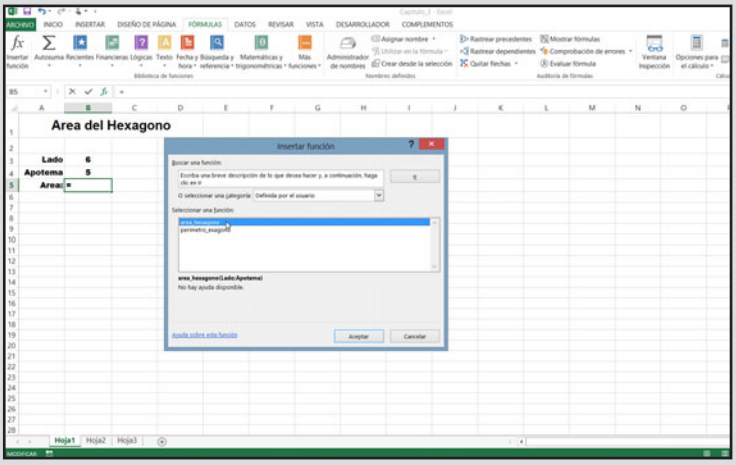
01 En una hoja de cálculo, ingrese los datos que muestra la imagen. Ubíquese en la celda B5 y, en **la ficha** Fórmulas, **seleccione** Insertar función.



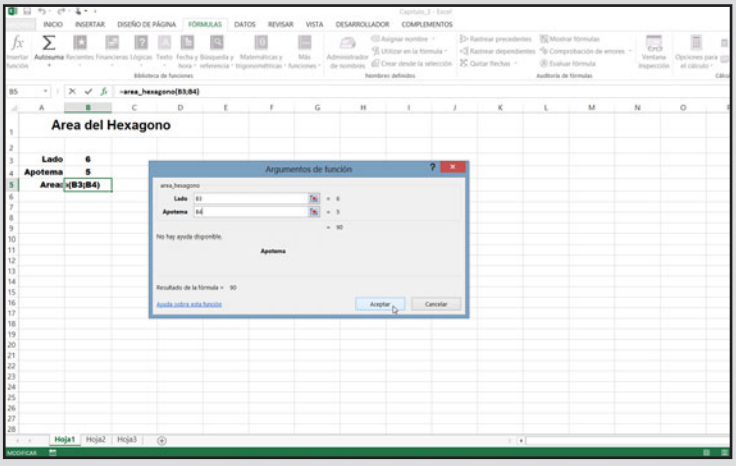
02 Excel abrirá el cuadro de diálogo Insertar función. En la lista desplegable O seleccionar una categoría, elija la opción Definida por el usuario.



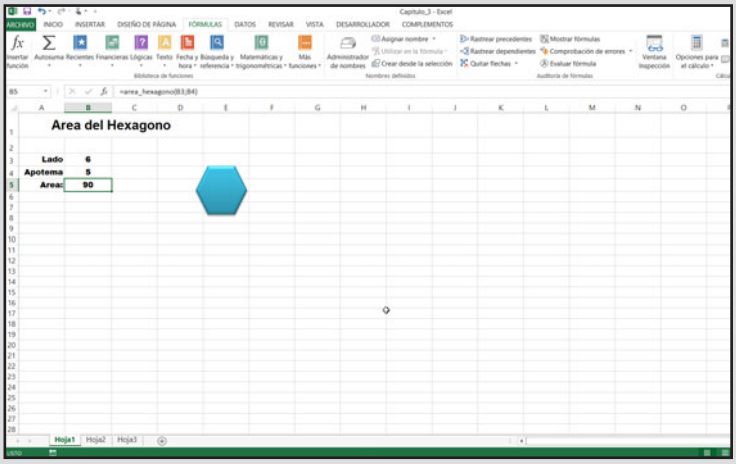
03 Seleccione la función que previamente habíamos creado como `Public` en el Editor de VBA, en este caso `área_hexagono`, y presione `Aceptar`.



04 En el cuadro de diálogo `Argumentos de función`, seleccione las celdas donde se encuentran los valores correspondientes y finalmente presione `Aceptar`.



05 En la hoja de cálculo, se ve el resultado de la función `area_hexagono`.



La segunda posibilidad para ejecutar una función desde una hoja de cálculo es aún más sencilla: debemos escribir directamente la función en la celda donde necesitamos obtener el resultado. Por ejemplo, primero nos ubicamos en la celda **B5**; luego, escribimos el símbolo igual (**=**), y, a continuación, ingresamos el nombre de la función seguido del paréntesis de apertura, seleccionamos los argumentos (lado y apotema), cerramos el paréntesis y presionamos **ENTER**.

PARA EJECUTAR UNA FUNCIÓN EN UNA HOJA DE CÁLCULO CONTAMOS CON DOS POSIBILIDADES

FUNCIÓN PÚBLICA

Un procedimiento **Función** es como una función integrada de Excel. Podemos usarlo todas las veces que queramos en la hoja de cálculo, siempre y cuando lo declaremos como **Public**. Si lo declaramos como **Private**, no estará disponible para el usuario y no aparecerá en el cuadro de diálogo **Insertar función**.

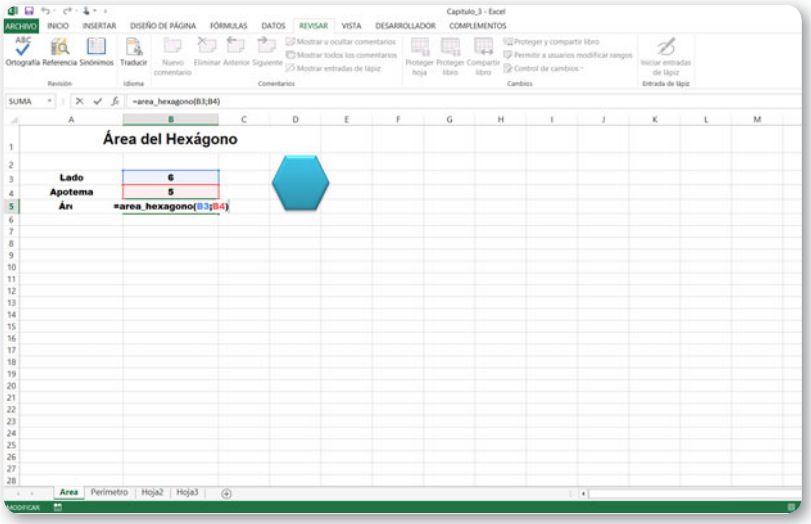


Figura 12. Cuando tenemos que trabajar con funciones complejas, es conveniente crear funciones definidas por el usuario (UDF).

Ejecutar una función desde otro procedimento

Para llamar una función, debemos hacerlo desde otro procedimiento. Para esto, tenemos que escribir en un módulo el nombre de la función seguida de los argumentos correspondientes. Por ejemplo, observemos el código que presentamos a continuación:

```
Sub llama_funcion()

   A = area_hexagono(6, 5)
   MsgBox A

End Sub
```

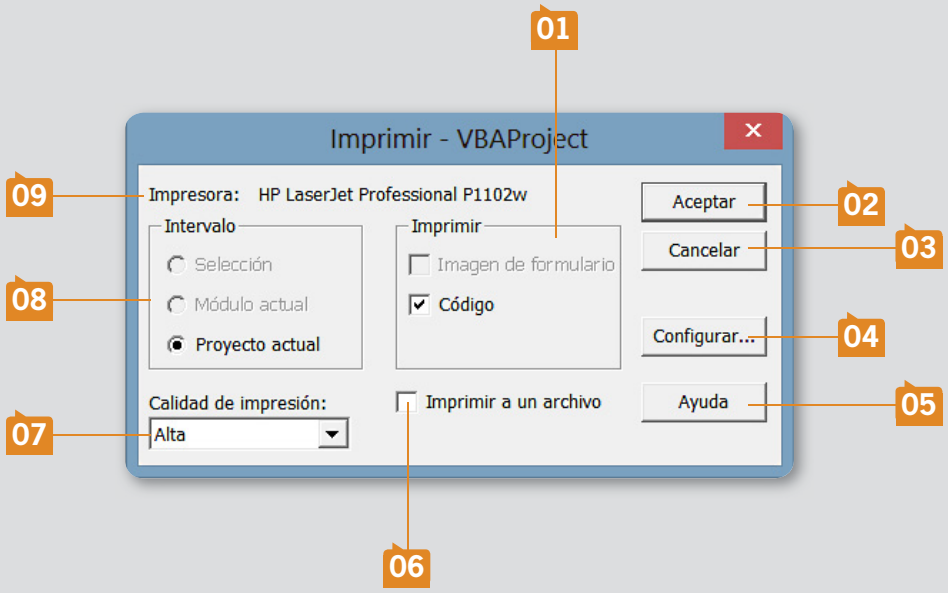
De esta manera, cuando ejecutemos el procedimiento denominado **llama_funcion**, Microsoft Excel nos mostrará un mensaje que contiene el resultado de la función ingresada.

Imprimir un módulo

Es posible imprimir el código de un módulo desde la ventana del editor de VBA. Para esto, vamos al menú **Archivo** y seleccionamos **Imprimir...**.

En la siguiente **Guía visual**, conoceremos las opciones de impresión.

GV: IMPRIMIR-VBAPROJECT



01 **Imprimir:** determina lo que se va a imprimir: **Imagen de Formulario**, **Código**. Se puede seleccionar más de una opción según lo que hayamos elegido en la opción **Intervalo**. Si nuestro proyecto contiene formularios, podremos imprimir únicamente la imagen del formulario o el código

02 **Aceptar:** activa la impresión de acuerdo con lo configurado anteriormente.

03 **Cancelar:** cancela la ventana de **Imprimir-VBAProject**.

04 **Configurar...:** permite seleccionar la impresora y configurar, por ejemplo, el tipo de papel y su orientación.

05 **Ayuda:** nos proporciona ayuda sobre la ventana **Imprimir**.

06 **Imprimir a un archivo:** envía la impresión al archivo especificado en el cuadro de diálogo **Imprimir en un archivo**.

07 **Calidad de impresión:** permite elegir entre alta, media, baja o borrador.

08 **Intervalo:** permite elegir qué parte del proyecto vamos a imprimir. **Selección** imprime el código seleccionado. **Módulo actual** imprime los formularios o el código del módulo seleccionado. **Proyecto actual** imprime los formularios o el código de todo el proyecto.

09 **Impresora:** identifica el dispositivo en el que se va a realizar la impresión.

Importar y exportar código

En ocasiones, podemos necesitar emplear el código de un módulo o un formulario creado en un proyecto para otro proyecto. En estas situaciones, es posible utilizar las opciones para exportar e importar del menú **Archivo** de la ventana de VBE.

Para **exportar un archivo**, vamos a la ventana del **Explorador de proyectos** y seleccionamos el objeto en cuestión. Hacemos clic en el menú **Archivo** y, luego, pulsamos **Exportar archivo...**. En el cuadro de diálogo que se abre, elegimos la carpeta donde vamos a guardar el archivo e ingresamos un nombre. Luego, presionamos **Aceptar**.

FUNCIONES PERSONALIZADAS

La función creada en un libro de Excel, solo estará disponible mientras el libro esté abierto. Para que pueda insertarse en cualquier libro, debemos guardarla en el libro personal de macros. Para esto, arrastramos el módulo que contiene la función a la carpeta **Personal.XLSB** del **Explorador de proyectos.**

Para **importar un archivo**, abrimos el proyecto donde queremos ingresarlo. En la ventana del Editor de VBA, hacemos clic en el menú **Archivo** y seleccionamos la opción **Importar archivo...**. En el cuadro de diálogo, seleccionamos la carpeta donde se encuentra el archivo y, luego, el nombre del archivo. Para finalizar, presionamos **Aceptar**.

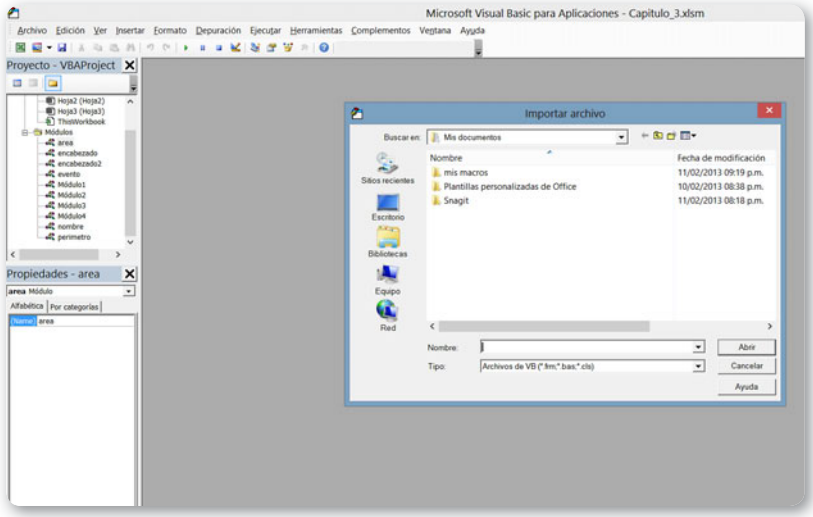


Figura 13. Este cuadro de diálogo nos permite importar un módulo o formulario.

RESUMEN

En este capítulo, aprendimos muchos conceptos clave que emplearemos posteriormente para desarrollar los procedimientos que nos permitirán automatizar tareas. Conocimos los diferentes procedimientos que existen en VBA, y la manera de crearlos y ejecutarlos. Aprendimos a escribir diferentes procedimientos siguiendo las reglas gramaticales y de sintaxis. Por último, vimos cómo podemos reutilizar los procedimientos en otros proyectos mediante las herramientas de exportación e importación.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es un procedimiento?
2. ¿Para qué se emplean los procedimientos **Sub**?
3. ¿Cuál es la sintaxis de un procedimiento **Function**?
4. ¿Cómo se puede ejecutar un procedimiento **Sub**?
5. ¿Qué es una subrutina?
6. ¿Cómo se llama a una subrutina?
7. ¿Cuál es el ámbito de los procedimientos?
8. ¿Cómo se puede importar un procedimiento?
9. ¿Cómo se imprime el código de un módulo?
10. ¿Cómo puedo dividir una instrucción en varias líneas?

EJERCICIOS PRÁCTICOS

1. Inserte un nuevo módulo.
2. Cámbiele la propiedad **Name** por **practico_1**.
3. Escriba el procedimiento del archivo **Cap3_actividad03.doc** que se encuentra en el sitio **http://www.redusers.com/premium/notas_contenidos/macrosexcel2013/**.
4. Ejecute el procedimiento creado desde la ventana de VBE.
5. Cree un botón en la hoja de cálculo y asigne a este botón el procedimiento creado. Ejecute el procedimiento a través del botón.

*04

Los datos en VBA

En Microsoft Excel podemos introducir diferentes clases de datos en las celdas y aplicarles un formato. Esto describe a una variable en cualquier lenguaje de programación. En este capítulo, conoceremos qué son las variables, y los diferentes tipos que existen. También aprenderemos a declararlas en VBA para usarlas en los distintos procedimientos.

Las variables

A menudo necesitaremos guardar valores temporalmente cuando realicemos cálculos con VBA. Por ejemplo, podemos almacenar los valores de las ventas y gastos totales de un negocio para luego usarlos en un cálculo de margen de ganancia.

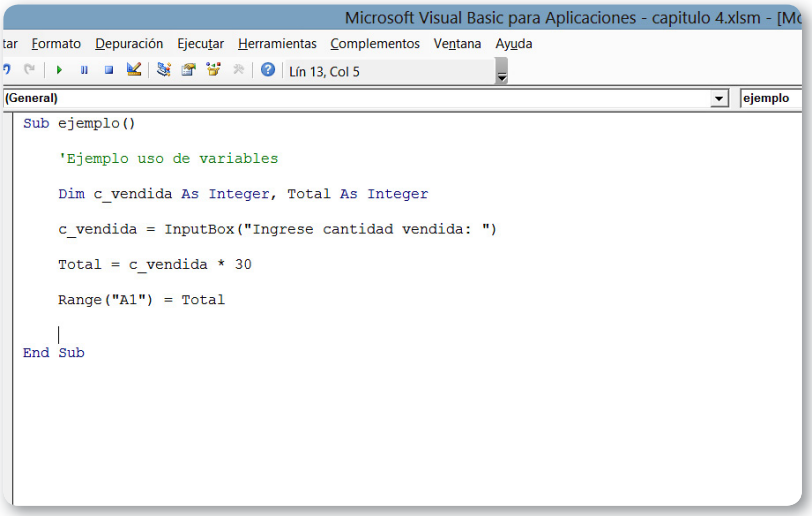


Figura 1. La expresión `Total` devolverá un valor distinto cada vez, dependiendo de los valores que indique el usuario.

UNA VARIABLE ES EL ELEMENTO DEL CÓDIGO QUE USAMOS PARA GUARDAR UN VALOR TEMPORAL

Una **variable** es el elemento del código que empleamos para guardar valores o información temporal que luego usaremos en la ejecución de un programa. El contenido que se guarda en la variable es la información que queremos manipular y se llama **valor** de la variable.

Podemos decir que las variables son las palabras mediante las cuales vamos a hacer referencia a una determinada información dentro de nuestro programa. El uso de variables se debe a que, en muchas ocasiones, no tendremos la información que queremos manipular de antemano, sino que esta se va a generar o a solicitar en un determinado momento. Por ejemplo, podemos pedirle

al usuario, mientras se ejecuta el programa, que ingrese la cantidad vendida de un producto. Luego, esta información la guardamos bajo el nombre **c_vendida**. De esta manera, a lo largo del desarrollo del programa, cuando tengamos que referirnos a la cantidad vendida que ingresó el usuario, simplemente hacemos referencia a **c_vendida**, en lugar de indicar la cantidad real ingresada por dicho usuario.

> EN VBA, PODEMOS DECLARAR LAS VARIABLES DE DOS MANERAS: IMPLÍCITA O EXPLÍCITA

Declaración de variables

Toda información que manejemos requiere un uso de la memoria RAM de la computadora que ejecuta nuestra aplicación, por eso es indispensable reservar un lugar de memoria para cada variable antes de utilizarla. Esta acción se conoce como **declaración**, inicialización o definición de la variable. Existen dos maneras de declarar las variables en VBA: **implícita** o **explícita**.

Declaración implícita

En VBA, no es necesario declarar una variable antes de usarla, porque la aplicación la creará en forma automática cuando le asignemos un valor al nombre de dicha variable. Por ejemplo, podemos crear de manera implícita las siguientes variables:

```
x = 12
recargo = 20
nombre = "Viviana"
```

NOMBRES DE VARIABLES

Al igual que en los procedimientos, podemos identificar a las variables en VBA con cualquier nombre válido. Es decir, el nombre debe comenzar por una letra y no por un número, y puede tener hasta 250 caracteres de longitud sin espacios. Tampoco puede ser igual a ninguna de las palabras clave de VBA Excel.

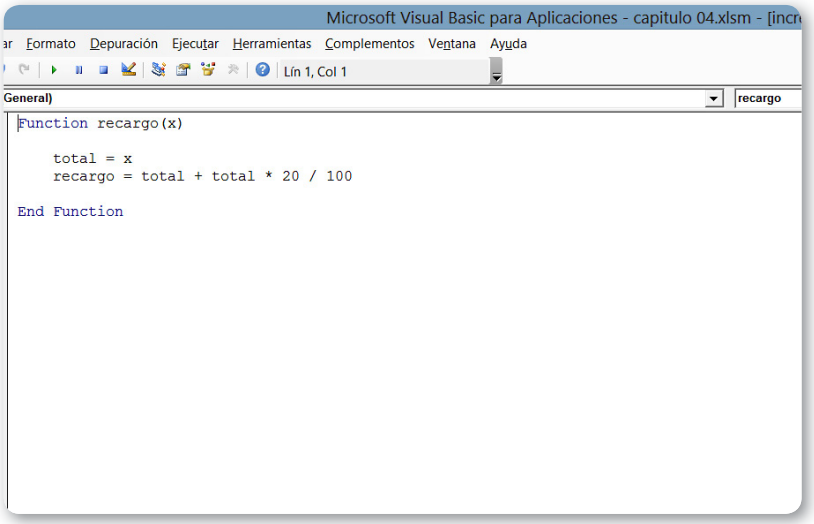


Figura 2. Una declaración de este tipo a veces es conveniente, pero limita el control sobre las variables

Declaración explícita

Declarar una variable de manera explícita significa que debemos definir los nombres de todas las variables que vamos a utilizar antes de emplearlas en el código VBA.

Para poder realizar esta declaración, tenemos que usar la instrucción **Dim**, seguida del nombre de la variable. Por ejemplo, si queremos crear la variable **nombre**, podemos usar la siguiente instrucción: **Dim nombre**. Si necesitamos crear la variable **domicilio**, tenemos que utilizar la instrucción: **Dim domicilio**. De esta manera debemos proceder con cada una de las variables que vayamos a incluir en el código.

LA NOTACIÓN CIENTÍFICA

La **notación científica** es una convención matemática que expresa un número entero o decimal como potencia de diez. Si el número es mayor que 10, desplazamos la coma decimal hacia la izquierda. En cambio, si el número es menor que 1, entonces la desplazamos hacia la derecha tantos lugares como sea necesario para que el único dígito que quede a la izquierda de la coma se encuentre entre 1 y 9, y todos los otros dígitos estén a la derecha de la coma.

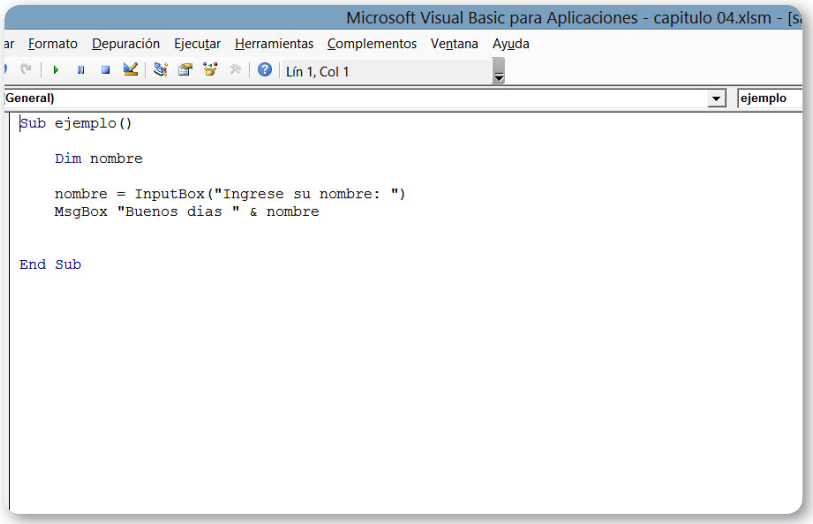


Figura 3. VBA inicializa las variables del tipo numérico con el valor **0** y de cadenas de texto con el valor (**“”**).

Option Explicit

Como vimos anteriormente, si bien podemos declarar las variables en forma implícita, esta manera no es la más conveniente, porque se pueden provocar errores en el código si nos equivocamos en el nombre que le asignamos a la variable. Para evitar estos problemas, podemos forzar la declaración de variables incluyendo la instrucción **Option Explicit** en la sección de declaraciones del módulo. Lo que hace esta instrucción es obligar a que todas las variables del módulo se declaren de manera explícita.

> DECLARAR VARIABLES DE FORMA EXPLÍCITA PUEDE GENERAR ERRORES EN EL CÓDIGO

DECLARACIÓN DE VARIABLES

Por costumbre, declaramos todas las variables al principio de cada procedimiento. Pero podemos poner las declaraciones de variables en cualquier lugar que deseemos dentro de un procedimiento VBA, siempre y cuando la instrucción **Dim** anteceda a la primera utilización de la variable en el procedimiento.

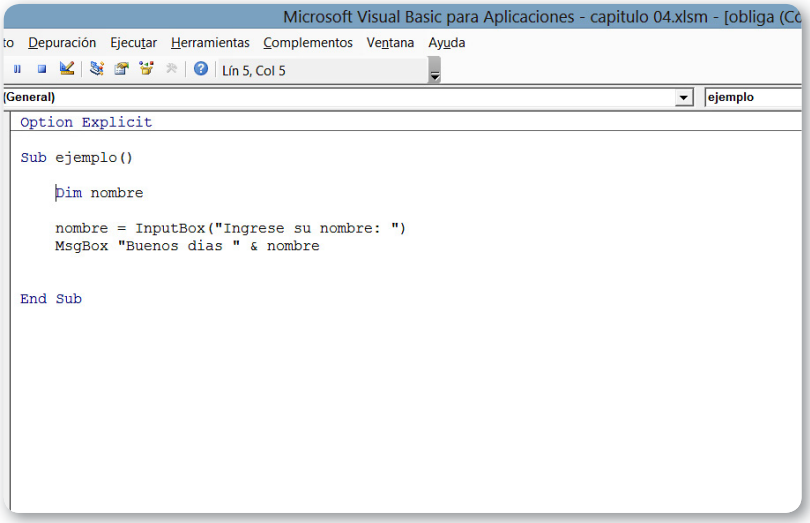


Figura 4. La instrucción **`Option Explicit`** le indica a Excel que todas las variables deben ser declaradas.

Cuando compilamos o ejecutamos un procedimiento donde empleamos esta instrucción, VBA comprobará si las variables han sido declaradas y, en el caso de no encontrar una variable no declarada, nos mostrará un mensaje de error.

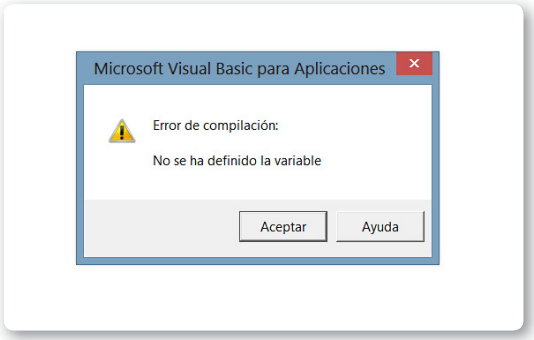


Figura 5. Cuando usamos la instrucción **`Option Explicit`** y no declaramos una variable, se producirá un error.

La instrucción la debemos escribir en la sección de declaraciones de todos los módulos en los que deseamos que VBA nos obligue a declarar explícitamente las variables. Si queremos que aparezca en forma predeterminada en todos los módulos, debemos activar la

opción **Requerir la declaración de variables**, desde el menú **Herramientas/Opciones/Editor**, como vimos en el **Capítulo 2**. De esta manera, cada vez que insertemos un nuevo módulo, aparecerá automáticamente la instrucción **Option Explicit**. Esta acción no afectará a los módulos ya existentes de nuestro proyecto.

Ámbito de las variables

El **ámbito** o **alcance** de una variable definirá en qué parte del proyecto es conocida la variable y dónde puede utilizarse, mientras que el **tiempo de vida** de una variable define la duración del valor asignado a esta.

En función del lugar donde se definen las variables, podemos distinguir los siguientes ámbitos.

Nivel de procedimiento

Las variables de nivel de procedimiento solamente estarán disponibles en el procedimiento donde las declaramos o usamos por primera vez. Fuera de ese procedimiento en particular, las variables de esta clase dejarán de existir.

Es decir, cuando declaramos una variable en un procedimiento, únicamente el código de dicho procedimiento puede tener acceso o modificar el valor de esa variable. Luego, la variable se destruye cuando finaliza el procedimiento.

Las variables locales se declaran mediantes las palabras clave **Dim**, **Static** o **Private** dentro de un procedimiento. Por ejemplo: **Dim x** o **Static x**.

NOMBRE DE LAS VARIABLES

Tengamos en cuenta que VBA recuerda el nombre de las variables a través del procedimiento, es decir, guarda un espacio de memoria con ese nombre dependiendo del tipo. Por ejemplo, si declaramos una variable llamada **gastoTotal** y posteriormente la introducimos como **gastototal**, VBA convertirá el nombre a **gastoTotal** de manera automática como parte de su comprobación sintáctica.

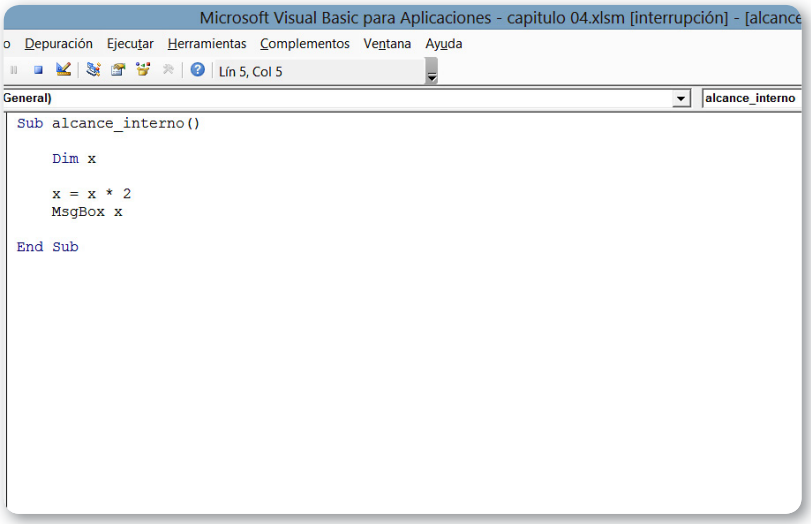


Figura 6. La variable **x** solo se utiliza en el procedimiento **alcance_interno**.

Las variables declaradas con la instrucción **Dim** solamente existirán mientras se ejecuta el procedimiento, es decir, se reinicializan cuando finaliza el procedimiento. En cambio, los valores de las variables declaradas con la instrucción **Static** conservan sus valores entre varias llamadas al procedimiento, siempre y cuando se ejecute el código. Veamos dos ejemplos de estas declaraciones:

```
Sub acumulador_1()
   Dim x
   x = x + 1
   MsgBox x
End Sub

Sub acumulador_2()
   Static x
   x = x + 1
   MsgBox x
End Sub
```

Entendamos la diferencia entre estas dos instrucciones. Cada vez que ejecutemos el procedimiento **acumulador_1**, nos mostrará el valor **1**, dado que la variable **x** se reinicializará en **0**. En cambio, cuando ejecutemos el procedimiento **acumulador_2**, el valor de **x** se incrementará en **1**, dado que la variable **x** conserva su valor.

Nivel de módulo

Las variables del nivel de módulo se crean con una declaración de tipo **Dim** o **Private** en la sección de declaraciones generales de un módulo de código o formulario, es decir, antes del primer procedimiento del módulo. Como su nombre lo indica, están disponibles para todos los procedimientos de ese módulo.

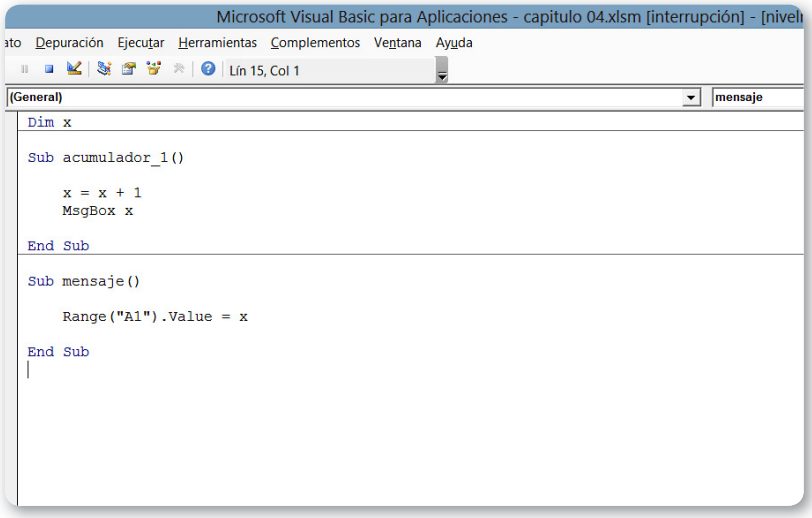


Figura 7. La variable **x** está disponible para todos los procedimientos de este módulo.

NOMBRES SIGNIFICATIVOS

Si bien podemos ponerles cualquier nombre a las variables, es recomendable que utilicemos nombres significativos, que sugieran lo que ellas representan; esto hará que el programa sea más legible y comprensible. También es una buena idea incluir breves comentarios que indiquen qué función cumple la variable.

Nivel de proyecto

Las variables del nivel de proyecto se crean con una declaración **Public** en la sección de declaraciones generales de un módulo de código o formulario. Están disponibles para todos los procedimientos del proyecto. La variable contendrá los valores hasta que se cierre el proyecto.

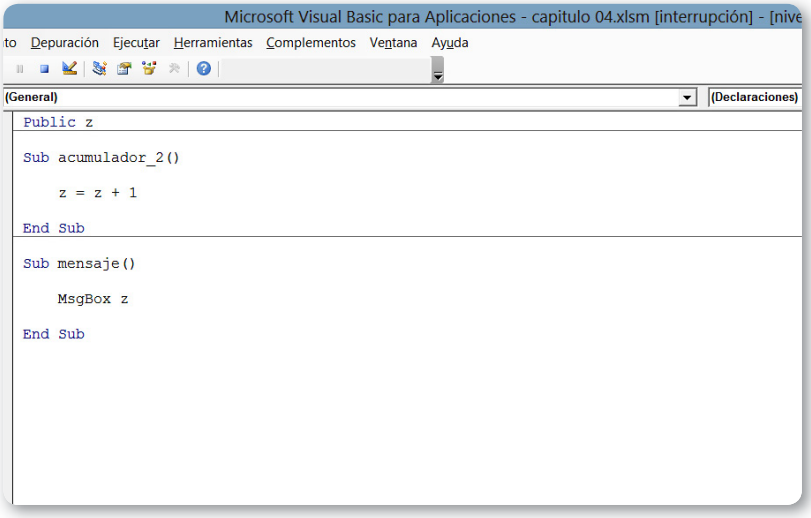


Figura 8. El nivel final de declaración de una variable se conoce como **el nivel proyecto**.

Tipos de variables

Las variables pueden almacenar diferentes tipos de datos. El tipo de dato es el que determina la naturaleza del conjunto de valores que puede tomar una variable. Todos los datos tienen un tipo asociado con

DECLARAR UNA SERIE DE VARIABLES

Visual Basic para Aplicaciones permite declarar una serie de variables empleando una sola línea de instrucción. Para ello, utilizamos la palabra clave **Dim** y, a continuación, los nombres de las variables y su tipo de dato, separados con comas. Por ejemplo: **Dim a As Integer, b As Integer**.

ellos. Un dato puede ser un simple carácter, tal como la letra **C** o un valor entero como, por ejemplo, **27**.

Cuando declaramos una variable, también podemos proporcionarle un tipo de dato. Para ello, empleamos la siguiente instrucción:

```
Dim nombre_de_variable [As tipo_de_dato]
```

La cláusula **As** permite definir el tipo de dato o de objeto de la variable que vamos a declarar. Los tipos de datos definen la clase de información que va a almacenar la variable. VBA soporta los tipos de datos que enumeramos a continuación:

- Datos numéricos.
- Datos fecha/hora.
- Datos de texto.
- Datos booleanos.
- Datos Variant.
- Datos de objeto.

Datos numéricos

VBA proporciona dos familias de datos numéricos: de **número entero** y de **coma flotante**. Como datos enteros, tenemos los siguientes tipos.

Byte

Los valores de tipo **Byte** son los números enteros comprendidos entre 0 y 255. Estos datos se almacenan en memoria como un byte (8 bits).

DIFERENCIAS

Los cálculos matemáticos exigen dos tipos de datos numéricos: enteros y reales. Como podemos esperar, las computadoras también diferencian uno y otro tipo. Si bien todos los datos se representan en memoria como cadenas binarias, la cadena binaria almacenada para el número entero **27** no es la misma que la cadena almacenada para el número real **27,0**.

Por ejemplo, si necesitamos declarar la variable **edad** como **Byte**, usamos la siguiente sentencia:

```
Dim edad As Byte
```

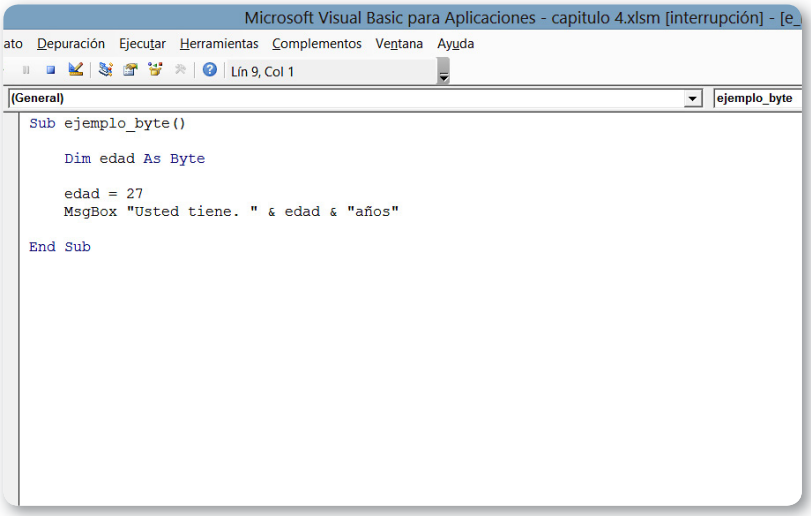


Figura 9. Las variables de tipo **Byte** son muy útiles cuando se necesita ahorrar espacio.

Integer

Las variables definidas como **Integer** pueden almacenar números enteros en el rango de valores de -32768 a 32767. Los enteros se almacenan en memoria como 2 bytes. Por ejemplo, para declarar la variable **cantidad** como **Integer**, usamos la siguiente sentencia:

```
Dim cantidad As Integer
```

Long

Las variables definidas como **Long** pueden almacenar números enteros comprendidos entre -2.147.483.648 y 2.147.483.647. Por ejemplo, para declarar la variable **precio** como **Long**, usamos la siguiente sentencia:

```
Dim precio As Long
```

Como datos de **coma flotante** tenemos los siguientes tipos.

Single

Las variables definidas como **Single** pueden almacenar números reales de 32 bits con una precisión de 7 decimales. Por ejemplo, para declarar la variable **masa** como **Single**, usamos la siguiente sentencia:

```
Dim masa As Single
```

Double

Las variables definidas como **Double** pueden almacenar números reales de 64 bits con una precisión de 16 decimales que van de -1.79769313486231E308 a -4,94065645841247E-324 para valores negativos y de 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos. Por ejemplo, si queremos una variable llamada **distancia** que nos permita almacenar la distancia entre la tierra y el sol, usamos la siguiente sentencia:

```
Dim distancia As Long
```

Currency

Las variables definidas como **Currency** pueden almacenar números reales que van de -922.337.203.685.477,5808 a 922.337.203.685.477,5807. Este tipo nos permite manejar valores con una precisión de 15 dígitos a la izquierda del punto decimal y 4 dígitos a la derecha. Esta clase de datos la usamos para realizar cálculos de

ASCII

American Standard Code For Information Interchange (ASCII) es un código de caracteres de 7 bits, en el que cada bit representa un carácter único. Los valores binarios entre 0 y 31 corresponden a instrucciones, los valores entre 32 y 127 corresponden al alfabeto alfanumérico y, entre 128 y 255, a caracteres de otros idiomas y signos menos convencionales.

tipo monetario, donde necesitamos evitar errores de redondeo. Por ejemplo, si queremos declarar la variable **precio** como **Currency**, tenemos que usar la siguiente sentencia:

```
Dim precio As Currency
```

Datos fecha/hora (Date)

El tipo de dato **Date** lo empleamos para almacenar fechas y horas como un número real. Por ejemplo, para declarar la variable **envasado** como **Date**, usamos la siguiente sentencia:

```
Dim envasado As Date
```

Los valores de fecha u hora se deben incluir entre dos signos **#**, como podemos ver en el siguiente ejemplo:

```
envasado = #27/12/14#
hora= #10:20:00 AM#
```

Datos de texto (String)

El tipo de dato **String** consiste en una secuencia de caracteres contiguos de cero o más caracteres correspondientes al código ASCII. Es decir, se trata de una cadena de caracteres que puede incluir letras, números, espacios y signos de puntuación. Los datos almacenados en una variable de tipo **String** serán tratados como texto y no como números. Por ejemplo, si necesitamos declarar la variable **apellido** como **String**, usamos la siguiente sentencia:

FECHAS

La unidad básica de tiempo es el día. Cada día está representado por un número de serie. El número de serie 1 representa la fecha 1/1/1900. Cuando introducimos una fecha, esta se almacena como el número que representa los días que hay entre la fecha inicial y la indicada. Por ejemplo, el día 28 de octubre de 1996 se representa con el número de serie 35366.

```
Dim apellido As String
```

De forma predetermina, en Visual Basic para Aplicaciones, las variables de tipo **String** son de **longitud variable**, es decir, la cadena crece y disminuye según le asignemos nuevos datos. Aunque también podemos declarar cadenas de **longitud fija** que nos permiten indicar un número específico de caracteres. Para declararlas, se utiliza el asterisco (*) seguido del número de caracteres. Por ejemplo, si queremos declarar la variable **dirección** con una longitud fija de 45 caracteres, usamos la siguiente sentencia:

> POR DEFECTO, EN VBA LAS VARIABLES DE TIPO STRING SON DE LONGITUD VARIABLE

```
Dim direccion As String * 45
```

Entonces, cuando le asignamos una cadena con menos de 45 caracteres, la variable **dirección** se rellenará con espacios en blanco hasta completar los 45 caracteres. En cambio, si le asignamos una cadena con más de 45 caracteres, VBA la truncará.

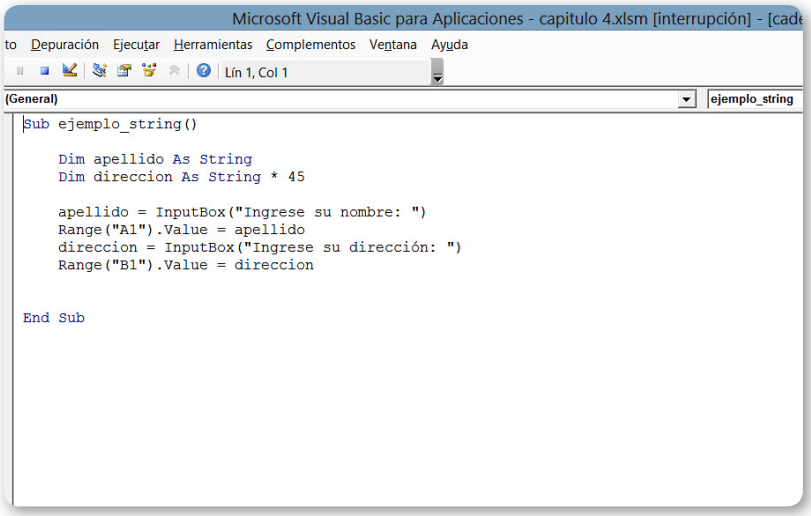


Figura 10. Podemos utilizar el sufijo **$** para establecer el tipo de variable **String** de longitud fija. Por ejemplo: **dirección$**.

Datos booleanos

Los datos **Boolean** son de tipo lógico; esto quiere decir que pueden tomar solo dos valores posibles: **True** (verdadero) y **False** (falso). El valor por defecto es **False**. Por ejemplo, para declarar la variable **ultimo** como **Boolean**, usamos la siguiente sentencia:

```
Dim ultimo As Boolean
```

Datos Variant

Los datos **Variant** pueden contener prácticamente cualquier tipo de valor. Se utilizan para declarar una variable cuyo tipo no se especifica en forma explícita. Cuando declaramos una variable sin especificar el tipo que va a contener o devolver, VBA la declara como un tipo **Variant**. La forma de declarar una variable explícitamente es la siguiente:

```
Dim mi_variable As Variant
```

Datos de objeto (Object)

Este tipo de variable permite almacenar referencias a objetos. De esta manera, podemos acceder a las propiedades de un objeto e invocar sus propiedades y métodos a través de la variable. Para declarar una variable **Object** cuando no conocemos el tipo de objeto, usamos la siguiente sentencia:

```
Dim mi_variable As Object
```

Para declarar una variable **Object** cuando conocemos el tipo de objeto, usamos la sentencia:

DATOS VARIANT

Cuando realizamos funciones u operaciones aritméticas usando variables del tipo **Variant**, el valor siempre debe ser un número. Además, si queremos concatenar cadenas, es decir, combinar dos o más cadenas de caracteres, tenemos que utilizar el operador **&** de unión, en lugar del operador **+ de suma**.

Dim nombre_variable As Tipo_Objeto

Donde **Tipo_Objeto** debe ser uno de los tipos de objetos de VBA, como por ejemplo, **Range** o **Worksheet** y no uno de los tipos de datos que vimos anteriormente. Por ejemplo, para declarar la variable **bloque** como objeto **Range**, emplearemos la siguiente sentencia:

Dim bloque As Range

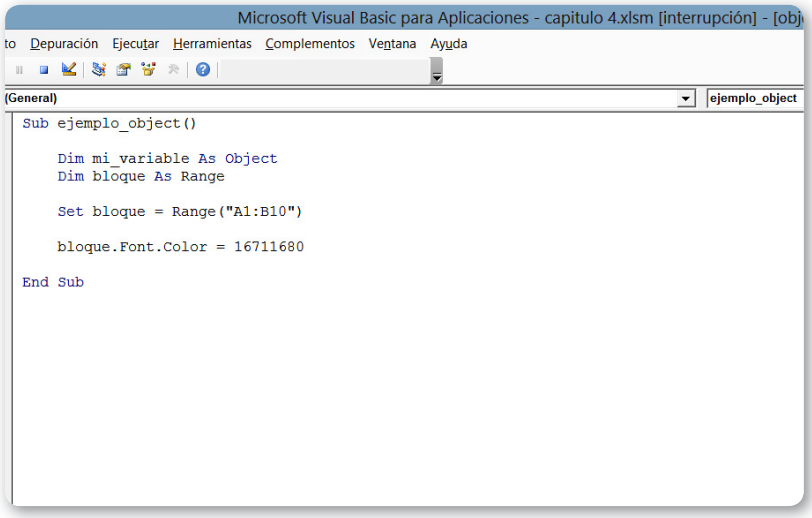


Figura 11. Las variables **Object** se declaran con las instrucciones **Dim** o **Public** en función de su alcance.

Para asignarle un objeto a una variable de tipo **Object**, usamos la instrucción **Set**. Siguiendo con el ejemplo anterior, para asignarle a la variable bloque el rango **A1:B10**, usamos la siguiente sentencia:

Set bloque = Range("A1:B10")

Tipos definidos por el usuario (UDT)

Además de los tipos de variables explicadas anteriormente, en VBA podemos definir un tipo de variable especial que agrupa variables de distintas clases. Este tipo de variables se conoce con el nombre de **variables definidas por el usuario** (**UDT**, *User Defined Type*) y

pueden contener muchas clases de datos, como, por ejemplo, **String**, **Integer**, **Currency** e incluso otros tipos definidos por nosotros.

Los datos UDT solo pueden definirse en la sección de declaración de módulo, es decir, fuera de cualquier procedimiento **Sub**. Para esto, utilizamos la siguiente sentencia:

```
Type nombre_tipo_dato
  Variable_1 As tipo_datos
  Variable_1 As tipo_datos
  Variable_1 As tipo_datos
  ......
End Type
```

Supongamos que queremos definir un tipo de datos llamado **Empresa**, que contenga los datos: **r_social**, **direccion**, **telefono** y **localidad**. Para esto, escribimos la siguiente sentencia:

```
Type Empresa
   r_social As String * 45
   direccion As String * 30
   localidad As String * 35
   telefono As Long
End Type
```

Cada vez que necesitemos una variable de este nuevo tipo, tendremos que declararla como cualquier variable, por ejemplo:

DATOS ESPECÍFICOS

Podemos convertir mediante funciones de Visual Basic para Aplicaciones valores en tipos de datos específicos, pero tengamos en cuenta que los valores que se pasan a una función de conversión deben ser válidos para el tipo de dato de destino o se producirá un error. En próximos capítulos ampliaremos el tema.

```
Dim Empresa_1 As Empresa
```

Para asignarle un valor a esa variable, especificamos el nombre de la variable, colocamos un punto (.) y, a continuación, la variable interna, por ejemplo:

```
Empresa_1.r_social = "Construcciones VMZ S. A"
```

Las constantes

Las **constantes**, como su nombre lo indica, son objetos invariables, es decir, mantienen su valor durante toda la ejecución de la aplicación. Se utilizan, por lo general, para almacenar valores que reaparecen una y otra vez o para almacenar números que resulten difíciles de recordar.

Tienen igual alcance que una declaración de variable, podemos declarar constantes a nivel procedimiento, módulo o proyecto. La sintaxis para declarar una constante es la siguiente:

> LAS CONSTANTES MANTIENEN SU VALOR DURANTE TODA LA EJECUCIÓN DE LA APLICACIÓN

```
[Public | Private] Const nombre_constante [As tipo] = expresión
```

Donde:

- **nombre_constante**: es un nombre simbólico válido para la constante (las reglas son las mismas que para las variables).

INICIALIZAR

Inicializar una variable consiste en darle un valor tan pronto como la variable ha sido declarada. Para esto, escribimos el nombre de la variable; a continuación, ingresamos el operador de asignación **=** (símbolo igual) seguido por el valor deseado. Por ejemplo, a la variable **cant**, le asignamos el valor 30 de la siguiente manera: **cant = 30**

- **expresión**: puede estar compuesta por constantes y operadores de cadena o numéricos.

Por ejemplo, podemos declarar la constante **espacio** que guarde la distancia entre la tierra y el sol. Para ello, usamos la sentencia:

```
Const espacio As Long = 149600000
```

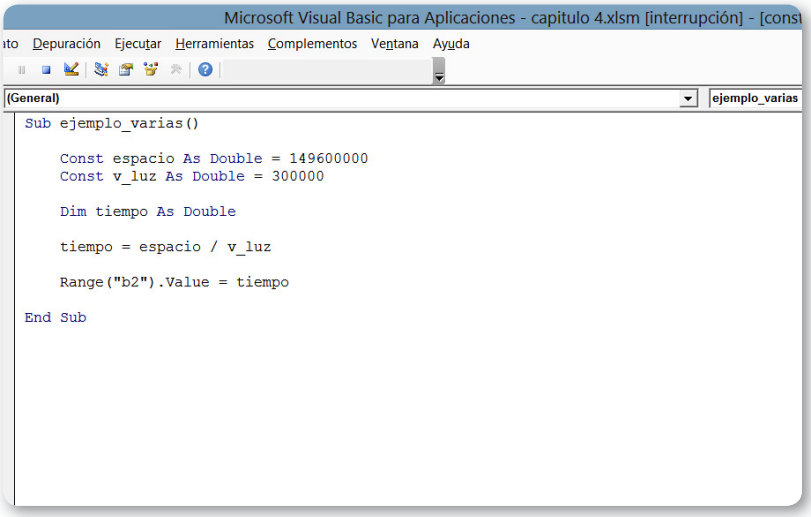


Figura 12. Los caracteres sufijos se pueden aplicar también a la declaración de variables, por ejemplo: **espacio#**.

Los operadores

Las variables y constantes se pueden procesar utilizando operaciones y funciones adecuadas a sus tipos de datos. Una **operación** es un conjunto de datos o funciones unidas por **operadores**, ya sea para modificar un valor o para producir un nuevo valor. Los operadores soportados por VBA son:

- **Operadores aritméticos**: **+** (suma), **–** (resta), ***** (multiplicación), **/** (división), **** (división entera), **Mod** (Resto), **^** (exponencial), **&** (concatenación).

- **Operadores comparativos**: **=** (igual), **<** (menor), **<=** (menor o igual), **>** (mayor), **>=** (mayor o igual), **<>** (distinto).
- **Operadores lógicos**: **Not** (negación lógica), **And** (conjunción lógica), **Or** (disyunción lógica).

Operadores aritméticos

Los operadores aritméticos son empleados para realizar las operaciones matemáticas, como suma, resta, multiplicación, división. A continuación, veremos cada uno en detalle.

Suma, resta, multiplicación, exponenciación

El operador **+** (más) se utiliza para añadir un valor a otro y también cuando necesitamos concatenar cadenas.

El operador **–** (menos) se usa para realizar la diferencia entre dos o más números y para presentar un número como valor negativo.

El operador ***** (multiplicación) se emplea para añadir un número a sí mismo una cantidad determinada de veces asignadas por otro número.

El operador **^** (exponencial) lo utilizamos para elevar un número a la potencia de otro número.

División

La división se utiliza para obtener la fracción de un número en términos de otra. VBA proporciona dos tipos de operaciones para la división sobre la base del resultado que deseamos obtener:

-**División entera**: cuando queremos que el resultado sea un número entero, utilizamos el operador ****. Los operandos pueden ser cualquier tipo de número válido, con partes decimales o sin ellas.

VALORES POR DEFECTO

Cuando declaramos una variable, esta se inicializa con un valor por defecto dependiendo del tipo de dato de la variable. Las variables del tipo numérico y fecha toman el valor **0**; las de tipo **Boolean**, el valor **False** o **0**; las de tipo **String**, la cadena vacía (**“”**) y las de tipo **Variant**, el valor **Empty**.

- **División decimal**: cuando deseamos que el resultado de la operación sea un número decimal, empleamos el operador **/**.

Por ejemplo, si tenemos dos variables **a** y **b**, siendo **a = 9** y **b = 4**, el resultado de **a / b** será **2,25**, en cambio, el resultado de **a \ b** es igual a **2**.

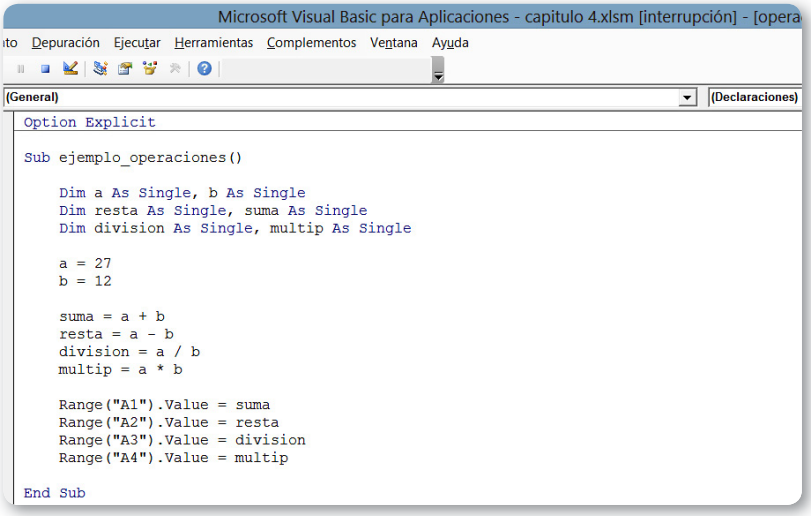


Figura 13. Tengamos en cuenta que, si el operando (**b**) es **0**, el resultado de **a/b** provocará un error.

Mod

Como sabemos, la operación de división da un resultado de un número con o sin valores decimales. A veces, necesitamos calcular el resto de dicha división, para esto utilizamos el operador **Mod**.

Por ejemplo, si tenemos dos variables **a** y **b**, siendo **a = 7** y **b = 2**, el resultado de la operación **7 Mod 2** es igual a **1**.

EXPRESIONES BOOLEANAS

Las expresiones boolenas (lógicas) son denominadas así en honor del matemático británico **George Boole** (1815-1864), uno de los fundadores de la lógica matemática. Estas expresiones se utilizan para analizar, seleccionar y procesar datos que se introducen en el componente de aplicación.

Concatenación

Para colocar dos o más cadenas juntas, utilizamos el operador **&**, que une las cadenas en el orden en que las coloquemos. Por ejemplo, podríamos tener, en la variable **nom**, el nombre de una persona y, en la variable **apell**, el apellido; para obtener el nombre completo de esta persona concatenando dichas variables, procedemos de esta manera:

```
Sub ejemplo_concatenar()

    Dim nom As String, apell As String
    Dim nombyapell As String

    nom = "Christian"
    apell = "Ballesteros"

    nombyapell = nom & " " & apell

    MsgBox ("Hola: ") & nombyapell
End Sub
```

Operadores comparativos

Los operadores comparativos, llamados también relacionales, comparan valores en expresiones, generando como resultado un valor lógico: verdadero o falso. Se utilizan respetando el siguiente formato:

variable_1 operador _relacional variable_1

constante_1 operador_relacional constante_2

TIPOS DE VARIABLES

Es conveniente que declaremos las variables con el tipo adecuado, es decir, que especifiquemos el tipo de la variable cada vez que la declaremos. Si no lo hacemos, VBA le asignará por defecto el tipo Variant y, aunque este sirva para todo, no es el recomendable para muchos casos, como para realizar bucles.

En la siguiente tabla, podemos ver diferentes ejemplos.

OPERADORES COMPARATIVOS

Variable a	Variable b	Expresión	Resultado
27	3	a > b	Verdadero
35	7	a < b	Falso
12	12	a = b	Verdadero
12	12	a <> b	Falso

Tabla 1. Podemos combinar varios operadores relacionales en una instrucción compleja.

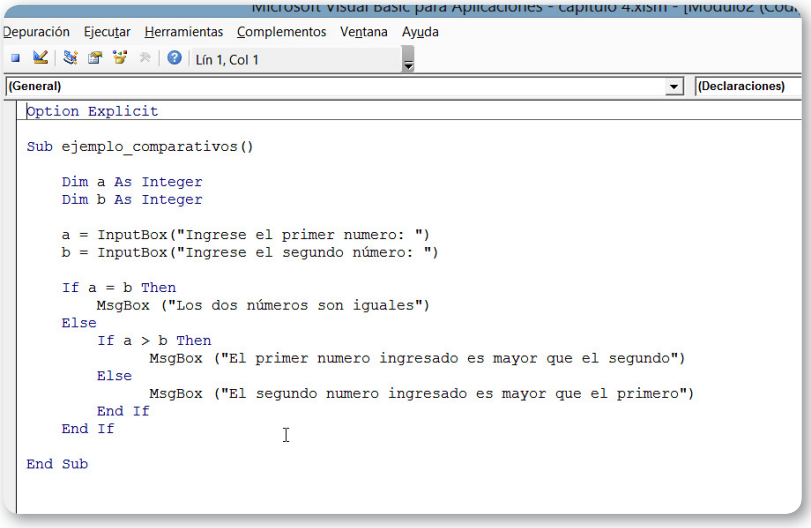


Figura 14. Por lo general, estos operadores se emplean con la instrucción condicional **`If`**.

JERARQUÍA DE LOS OPERADORES

El operador **not** tiene la prioridad más alta, seguida de los operadores multiplicativos (multiplicación, división, **mod** y **and**), de los operadores aditivos (suma, resta, **or**) y, por último, de los operadores de relación. Por tal motivo, los operadores de relación se pueden utilizar con paréntesis a fin de prevenir errores.

Operadores lógicos

Las expresiones lógicas pueden combinarse para formar expresiones más complejas utilizando los operadores lógicos **and**, **or** y **not**. Estos operadores se utilizan generalmente como condiciones.

La operación **and** (y) combina dos condiciones simples y produce un resultado verdadero, solo si los dos operandos son verdaderos. Por ejemplo: **(nota >= 6) And (asistencia >=7)**.

La operación **or** (o) es verdadera si uno de los dos operandos es verdadero. Por ejemplo: **(nota1 >=6) Or (nota2 >=4)**.

La operación **not** (no) actúa sobre una sola condición simple y el operando y niega su valor. Por ejemplo: **Not (a < b)**.

Array

Un **array**, llamado también arreglo, es una estructura en la que se almacena un conjunto de datos del mismo tipo (elementos) ordenados en forma lineal, uno a continuación de otro, con un nombre en común.

Es decir, es una lista de un número finito de **n** elementos del mismo tipo que se caracteriza por:

- Almacenar los datos del array en posiciones de memoria contigua.
- Tener un solo nombre de variable que representa a todos los elementos. Estos, a su vez, se diferencian por un índice o subíndice.
- Es posible acceder a los elementos de un array en forma aleatoria por medio de su índice.

En lo que se refiere a su dimensión, podemos decir que contamos con cinco tipos de arrays diferentes:

MATRICES

El empleo de matrices nos ayudará a crear código más pequeño y sencillo, puesto que las matrices permiten hacer referencia por el mismo nombre a una serie de variables y usar un número para distinguirla. En los próximos capítulos, explicaremos el uso de arrays mediante bucles.

- **Array de una dimensión**: son los llamados vectores o listas unidimensionales. Los elementos de un vector se almacenan en posiciones contiguas de memoria, a cada una de las cuales se puede acceder directamente.
- **Array de dos dimensiones**: son matrices o tablas. Una matriz es un array con dos índices, el primero corresponde a las filas, y el segundo, a las columnas.
- **Array multidimensionales**: son los que poseen más de 3 índices. El número máximo de índices es 64.

En cuanto al tamaño, tenemos dos tipos de array:

- **Array estáticos**: son aquellos que tienen siempre el mismo número de elementos. Este tipo de matriz la utilizamos cuando sabemos el número exacto de elementos que va a contener.
- **Array dinámicos**: son aquellos cuyo tamaño se puede cambiar en cualquier momento. Este tipo de matriz la empleamos cuando no conocemos con exactitud el número de elementos que va a contener.

Declaración de array

Un array se declara como cualquier otra variable. Para esto, utilizamos la siguiente sintaxis:

[Dim|Private|Public|Static] nombre ([dimensión]) [As tipo]

Donde:

- **nombre**: es un nombre válido que identifica al array.
- **dimensión**: es la lista de expresiones numéricas, separadas por comas, que definen las dimensiones del array. Si se omite la dimensión, el arreglo será dinámico.

UDT

Las **UDT** (variables definidas por el usuario) se asemejan a las matrices multidimensionales en las que se pueden almacenar valores relacionados mediante un nombre de variable. Pero debemos saber que las UDT se construyen a partir de diferentes tipos de datos, y las matrices deben contener el mismo tipo.

El índice del elemento de la matriz parte de 0 a menos que se use la instrucción **Option Base {0 | 1}** en la sección de declaración de módulo.

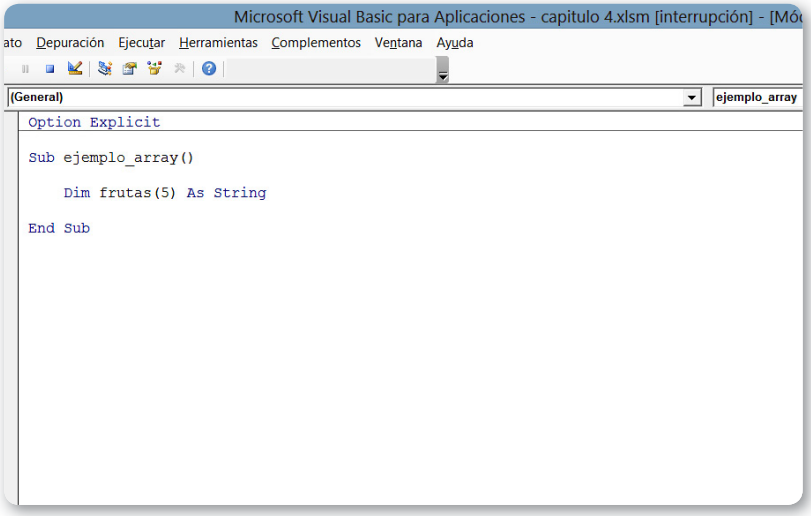


Figura 15. Los array son muy útiles para trabajar con bucles y con bases de datos.

Declarar un vector estático

Si, por ejemplo, queremos declarar un array llamado **código** para almacenar el nombre de cinco frutas, debemos escribir la sentencia que presentamos a continuación:

```
Sub ejemplo_array1()
   Dim frutas(5) As String
   frutas(0) = "Ananá"
   frutas(1) = "Pera"
   frutas(2) = "Manzana"
   frutas(3) = "Uva"
   frutas(4) = "Naranja"
End Sub
```

Observemos que, al no haber especificado la instrucción **Option Base**, el índice del vector comienza en **0**. En cambio, si especificamos

la instrucción **Option Base 1**, el índice del vector comienza en **1**, por lo tanto, el procedimiento lo deberíamos escribir de la siguiente manera:

```
Option Base 1
Sub ejemplo_array()
    Dim frutas(5) As String
    frutas(1) = "Ananá"
    frutas(2) = "Pera"
    frutas(3) = "Manzana"
    frutas(4) = "Uva"
    frutas(5) = "Naranja"
MsgBox frutas(1) & ", " & frutas(2) & ", " & frutas(3) & ", " & frutas(4) & ", " &
frutas(5)
End Sub
```

Declarar una matriz estática

En cambio, si, por ejemplo, queremos declarar una matriz llamada **x** de dos dimensiones (2 filas y 3 columnas) y asignarle unos valores, debemos escribir la siguiente sentencia:

```
Sub ejemplo_matriz()
    Dim x(3, 2) As String
    Dim i As Integer, j As Integer
    x(0, 0) = "Manzana"
    x(0, 1) = 8
    x(1, 0) = "Naranja"
    x(1, 1) = 7
    x(2, 0) = "Pomelo"
    x(2, 1) = 9
End Sub
```

Declarar vectores y matrices dinámicas

En el caso de que queramos crear un vector o una matriz dinámica, no necesitaremos especificar su tamaño cuando los declaramos. Por

ejemplo, para crear un vector dinámico llamado **datos**, simplemente escribimos la siguiente sentencia:

```
Dim datos() As String
```

Podemos cambiar el tamaño de un array, con la instrucción **ReDim**, por ejemplo:

```
ReDim datos(8)
```

Tras dimensionar un array con esta instrucción, los valores que contenía se reinicializan, tomando el valor por defecto del tipo de dato declarado.

Si deseamos conservar los valores que tenía un array dinámico antes de redimensionarlo, debemos utilizar la instrucción **Preserve** entre **Redim** y el nombre del array. En los próximos capítulos, ampliaremos el tema de vectores y matrices.

RESUMEN

En este capítulo, describimos los datos y las clases de datos utilizados por el lenguaje Visual Basic para Aplicaciones. Aprendimos a declarar las variables y a definir el tipo de variables adecuado. Vimos cómo podemos utilizar las variables y las constantes en operaciones básicas, como la suma, resta, multiplicación y división, entre otras. Explicamos cómo crear variables definidas por el usuario y, para finalizar, describimos cómo declarar array unidimensionales y multidimensionales.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es una variable?
2. ¿Para qué se emplean los tipos de variables?
3. ¿Cómo se declara una variable?
4. ¿Qué hace la instrucción **Option Explicit**?
5. ¿Qué es un **String**?
6. ¿Qué instrucción se utiliza para asignarle un objeto a una variable de tipo **Object**?
7. ¿Qué diferencia hay entre un tipo de dato definido por el usuario y un **array**?
8. ¿Cómo se declara un **Array**?
9. ¿Cómo se declara una matriz?
10. ¿Para qué se emplea la instrucción **Option Base1**?

EJERCICIOS PRÁCTICOS

1. Inserte un nuevo módulo.
2. Cámbiele la propiedad **Name** por **practico_2**.
3. Escriba el procedimiento del archivo **Cap4_actividad03.doc**, que se encuentra en **http://www.redusers.com/premium/notas_contenidos/macrosexcel2013/**.
4. Desde la ventana de VBE, ejecute el procedimiento creado.
5. Cree un botón en la hoja de cálculo y asígnele el procedimiento creado. Ejecute el procedimiento a través del botón creado.

*05

Funciones

Las funciones son uno de los elementos básicos en programación. Además de las que proporciona Excel, es posible crear funciones por medio de los procedimientos Function. Asimismo, VBA tiene funciones integradas que podemos usar dentro de las subrutinas. En este capítulo, trataremos algunas de las funciones más utilizadas.

Funciones InputBox y MsgBox

En algunas ocasiones, necesitamos interactuar con el usuario durante la ejecución de una macro, ya sea para mostrarle información o para obtener de este una respuesta. En VBA, podemos emplear los cuadros de diálogo predefinidos **InputBox** y **MsgBox** para establecer este vínculo.

InputBox

Cuando necesitamos solicitar información específica al usuario durante la ejecución de un procedimiento, utilizamos la función **InputBox**. Esta muestra en la pantalla un cuadro de diálogo modal en el cual el usuario podrá introducir la información requerida, que luego se almacenará dentro de una variable.

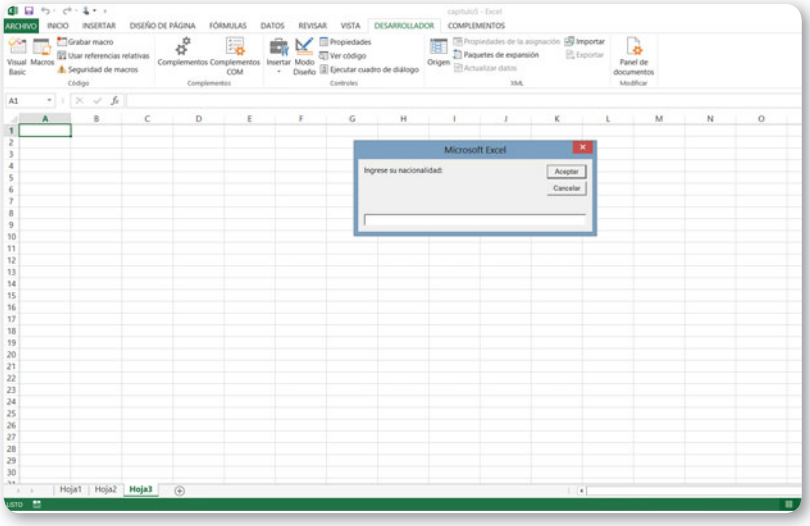


Figura 1. La función **InputBox** presenta una petición dentro de un cuadro de diálogo y devuelve lo que el usuario haya escrito.

Esta función devuelve los datos introducidos como una cadena de caracteres, si el usuario presiona el botón **Aceptar** o la tecla **ENTER**; en cambio, si presiona el botón **Cancelar** o la tecla **ESC**, esta función

devuelve una cadena de longitud cero (“”). La sintaxis completa de la función **InputBox** es la siguiente:

InputBox(promt [,title][,default][,xpost][,ypost][,helpfile, context])

Donde los argumentos se refieren a:

- **promt** (mensaje): es una cadena o variable del tipo cadena, cuyo valor es presentado por VBA en el cuadro de diálogo al momento de su ejecución. El mensaje es el único parámetro requerido que se debe pasar a esta función. Su longitud está limitada a 1.024 caracteres aproximadamente. El mensaje no tiene división de líneas y es preciso añadir explícitamente separadores de línea mediante el carácter de retorno de carro (**Chr(13)**) o un carácter de avance de línea (**Chr(10)**). El siguiente código muestra un ejemplo en el que dividimos el mensaje en dos líneas:

PARA SOLICITARLE INFORMACIÓN AL USUARIO, UTILIZAMOS LA FUNCIÓN INPUTBOX

```
Sub ejemplo_2()

   Dim rpta As String

   rpta = InputBox("Por favor " & Chr(13) & _
   "Ingrese su nacionalidad: ")
End Sub
```

MODAL

Los cuadros de diálogo predefinidos son siempre modales. Esta característica significa que los cuadros se deben cerrar antes de poder continuar trabajando con el resto de la aplicación. Por ejemplo, un cuadro de diálogo es modal si se requiere que se haga clic en el botón **Aceptar** o en el botón **Cancelar** para poder continuar ejecutando el procedimiento.

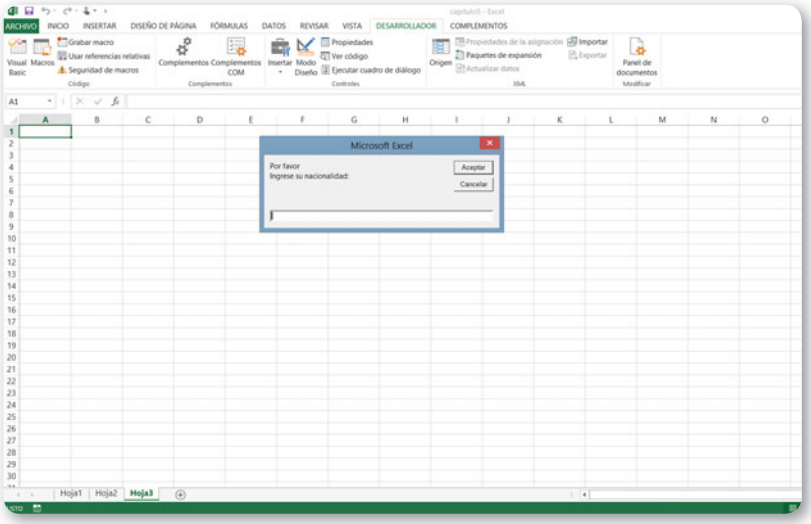


Figura 2. Cuando necesitamos dividir un mensaje en varias líneas, usamos el carácter **Chr(13)**.

- **title** (título): contiene la palabra empleada en la barra de título del **InputBox**. Es opcional; si no se utiliza, la barra de título mostrará: **Microsoft Excel**. Este es un ejemplo de cómo cambiar el título:

```
Sub cuadro_dialogo_titulo()

   Dim rpta As String

   rpta = InputBox("Ingrese su nacionalidad: ", _
   "Nacionalidad")

End Sub
```

INTERACTUAR CON LOS USUARIOS

Para interactuar con los usuarios, además de los cuadros de diálogo predefinidos de Visual Basic para Aplicaciones, podemos crear cuadros de diálogo personalizados, llamados formularios, que nos permitan mostrar o introducir datos. En el **Capítulo 9**, describiremos cómo crear formularios personalizados.

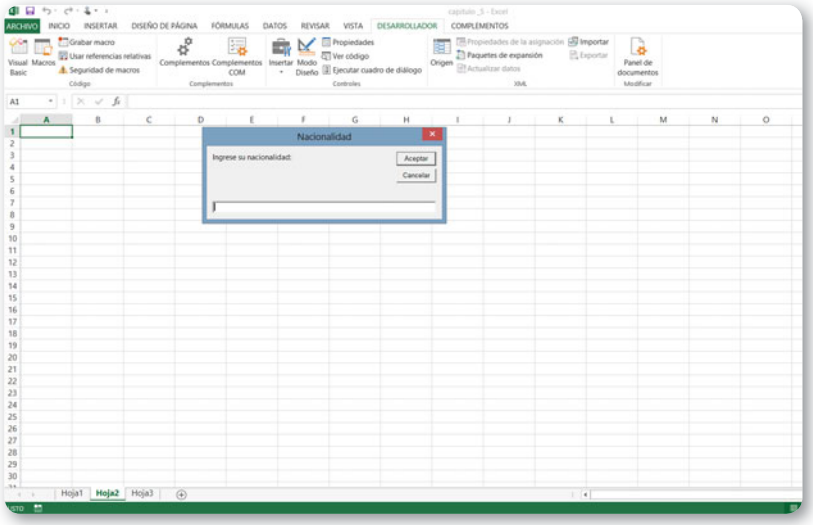


Figura 3. Para cambiar el título del cuadro de diálogo usamos el argumento **`Title`**.

- **default** (valor por omisión): también es un parámetro opcional y permite presentar un texto por defecto en el cuadro en el que el usuario introducirá la información. Si se omite, el cuadro aparecerá en blanco. Por ejemplo, supongamos que estamos pidiendo a los usuarios que ingresen su nacionalidad y sabemos que la mayoría de ellos han nacido en la Argentina, entonces podemos establecer como valor predeterminado para el cuadro de texto la palabra **Argentino**, como se muestra en el siguiente procedimiento:

EL PARÁMETRO TITLE PERMITE INGRESAR UN TÍTULO PERSONALIZADO PARA EL INPUTBOX

```
Sub cuadro_dialogo_default)

   Dim rpta As String

   rpta = InputBox("Ingrese su nacionalidad: ", _
   "Nacionalidad", "Argentino")

End Sub
```

De esta manera, el usuario no tiene que escribir nada si la información predeterminada es correcta; solamente deberá presionar la tecla **ENTER** o hacer clic en el botón **Aceptar**. Si, en cambio, la información predeterminada no es correcta, el usuario podrá sustituir el texto por defecto con su propia información.

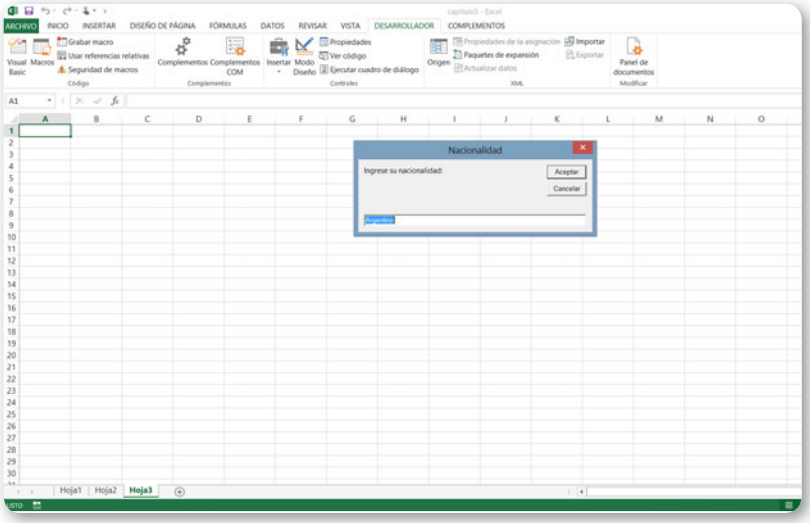


Figura 4. El dato por defecto **Argentino** es seleccionado (resaltado) de manera automática.

- **xpost** (posición x): es un número que identifica la distancia horizontal entre el lado izquierdo de la pantalla y la ventada de entrada. Si se omite este valor, la ventana aparecerá centrada horizontalmente. La distancia se expresa en **twips**. Este parámetro es opcional.
- **ypost** (posición y): es un número que identifica la distancia vertical, expresada en twips, entre el borde superior de la pantalla y la

SINTAXIS DE LA FUNCIÓN INPUTBOX

Cada argumento dentro de los paréntesis se separa con una coma. Si se omite alguno, también debe usarse la coma como separador. Por ejemplo, **respuesta = InputBox("Ingrese su nacionalidad: ", , "Argentino")**. Los argumentos deben ir en el orden correcto (mensaje, título, valor por defecto, posición horizontal, posición vertical, archivo ayuda, número de contexto para la ayuda).

ventana de entrada. Si se omite este valor, la ventana se mostrará en el primer tercio de la pantalla. Este parámetro también es opcional.

El siguiente ejemplo muestra cómo cambiar la ubicación horizontal y vertical del cuadro de diálogo:

```
Sub cuadro_dialogo_defaul()

Dim rpta As String
   rpta = InputBox("Ingrese su nacionalidad: ", _
   "Nacionalidad", "Argentino", 3000, 1500)

End Sub
```

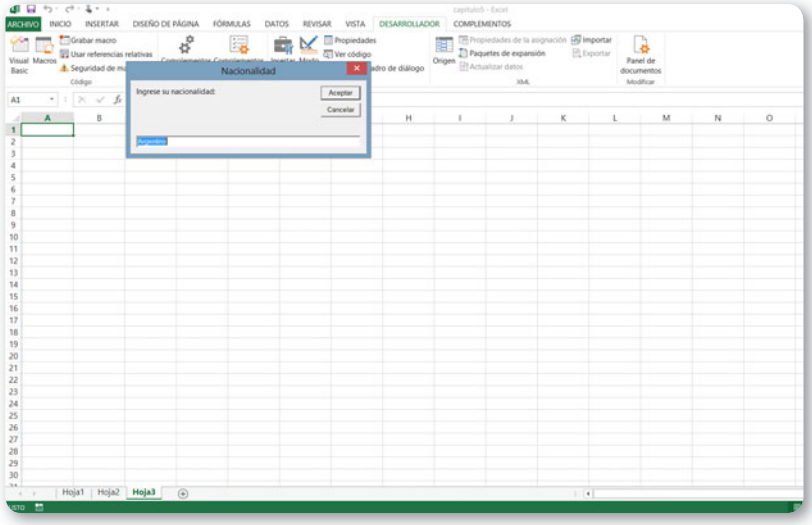


Figura 5. Para cambiar la posición del cuadro de diálogo en la ventana, usamos los argumentos **xpost**, **ypost**.

- **helpfile** (archivo ayuda): identifica el archivo de ayuda que se utilizará para apoyar la realización de esta ventana. Si se ingresa este parámetro, obligatoriamente se debe especificar el parámetro **context**.
- **context** (contexto): es el número de índice que corresponde al tema de ayuda asignado por el autor, cuya función es permitir la localización del texto de ayuda que se debe mostrar.

MsgBox

Utilizamos la función **MsgBox** cuando necesitamos obtener respuestas del tipo **sí** o **no** de los usuarios, y para mostrar mensajes breves, como errores, advertencias o alertas en un cuadro de diálogo, durante la ejecución de un procedimiento. Este cuadro de diálogo puede tener un icono e incluir hasta tres botones.

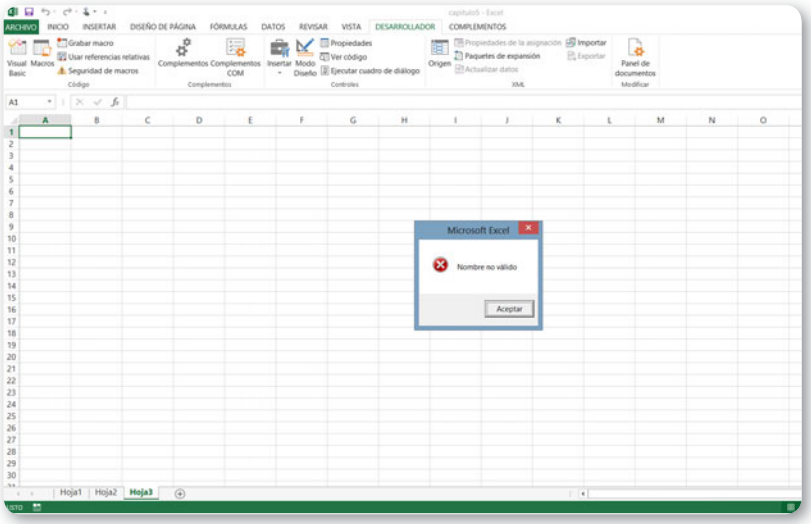


Figura 6. El **MsgBox** es una buena manera de alertar al usuario acerca de algún problema o para pedirle una respuesta del tipo **sí/no**.

La función **MsgBox** hace dos cosas: muestra un cuadro de diálogo para proporcionarle información al usuario y devuelve un valor de tipo **Integer** en función del botón que el usuario ha presionado. Si deseamos conocer el botón que ha presionado el usuario y actuar en consecuencia, debemos almacenar en una variable el valor que se genera al pulsar un botón de mensaje. La sintaxis completa de la función es la siguiente:

TWIP

Un twip es una unidad de medida de pantalla que es igual a 1/20 de punto de impresora. En una pulgada, hay 1440 twips y, en un centímetro, hay 567 twips. Es una unidad de medida utilizada en los lenguajes de programación para asegurar la ubicación y la proporción de los elementos de la pantalla de la aplicación.

MsgBox(prompt[, buttons][, title][, helpfile, context])

Donde los argumentos se refieren a:

- **promt** (mensaje): es una cadena o variable del tipo cadena, cuyo valor es presentado por VBA en el cuadro de diálogo al momento de su ejecución. Posee las mismas características que el mismo argumento de los **InputBox**. El mensaje es el único argumento requerido, aunque los botones y el título se incluyen normalmente. La siguiente sintaxis ilustra un ejemplo de un **MsgBox**:

MsgBox "La macro terminó de ejecutarse"

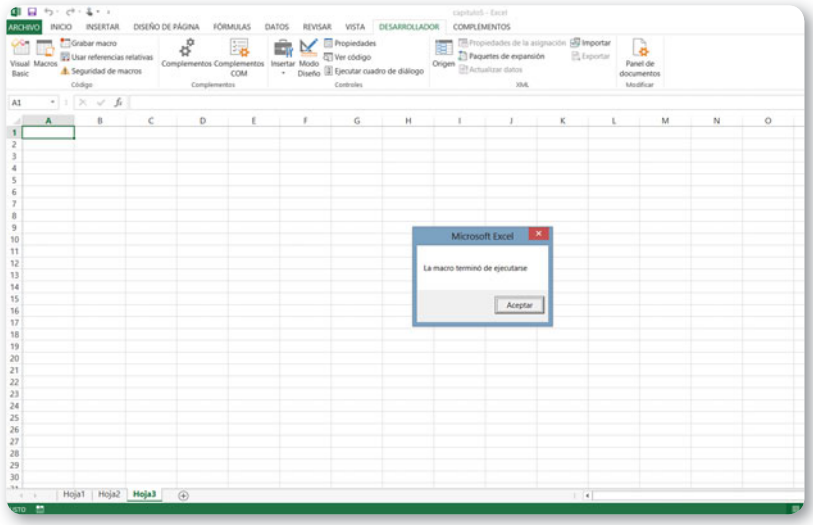


Figura 7. Un simple cuadro de mensaje que solo brinda información a los usuarios acerca de un procedimiento.

- **buttons** (botones): es la suma de valores que especifica varias propiedades, número y tipos de botones por mostrar, estilo de icono, el botón activado por defecto y la modalidad del cuadro de mensaje. Si se omite este argumento, el valor predeterminado es **0**, que corresponde al botón **Aceptar**.
- **title** (título): contiene el nombre empleado en la barra de título del cuadro de diálogo del **MsgBox**. Es un argumento opcional; si no se utiliza, la barra de título mostrará **Microsoft Excel**.

- **helpfile, context**: estos argumentos opcionales están disponibles para agregar una ayuda al cuadro de diálogo.

El argumento buttons

El argumento **buttons** nos permite mostrar distintos cuadros de mensajes de acuerdo con el valor que tome este argumento. Este valor se obtiene combinando diferentes códigos para definir el tipo de botones, para especificar el tipo de iconos, para programar el botón seleccionado por defecto e indicar el modo de ejecución. En las siguientes tablas veremos los valores que se pueden tomar:

CUADRO DE MENSAJES DEL ARGUMENTO BUTTONS

Botones por mostrar	Valor	Constante
Aceptar	0	vbOkOnly
Aceptar y cancelar	1	vbOkCancel
Anular, reintentar e ignorar	2	vbAbortRetrylgnore
Sí, no y cancelar	3	vbYesNoCancel
Sí y no	4	vbYesNo
Reintentar y cancelar	5	vbRetryCancel

Tabla 1. Valores que puede tomar el argumento **buttons**.

En el ejemplo que presentamos a continuación, utilizamos la función **MsgBox** para mostrar un mensaje en un cuadro de diálogo que contiene los botones **Sí** y **No** para que el usuario seleccione la respuesta:

```
Sub ejemplo_botones()

   MsgBox "¿Desea eliminar los datos?", _
   vbYesNo, "Información Importante"

End Sub
```

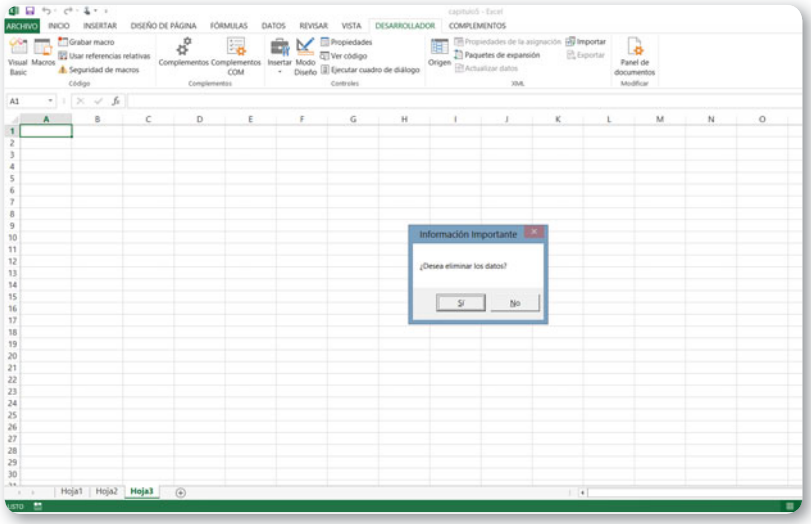


Figura 8. Solamente podremos presionar los botones **Sí** y **No** para continuar. Este mensaje no se puede cancelar con la tecla **ESC**.

De manera alternativa, en lugar de emplear la constante **vbYesNo**, podríamos utilizar el valor **4**, que es el que representa a esa constante, como podemos observar en el código que presentamos a continuación. Las dos posibilidades generan el mismo mensaje.

```
Sub ejemplo_msgbox()

   MsgBox "Nombre no válido", 4

End Sub
```

CARACTERES ESPECIALES

Para mejorar el formato de presentación, en los mensajes de cuadros de diálogo, es posible insertar algunos caracteres especiales utilizando, por ejemplo, las siguientes constantes: **vbCrlf** (retorno de carro y salto de línea), **vbCr** (salto de párrafo), **vbNullChar** (carácter nulo), **vbNullString** (cadena de longitud nula), **vbTab** (tabulación) y **vbBlack** (retroceso).

CONSTANTES DEL ARGUMENTO BUTTONS

Constante	Valor
vbCritical	16
vbQuestion	32
vbExlamation	48
vbInformation	62

Tabla 2. Otros valores que puede tomar el argumento **buttons**.

Continuando con el ejemplo que mostramos anteriormente, en esta ocasión vamos a utilizar la función **MsgBox** para presentar un mensaje en el cuadro de diálogo que no solo contiene los botones **Sí** y **No**, para que el usuario pueda seleccionar una respuesta, sino que también muestra un icono de advertencia:

```
Sub ejemplo_botones2()

    MsgBox "¿Desea eliminar los datos?", _
    vbYesNo + vbExclamation, "Información _
    Importante"

End Sub
```

BOTÓN ACTIVADO POR DEFECTO

Botón por defecto	Valor	Constante
Primer botón	0	vbDefaultButton1
Segundo botón	256	vbDefaultButton2
Tercer botón	512	vbDefaultButton3
Cuarto botón	768	vbDefaultButton4

Tabla 3. Opciones que nos permiten programar cuál será el botón predeterminado.

MODALIDAD DE UN MENSAJE

Modalidad	Valor	Constante
Aplicación modal	0	vbApplicationModal
Sistema modal	4096	vbSystemModal

Tabla 4. Opciones que nos permiten establecer la modalidad de un **MsgBox**.

PRESENTACIÓN DE UN MENSAJE

Descripción	Valor	Constante
Muestra la ventana del cuadro de mensaje en primer plano.	65536	vbMsgBoxSetForeground
Alinea el texto a la derecha.	524288	vbMsgBoxRight
Especifica el orden de lectura de derecha a izquierda para los sistemas hebreo y árabe.	1048576	vbMsgBoxRtlReading

Tabla 5. Opciones que nos permiten establecer la presentación de un **MsgBox**.

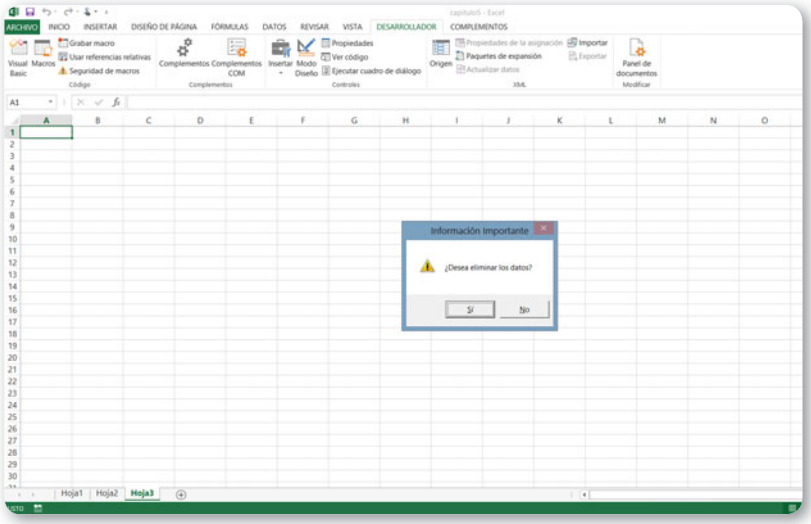


Figura 9. Ahora, tenemos un cuadro de mensaje en el que hemos agregado un icono de exclamación para llamar la atención del usuario.

Por ejemplo, si queremos alinear el texto del mensaje y del título a la derecha, usamos el siguiente código:

```
Sub ejemplo_botones2()

    MsgBox "¿Desea eliminar los datos?", _
    vbYesNo + vbExclamation + vbMsgBoxRight, _
    "Información Importante"

End Sub
```

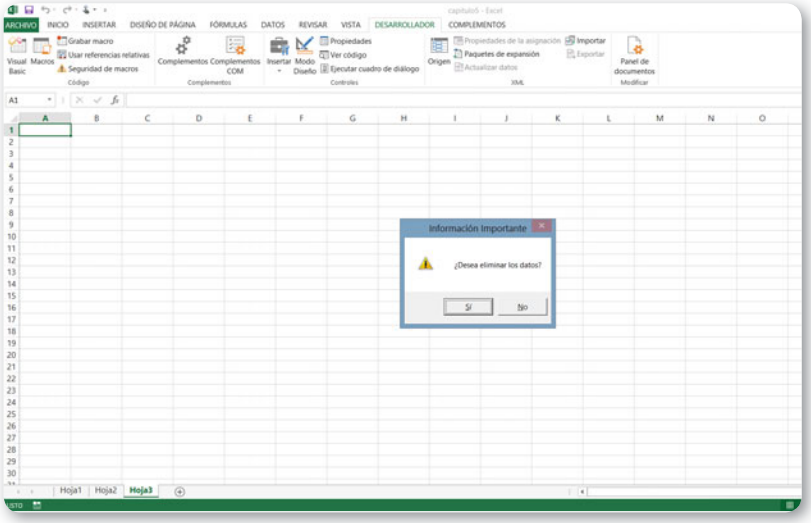


Figura 10. Hemos configurado un cuadro de mensaje cuyo texto se encuentra alineado a la derecha.

SISTEMA MODAL

Como vimos anteriormente, cuando un cuadro de diálogo **MsgBox** se establece como sistema modal, el usuario no podrá continuar el trabajo con ninguna aplicación hasta que responda el cuadro de mensaje que se le presenta. Por lo general, esta opción no es muy empleada, ya que se bloquean todas las aplicaciones hasta que el usuario responda el mensaje.

Los valores de retorno

Como hemos mencionado con anterioridad, el cuadro de diálogo **MsgBox** devuelve un número entero entre 1 y 7 dependiendo del botón que se ha presionado. Este valor puede almacenarse en una variable para luego tomar decisiones adecuadas sobre la base de la elección que han realizado los usuarios.

En la tabla que presentamos a continuación, vemos cada uno de los valores que puede devolver el cuadro de diálogo **MsgBox** de acuerdo con el botón que ha presionado el usuario.

VALORES DE RETORNO

Constante	Valor	Botón presionado
vbOk	1	Aceptar
vbCancel	2	Cancelar
vbAbort	3	Anular
vbRetry	4	Reintentar
vbIgnore	5	Ignorar
vbYes	6	Sí
vbNo	7	No

Tabla 6. Opciones que nos permiten establecer el valor de retorno de un **MsgBox**.

En el siguiente código mostramos cómo podemos utilizar el valor de retorno de la función **MsgBox**:

FUNCIONES LÓGICAS

Microsoft Excel posee un amplio conjunto de funciones lógicas. Como podemos esperar, la mayoría de ellas utiliza pruebas condicionales para determinar si una condición especificada es verdadera o falsa. La prueba condicional es una ecuación que compara dos números, funciones, fórmulas, cadenas o valores lógicos y debe incluir, al menos, un operador lógico. En el próximo capítulo, nos dedicaremos a estudiar las estructuras condicionales de VBA.

```
Sub ejemplo_mensaje10()

    Dim rpta As Integer

    rpta = MsgBox("¿Desea continuar?", 4)
    If rpta = 7 Then
        MsgBox ("Hasta pronto...")
        Exit Sub
    Else
        MsgBox ("Continuamos trabajando")
    End If

End Sub
```

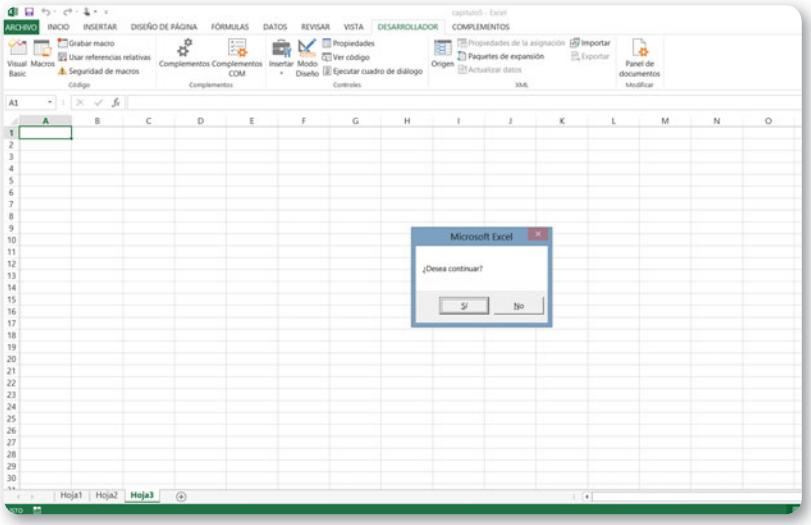


Figura 11. El cuadro de diálogo proporciona retroalimentación a la macro según el botón que ha presionado el usuario.

Cuando este procedimiento se ejecuta, muestra un cuadro de mensaje con los botones **Sí** y **No** y, luego, recibimos la respuesta del usuario en la variable **rpta**. Esta respuesta puede ser **6** o **7** en función de si el usuario hace clic en el botón **Sí** o **No**. Usamos la estructura **If ... Then** (más adelante explicaremos en detalle el uso de esta estructura)

para probar el valor de la variable **rpta**. Si el valor es **7**, significa que el usuario presionó el botón **No**, por lo tanto, se le muestra un mensaje y sale de la subrutina, pero si hace clic en el botón **Sí**, el procedimiento continúa mostrando otro mensaje en pantalla.

Funciones de conversión de tipo

Como hemos visto en el capítulo anterior, las variables pueden almacenar diferentes tipos de datos. El tipo de dato determina la naturaleza del conjunto de valores que toma una variable. Puede ocurrir que, a veces, necesitemos importar o vincular datos de fuentes externas o utilizar los datos de manera diferente al tipo de dato predeterminado. Para esto, vamos a utilizar las funciones de conversión de tipo que provee VBA. Debemos tener en cuenta que no todos los tipos de datos se pueden convertir a cualquier otro tipo de datos. Cuando usamos una función de conversión de tipo, esta devuelve el valor convertido, pero no cambia el valor almacenado.

> NO TODOS LOS TIPOS DE DATOS SE PUEDEN CONVERTIR A CUALQUIER OTRO TIPO DE DATO

VBA proporciona varias funciones de conversión que podemos usar para convertir valores en tipos de datos específicos. La sintaxis de las funciones es la siguiente: **Nombre_funcion(argumento)**. Donde:

- **Nombre_funcion**: determina el tipo de retorno.
- **argumento**: puede ser un valor variable, constante o una expresión.

A continuación, describiremos cada una de las funciones de conversión que podemos utilizar en VBA.

Función CBool

La función **CBool** convierte una expresión numérica en un valor booleano. Los valores cero los convierte en **Falso** (**False**) y todos los demás

en **Verdadero** (**True**). Si la expresión es una cadena de caracteres alfabéticos, se obtiene un error de coincidencia de tipos. Esta función es ideal para usarla en las sentencias condicionales. Su sintaxis es **CBool(argumento)** .

Por ejemplo, las siguientes expresiones devuelven **Falso** porque cada argumento se evalúa como cero (**0**):

```
X= CBool("0")

X= CBool(27-27)
```

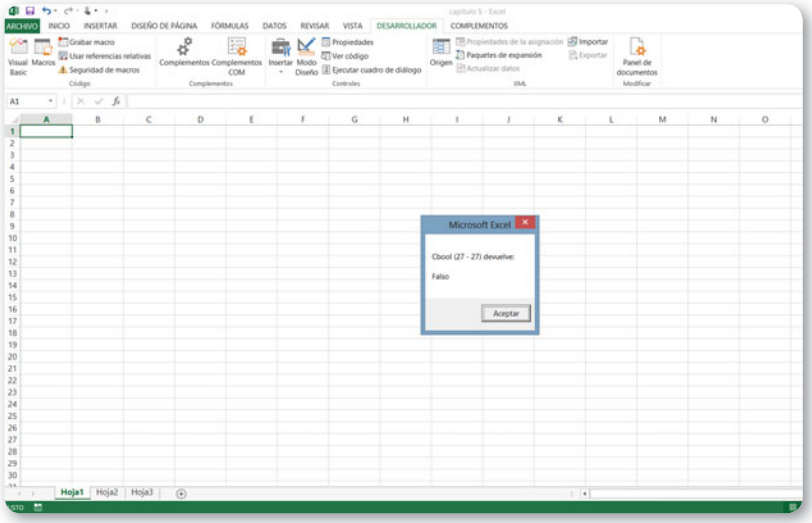


Figura 12. Un valor booleano puede ser **Verdadero** o **Falso**. El valor **Falso** es el número o el carácter **0**.

Las siguientes expresiones devuelven **Verdadero**, porque todos los argumentos se evalúan como un valor distinto a cero (**0**):

```
X= CBool("1")

X= CBool(27-3)
```

La siguiente expresión devuelve error, porque se utilizan como argumento caracteres alfabéticos:

```
X= CBool("abc")
```

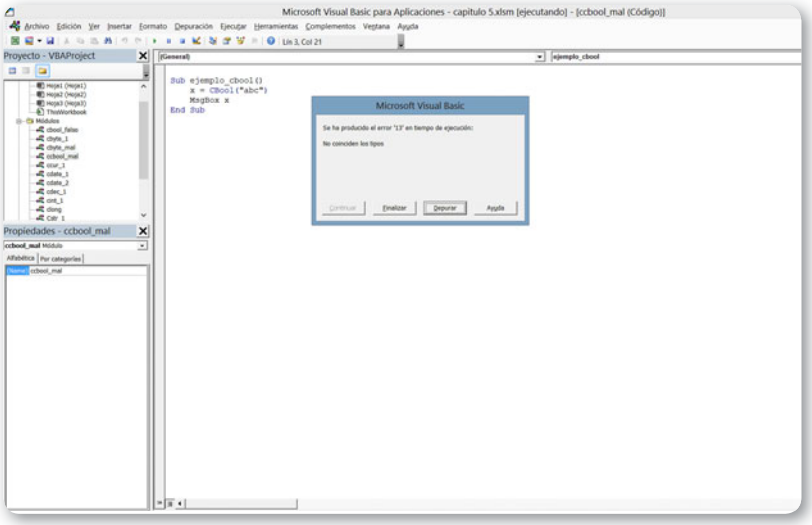


Figura 13. Si utilizamos como argumento caracteres alfabéticos, se obtiene un error de coincidencia de tipos.

Función CByte

La función **CByte** convierte una expresión numérica en byte. Esta función tiene algunas restricciones que es importante tener en cuenta. El valor de la expresión se redondea siempre hacia arriba, no debe exceder de 255 ni tampoco puede ser un número negativo.

Su sintaxis es **CByte(argumento)**. Por ejemplo, la siguiente expresión, que devuelve **21**:

```
x = CByte(21.267)
```

LA FUNCIÓN CBYTE CONVIERTE UNA EXPRESIÓN DE TIPO NUMÉRICA EN BYTE

FUNCIONES DE CONVERSIÓN REPETIDAS

En VBA, existen funciones de conversión aparentemente repetidas como **str()**, **cstr()**, **dbl()**, **cdbl()**, entre otras. Las que no tienen la **C** delante son más antiguas y trabajan en el formato nativo del Basic, el inglés americano, donde, por ejemplo, el separador decimal es el punto. En cambio, las funciones precedidas por la letra **C** son más actuales y usan la configuración regional del sistema.

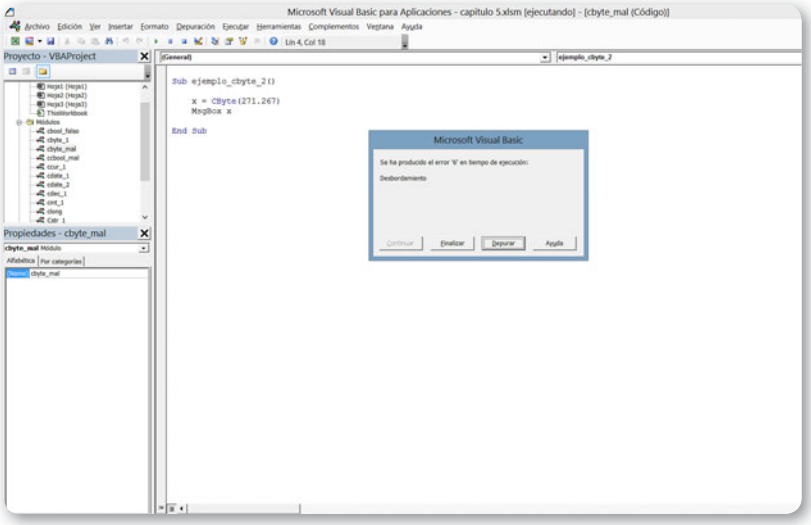


Figura 14. Si la expresión queda fuera del intervalo apropiado para el subtipo **Byte**, se produce un error.

Función CCur

La función **CCur** convierte una expresión numérica al tipo moneda. El resultado se redondea a cuatro dígitos después del punto decimal. Su sintaxis es **CCur(argumento)**. Este ejemplo devuelve **271.2196**:

```
x = CCur(271.21967)
```

Función CDate

La función **CDate** convierte una expresión numérica o de cadena que tenga formato de fecha u hora a un tipo de dato **Date**. Determina válidos los formatos fecha/hora de acuerdo con la configuración regional de Windows. Su sintaxis es **CDate(argumento)**.

Si la expresión es un valor numérico, **CDate** convierte la parte entera del número de acuerdo con el número de días, desde el 30 de diciembre de 1899. Si la expresión contiene un valor decimal, se convierte a tiempo multiplicando el decimal por 24. Por ejemplo, la siguiente expresión que devuelve **02/04/1974 11:12:29 p.m.**

```
CDate(27121.967)
```

Si el argumento es un valor de cadena, la función convierte la cadena siempre que esta represente una fecha válida. Por ejemplo, la siguiente expresión que devuelve **28/10/1996**.

```
CDate("28 octubre 1996")
```

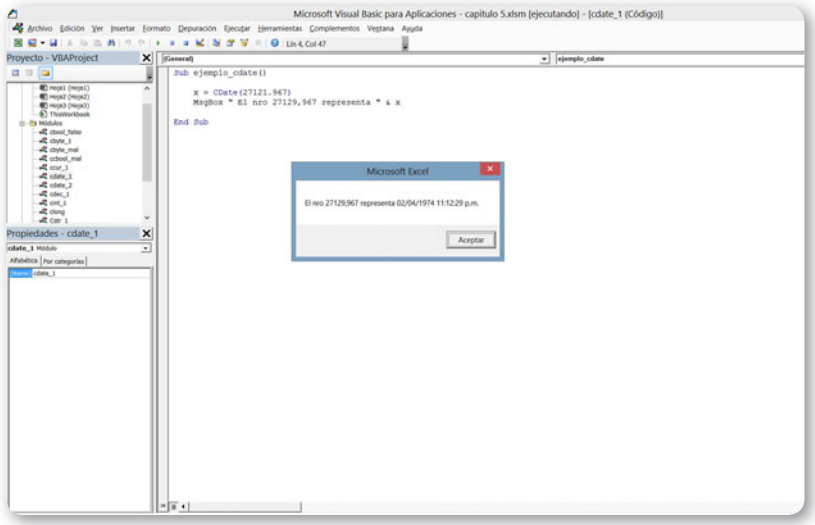


Figura 15. La función **CDate** reconoce formatos de fecha definidos en la configuración regional del sistema.

Función CDbl

La función **CDbl** convierte una expresión a un tipo de datos **Double**. La expresión debe ser un número que esté dentro del rango doble. Su sintaxis es **CDbl(argumento)**. Este ejemplo devuelve **271221,967**:

```
CDbl("27121.967")
```

UNA CONVERSIÓN EXITOSA

Los valores que pasamos a una función de conversión deben ser válidos para el tipo de dato de destino o se producirá un error. Por ejemplo, si intentamos convertir un tipo **Long** en un **Integer**, el tipo **Long** debe estar en el intervalo válido de tipo de datos **Integer**.

Función CDec

La función **CDec** convierte una expresión a un tipo de datos **Decimal**. Como vimos en los casos anteriores, esta función también tiene algunas restricciones que debemos tener en cuenta antes de su aplicación. La expresión debe ser un número o una cadena numérica que se encuentre dentro del rango decimal. Su sintaxis es **CDec(argumento)**. Por ejemplo, tenemos la siguiente expresión que devuelve el valor **271221,967**.

```
CDec(27121.967)
```

Función CInt

La función **CInt** convierte una expresión a un tipo de dato **Integer**. La expresión debe ser un número o una cadena numérica que se encuentre dentro del rango de número entero (-32678 a 32767). Si el argumento tiene un valor decimal, VBA redondea al número entero más próximo.

Su sintaxis es **CInt(argumento)**. Por ejemplo, tenemos la siguiente expresión que devuelve **271222**.

```
CInt(27121.967)
```

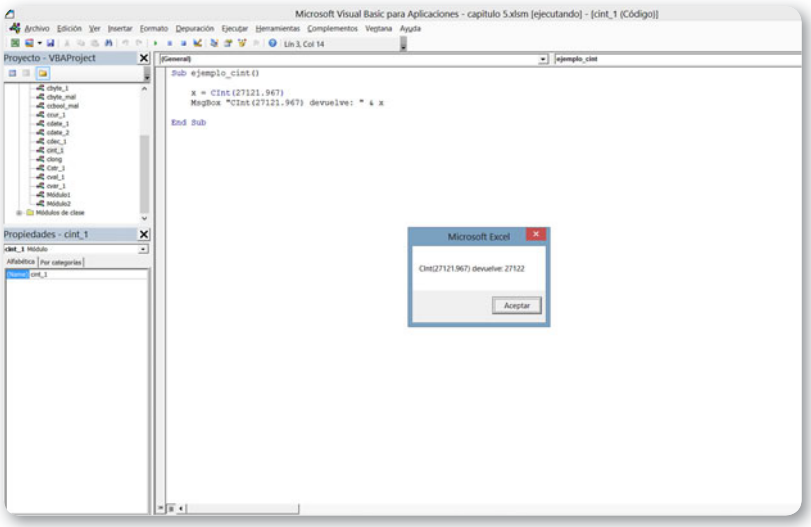


Figura 16. Los valores iguales o mayores a 0,5 se redondean hacia arriba, y los valores menores a 0,5 se redondean hacia abajo.

Función CLng

La función **CLng** convierte una expresión numérica a un entero largo. La expresión debe ser un número o cadena numérica que esté dentro del rango entero largo. Su sintaxis es **CLng(argumento)**. Por ejemplo, la siguiente expresión que devuelve **272**.

```
CLng(271.967)
```

Función CSng

La función **CSng** convierte una expresión numérica a una de simple precisión. Su sintaxis es **CSng(argumento)**. Por ejemplo, la siguiente expresión que devuelve **271967**.

```
CSng("271.967")
```

Función CStr

La función **CStr** convierte una variable de cualquier tipo de datos a una variable tipo **String** (cadena). Esta función no posee restricciones sobre las expresiones para su aplicación. Su sintaxis es **CStr(argumento)**.

Por ejemplo, en este caso, tenemos la variable **X** del tipo booleana, a la cual le asignamos el valor **True**, y entonces, la siguiente expresión devuelve la cadena **Verdadero**.

```
CStr(x)
```

LA FUNCIÓN CSTR CONVIERTE UNA VARIABLE DE CUALQUIER TIPO DE DATO EN STRING

INTERVALO DE FECHAS

Visual Basic para Aplicaciones acepta un intervalo de fechas desde el 1 de enero de 100 hasta el 31 de diciembre de 9999. En cambio Excel para Windows solo admite el intervalo de fechas entre el 1 de enero de 1990 y el 31 de diciembre de 9999, y Excel para Macintosh tiene un intervalo más pequeño que va desde el 1 de enero de 1904 al 31 de diciembre de 9999.

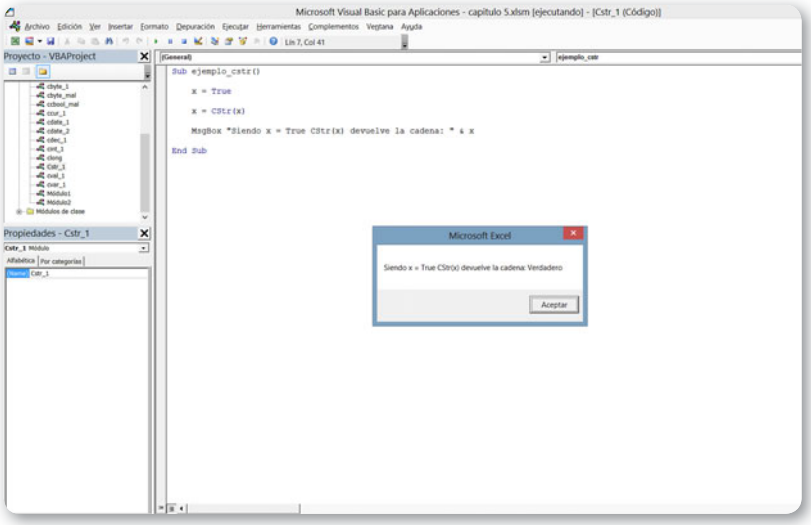


Figura 17. Empleamos la función **CStr** para transformar un valor de tipo booleano en una cadena.

Función CVar

La función **CVar** convierte cualquier variable al tipo **Variant**. Podemos usar esta función para conseguir que una expresión no numérica nos devuelva una numérica. Su sintaxis es **CVar(argumento)**. Por ejemplo, la siguiente expresión que devuelve **27**.

```
x = CVar(27)
```

Función Val

La función **Val** convierte una cadena en un número, quita los espacios en blanco de un argumento de cadena y convierte los

CONVERSIÓN CON FUNCIONES

Si bien VBA realiza la conversión implícita de tipos de datos en tiempo de ejecución entre tipos de datos compatibles, es decir, efectúa las conversiones de datos sin una sintaxis especial en el código, es preferible que nosotros hagamos las conversiones empleando las funciones de conversión de tipo para prevenir errores.

caracteres restantes en un número. Si la cadena tiene algún carácter no numérico, devuelve cero (**0**). Su sintaxis es **Val(argumento)**. Por ejemplo, la siguiente expresión que devuelve el número **-123**.

```
x = Val("-123")
```

Funciones de comprobación

Visual Basic para Aplicaciones proporciona un conjunto de funciones que nos ayudarán a comprobar o validar el tipo de datos ingresados por el usuario o los datos contenidos en las celdas. Estas funciones prueban el valor para ver si se trata de un tipo especificado.

A continuación, describiremos algunas de las funciones de comprobación.

> LAS FUNCIONES DE COMPROBACIÓN VALIDAN EL TIPO DE DATOS INGRESADO POR EL USUARIO

Función IsDate

Esta función comprueba si la expresión contiene un valor que se puede convertir a una fecha. Si el contenido representa una fecha válida, la función devuelve **Verdadero** (**True**).

Por ejemplo, el siguiente procedimiento comprueba si el contenido de la celda **A2** es una fecha. Si lo es, devuelve **Verdadero** en la celda **C2**, en caso contrario devuelve **Falso**.

```
Sub ejemplo_isdate()

   Set x = Range("A2")

   r = IsDate(x)
   Range("c2").Value = r

End Sub
```

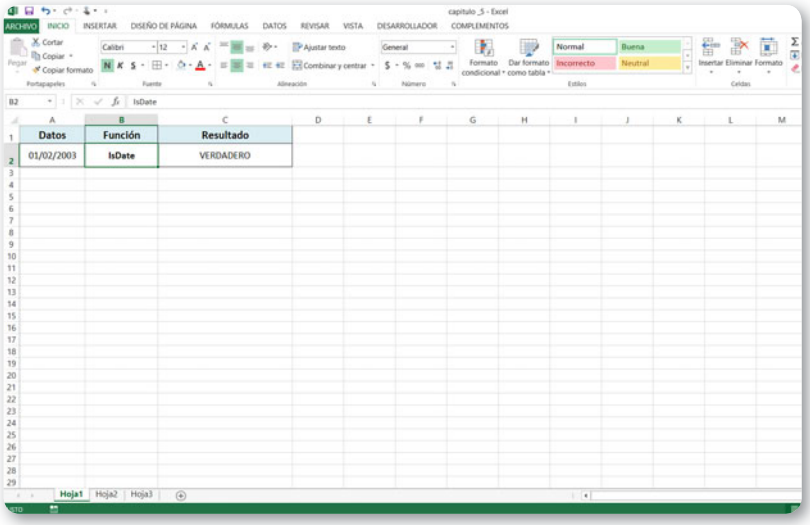


Figura 18. En este ejemplo, empleamos la función **`IsDate`** para comprobar el valor de la celda **A2**.

Función IsNumeric

Esta función comprueba si la expresión contiene un valor que se puede interpretar como número. Por ejemplo, el contenido de **A3**.

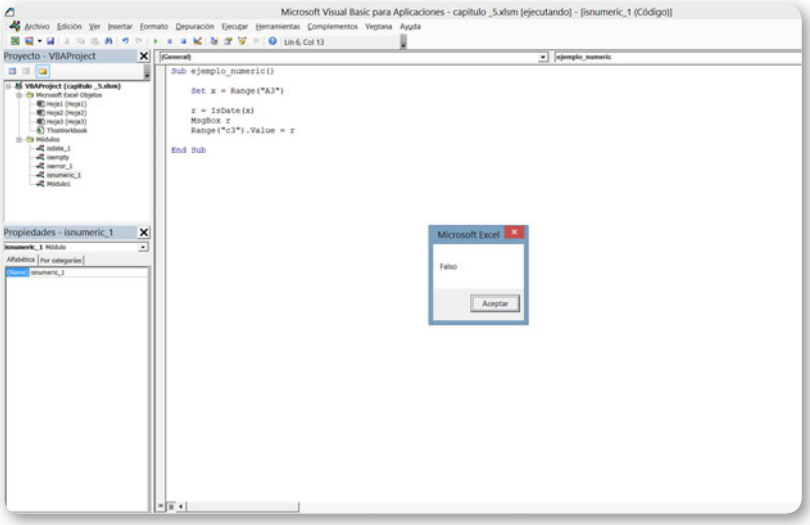


Figura 19. En este ejemplo, empleamos la función **`IsNumeric`** para comprobar el valor de la celda **A3**.

Si el contenido es numérico, devuelve **Verdadero** en la celda **C3**, en caso contrario, devuelve **Falso**.

```
Sub ejemplo_numeric()

   Set x = Range("A3")

   r = IsDate(x)
   Range("c3").Value = r

End Sub
```

Función IsNull

Esta función comprueba si la expresión contiene un valor nulo; devuelve **Verdadero** en el caso de que así sea y **Falso** en el resto de los casos. Por ejemplo, el siguiente procedimiento devuelve **Verdadero** porque la variable **x** contiene un valor nulo:

```
Sub ejemplo_isnull()

   x = Null
   x = IsNull(x)
   MsgBox x

End Sub
```

LA FUNCIÓN VAL

Una ventaja de usar la función **Val** es que, si el argumento de cadena no se puede convertir a un número, esta función siempre devolverá **0** (cero), mientras que las funciones equivalentes (**CByte**, **CCur**, **CDbl**, **CDec**, **CLng**, **CSng**) devolverían el mensaje de error: **"No coinciden los tipos"**.

Función IsEmpty

Esta función se utiliza para determinar si las variables individuales están inicializadas; devuelve **Verdadero** si esto se cumple y, en caso contrario, devuelve **Falso**. El siguiente código comprueba si en la celda **A3** de la hoja activa hay datos:

```
Sub ejemplo_empty()

   MsgBox IsEmpty(Range("A3"))

End Sub
```

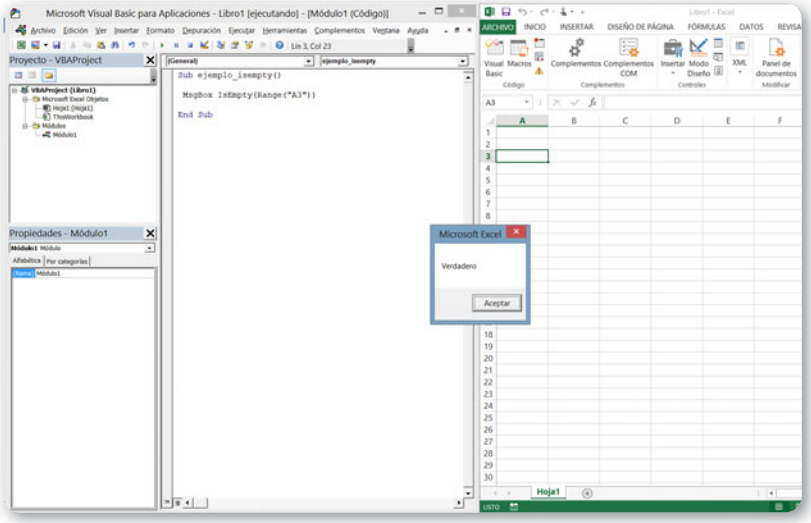


Figura 20. Como en la celda **A3** no hay datos, **`IsEmpty(Range("A3"))`** devuelve **`Verdadero`**.

VALOR EMPTY

Una variable **Variant** tiene el valor **Empty** antes de asignarle un valor. Este es un valor especial distinto de **0** (cero), una cadena de longitud cero (**""**) o el valor **Null**, que desaparece en cuanto se le asigna cualquier valor (incluyendo **0**, una cadena de longitud cero o **Null**). Para restablecer una variable **Variant** como **Empty**, se le debe asignar la palabra clave **Empty**.

Función IsObject

Esta función se utiliza para comprobar si una variable representa otra variable de tipo **Object**; si es así, devuelve **Verdadero**. El siguiente código muestra esta comprobación:

```
Sub ejemplo_isobject()

Dim y As Object
   MsgBox IsObject(y)

End Sub
```

Funciones matemáticas

Visual Basic para Aplicaciones también cuenta con funciones matemáticas propias para realizar operaciones matemáticas tales como raíz cuadrada y valor absoluto, que podemos emplear en nuestros procedimientos. A continuación, vamos a describir algunas de las funciones matemáticas más comunes.

Función Abs

La función **Abs** devuelve el valor absoluto de un número. El valor absoluto de un número es la distancia que hay entre un número y el 0, es decir, dicho número sin signo. Su sintaxis es **Abs(número)**.

Por ejemplo, tenemos la siguiente instrucción:

FUNCIONES DE EXCEL Y DE VBA

Las funciones integradas de Visual Basic para Aplicaciones no son las mismas que las de Microsoft Excel, sin embargo, podemos utilizarlas en los procedimientos de VBA, mediante el objeto **WorksheetFunction**, que está contenido en el objeto **Application**. Este objeto contiene todas las funciones de Microsoft Excel que podemos llamar desde los procedimientos de VBA.

```
Distancia = Abs(B - A)
```

Siendo **A = 35** y **B = 45**, la función devuelve **10**.

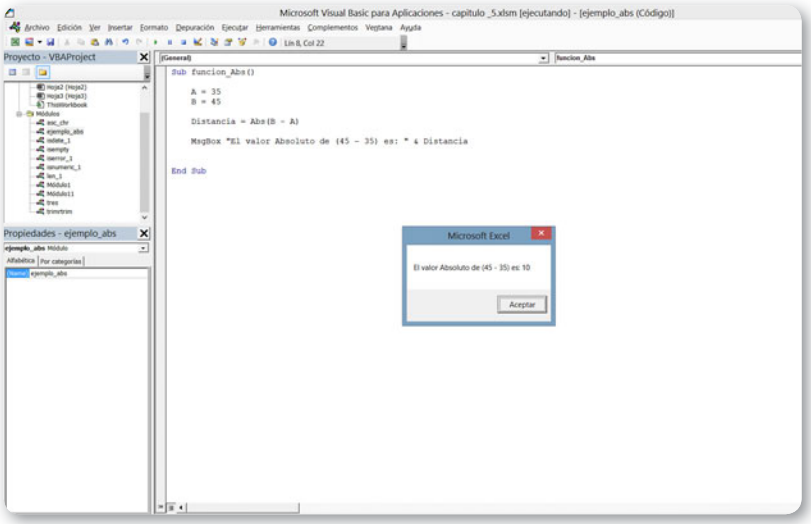


Figura 21. La función **Abs** devuelve el valor absoluto del término que se encuentra entre paréntesis.

Función Int

La función **Int** devuelve la parte entera de un número decimal. Su sintaxis es **Int(número)**. Por ejemplo, la instrucción que devuelve **27**:

```
Int(27,12)
```

Si este número es negativo, **Int** devuelve el primer entero negativo que es menor o igual que el número. Por ejemplo, la siguiente instrucción que devuelve **-28**:

```
Int(-27,12)
```

Función Fix

La función **Fix** también devuelve la parte entera de un número decimal, pero, a diferencia de la función **Int**, si el número es negativo devuelve

el primer entero negativo que es mayor de ese número. Su sintaxis es **Fix(número)**. Por ejemplo, la siguiente instrucción que devuelve **-27**:

```
Fix(-27,12)
```

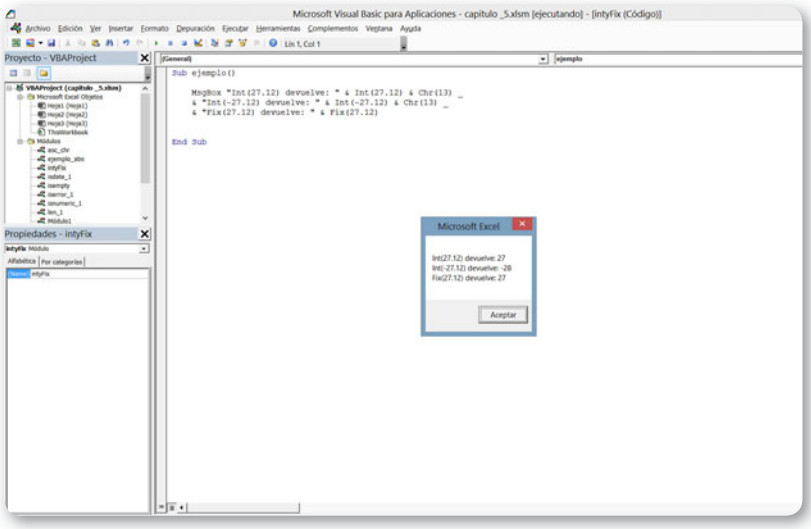


Figura 22. En este código, podemos comparar el uso de las funciones **`Int`** y **`Fix`**. Ambas devuelven la parte entera de un número decimal.

Función Rnd

La función **Rnd** genera un número aleatorio entre los valores 0 y 1. Su sintaxis es **Rnd(número)**.

El argumento **número** determina cómo la función genera el número aleatorio. Si es **0**, se obtiene el último número aleatorio generado; si es **menor que 0**, se repite el número cada vez que se ejecute el procedimiento; en cambio, si es **mayor que 0**, genera el siguiente número aleatorio en la secuencia. Esto se debe a que los números generados por la función **Rnd** son solo números pseudoaleatorios, es decir, la primera vez que se ejecuta la función, arranca la generación de números aleatorios a partir de un número denominado **semilla** y ejecuta operaciones matemáticas relacionadas con esa semilla para generar el primer número aleatorio. Para solucionar esta pseudoaleatoriedad, se debe cambiar la semilla que se usa como base

para la generación de números aleatorios, usando antes de la función **Rnd**, la instrucción **Randomize**.

Función Sqr

La función **Sqr** devuelve la raíz cuadrada de una expresión numérica, la cual debe ser un valor no negativo para evitar que se produzcan errores durante la ejecución del procedimiento. Su sintaxis es **Sqr(número)**.

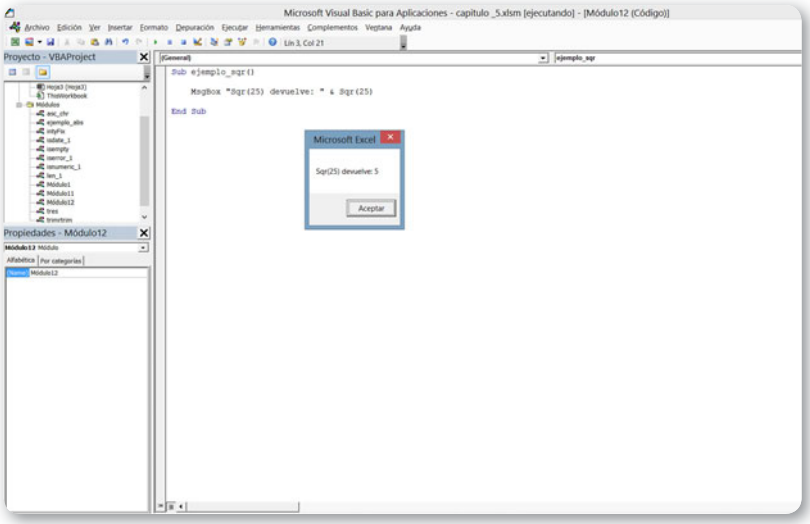


Figura 23. En este código, hemos utilizado la función **Sqr**, que devuelve **5** como resultado.

Funciones de cadenas

Visual Basic para Aplicaciones también ofrece una gran cantidad de funciones diseñadas para trabajar con las variables de cadena, que podemos utilizar, por ejemplo, para analizar una porción de una cadena, comprobar si una cadena contiene otra, sustituir partes de una cadena con otro valor y concatenar cadenas, entre otras posibilidades.

A continuación, describiremos algunas funciones de cadena más comunes que nos permitirán realizar todas estas tareas y también algunas otras de gran utilidad.

Función Asc

La función **Asc** devuelve un valor entero (**Integer**) entre 0 y 255, que representa el valor **ASCII** del primer carácter de la cadena. Es importante tener en cuenta que, si la cadena no contiene caracteres, se producirá un error cuando ejecutemos el código.

Su sintaxis es **Asc (cadena)**.

Por ejemplo, tenemos la siguiente instrucción que devuelve el número **67**:

```
Asc("Christian")
```

> SI LA CADENA NO CONTIENE CARACTERES, SE GENERARÁ UN ERROR EN EL CÓDIGO

Función Chr

La función **Chr** es la inversa de la función **Asc**, es decir, devuelve el carácter de un número. El argumento es dato de tipo **Long** que identifica a un carácter. Su sintaxis es **Chr (cadena)**.

Por ejemplo, la siguiente instrucción que devuelve el carácter **C**:

```
Chr(67)
```

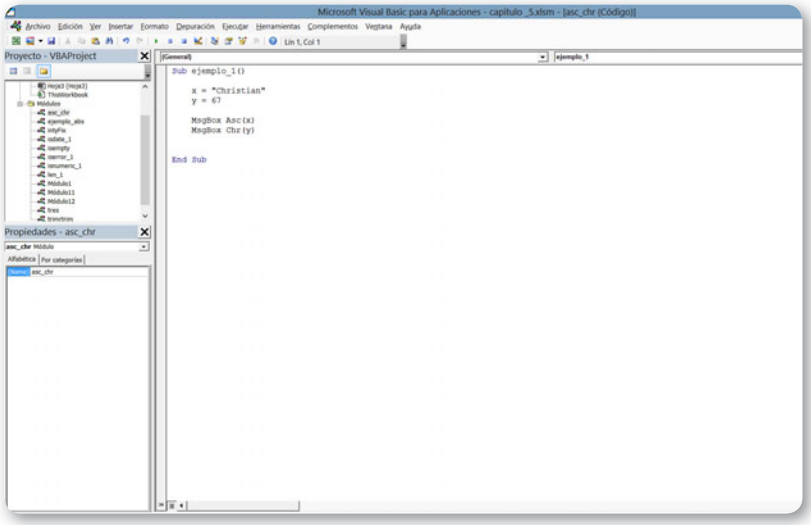


Figura 24. En este código, podemos comprobar el funcionamiento de la función **Asc** y de su inversa **Chr**.

Función Len

La función **Len** se utiliza para determinar el número de caracteres de una cadena; devuelve un entero largo, excepto cuando la cadena es nula, en cuyo caso devuelve un valor nulo. Su sintaxis es **Len(cadena)** .

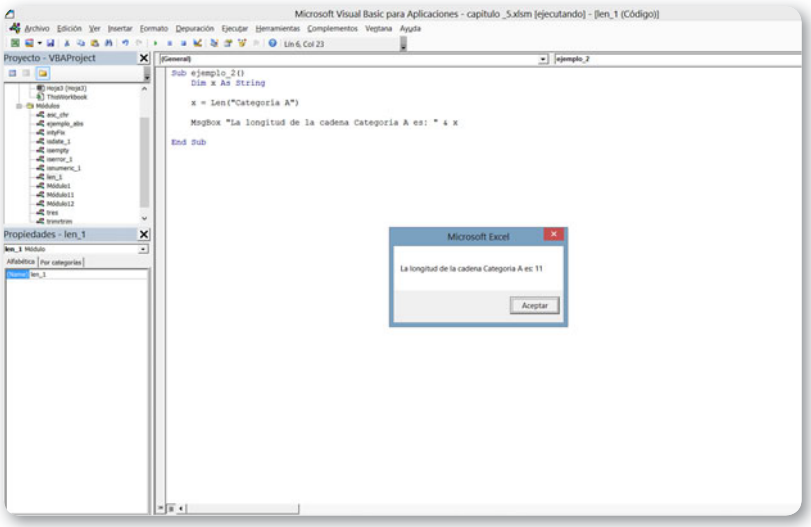


Figura 25. La función **Len** es una de las funciones de texto más utilizadas, en este ejemplo, **`Len("Categoría A")`**, devuelve **`11`**.

Función Left

La función **Left** devuelve la serie de caracteres más a la izquierda de un argumento de cadena. Su sintaxis es **Lef(cadena,nro_caracteres)**, donde **nro_caracteres** indica la cantidad de caracteres por extraer del argumento cadena. Por ejemplo, la siguiente instrucción que devuelve **Categoría**:

```
Left("Categoría A",9)
```

CINT VS VAL

Una de las ventajas de la función **CInt** con respecto a la función **Val** es que utiliza la configuración regional del sistema y, por lo tanto, reconoce el separador de miles, mientras que la función **Val**, al no reconocer la configuración regional, convierte la expresión 1000 a 1.

Función Right

Similar a la función anterior, **Right** devuelve la serie de caracteres más a la derecha de un argumento de cadena. Su sintaxis es **Right(cadena,nro_caracteres)**, donde **nro_caracteres** indica la cantidad de caracteres por extraer del argumento cadena.

Por ejemplo, la siguiente instrucción devuelve la letra **A**:

Right("Categoría A",1)

Función Mid

La función **Mid** devuelve los **n** caracteres de una cadena especificada, situados a partir de una determinada posición. Su sintaxis es **Mid(cadena, start[, lenght])**, donde:

> MID DEVUELVE LOS N CARACTERES DE UNA CADENA SITUADOS A PARTIR DE UNA POSICIÓN

- **start**: indica la posición a partir de la cual vamos a extraer los caracteres. Este número debe ser de tipo **Long** y mayor que 0. Si es mayor que el número de caracteres de la cadena, la función devuelve una cadena de longitud cero.
- **lenght**: indica el número de caracteres que queremos extraer. Su tipo es **Log**. Este parámetro es opcional; si no se encuentra, la función devolverá el resto de la cadena, empezando por la posición **start**.

Por ejemplo, la siguiente instrucción que devuelve la cadena **"gor"**:

Mid("Categoría A",5,3)

FUNCIONES TRIGONOMÉTRICAS

Las funciones trigonométricas básicas que proporciona Visual Basic para Aplicaciones son algunas de las más utilizadas en esta área: **Sin(número)** que permite calcular el seno, **Cos(número)** para calcular el coseno, **Tan(número)** que permite calcular la tangente y **Atn(número)** para calcular el arcotangente. En estos casos la única restricción es que el argumento de estas funciones siempre debe ser una expresión numérica que exprese un ángulo en radianes.

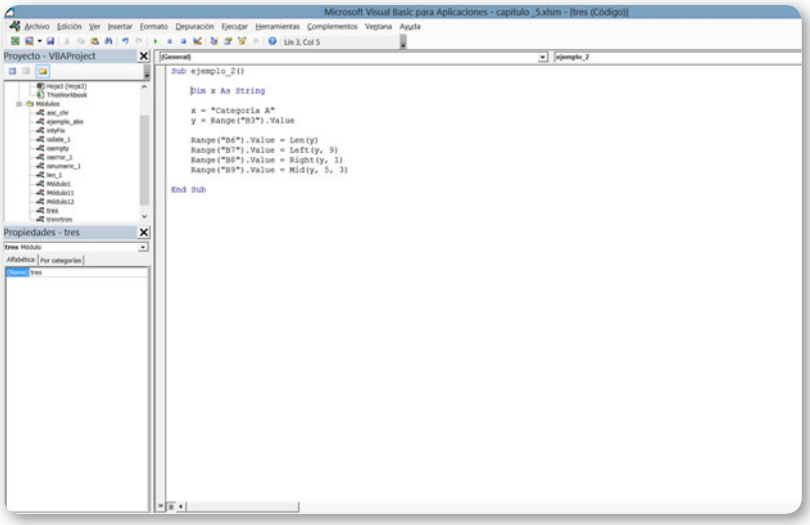


Figura 26. Las funciones **Left**, **Right** y **Mid** regresan una porción de cadena dependiendo de la función y de los argumentos proporcionados.

Función LTrim

Esta función quita los espacios iniciales de una cadena. Su sintaxis es **LTrim(cadena)**. Por ejemplo, si **x** contiene **“ Hola “** devuelve **“Hola “**.

LTrim(x)

Función RTrim

Esta función quita los espacios finales de una cadena. Su sintaxis es **LTrim(cadena)**. Por ejemplo, si **x** contiene **“ Casa “** devuelve **“ Casa“**.

RTrim(x)

USAR COMILLAS

Si el mensaje de un cuadro de diálogo es una cadena de texto, lo pondremos entre comillas. Si en cambio es una variable o una expresión, irá sin comillas. Además, recordemos que, si queremos escribir más de una línea en el mensaje del cuadro de diálogo, utilizaremos el carácter de cambio de línea **Chr(13)**.

Función Trim

Quita los espacios iniciales y finales de una cadena. Su sintaxis es **Trim(cadena)**. Por ejemplo, si **x** contiene **“ Árbol “** devuelve **“Árbol“**.

Trim(x)

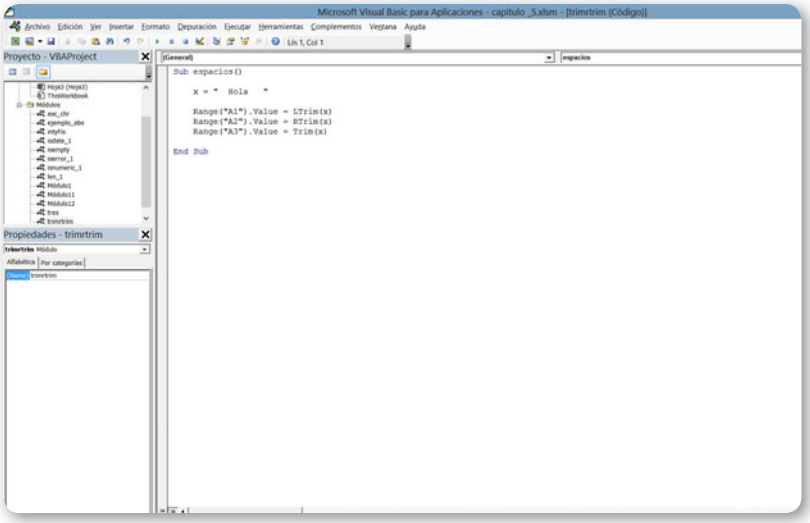


Figura 27. Las funciones **`LTrim`**, **`RTrim`** y **`Trim`** devuelven un tipo string que contiene la copia de una cadena determinada a la que se le han eliminado los espacios en blanco.

Función UCase

La función **UCase** convierte en mayúscula todos los caracteres de una cadena de texto. Su sintaxis es **UCase(cadena)**. Por ejemplo, la función **Ucase (“hola”)** devuelve la cadena **HOLA**.

OTRAS FUNCIONES

En Visual Basic para Aplicaciones también existen otras funciones de cadena, como, por ejemplo, la función **StrComp(string1, string2 [comparar]**, que devuelve un entero como resultado de la comparación de dos cadenas, y la función **StrReverse(cadena)**, que devuelve una cadena con los caracteres invertidos respecto a la cadena presentada como argumento.

Función LCase

La función **LCase** convierte en minúscula todos los caracteres de una cadena de texto. Su sintaxis es **LCase(cadena)**. Por ejemplo, la función **Ucase ("HOLA")** devuelve **hola**.

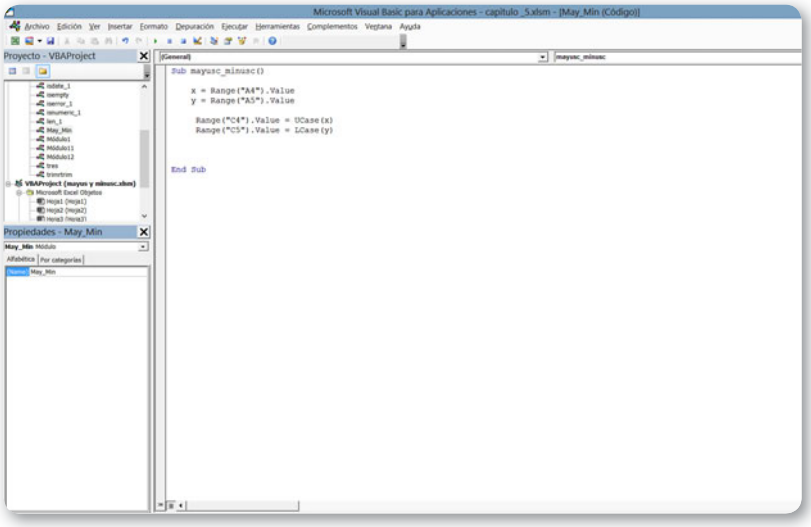


Figura 28. Las funciones **UCase** y **LCase** nos permiten pasar una cadena de caracteres a mayúsculas y minúsculas.

Función InStr

La función **InStr** devuelve la posición de una subcadena dentro de una cadena. Esta función distingue entre mayúsculas y minúsculas.

Su sintaxis es **InStr([start,] string1, string2[comparar])**, donde:

- **start**: es un parámetro opcional que indica la posición donde empieza a buscar.

FORMATO

VBA también posee la función **Format** que permite cambiar el formato de una cadena a mayúsculas y minúsculas. Por ejemplo, la **función Format("cadena",">")** devuelve la cadena convertida a mayúscula, en cambio, si usamos como cadena de formato **"<"**, nos devuelve la cadena convertida a minúscula.

- **string1**: es la cadena o expresión de cadena en la que se busca.
- **string2**: es la cadena buscada.
- **comparar**: es una constante opcional que indica que se consideran cadenas iguales. Si no se escribe, toma el valor por defecto. En la siguiente tabla, podemos ver las opciones para **comparar**.

VALORES COMPARAR

Constante VBA	Valor	Descripción
vbUseCompareOption	-1	Sigue el criterio definido en Option Compare.
vbBinaryCompare	0	Hace una comparación a nivel bytes.
vbTextCompare	1	Compara los textos.

Tabla 7. Opciones que puede tomar el argumento **comparar** de la función **InStr**.

Por ejemplo, la función **InStr(1,"Christian Germán","Germán")** devuelve **11**, porque el argumento **Germán** comienza en la onceava posición dentro de la cadena **Christian Germán**.

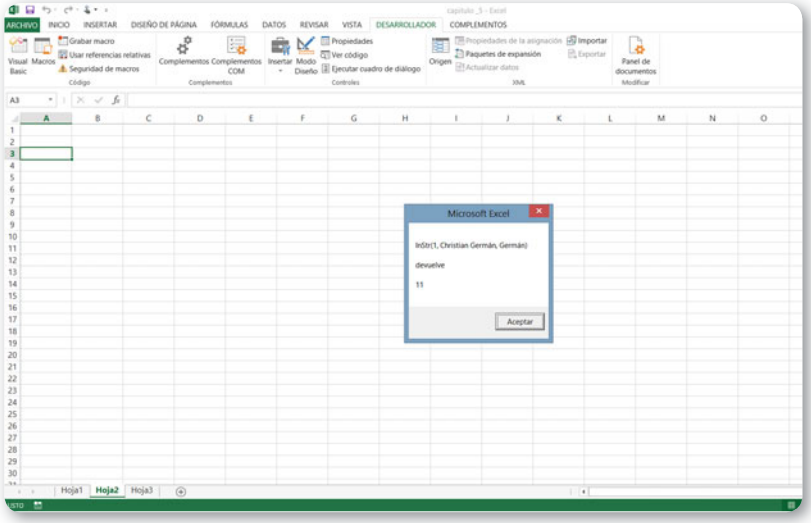


Figura 29. La función **InStr** devuelve un tipo **Variant** (**Long**), que indica la posición en la que se encuentra la subcadena buscada.

La función **InStrRev** es similar a la función **InStr**, solo que la búsqueda empieza desde el final de la cadena.

Función Replace

La función **Replace** se utiliza para reemplazar una subcadena por otra en una cadena. Su sintaxis es:

Replace(expresión, encontrar, reemplazarCon [, inicio[, contar[, comparar]]]).

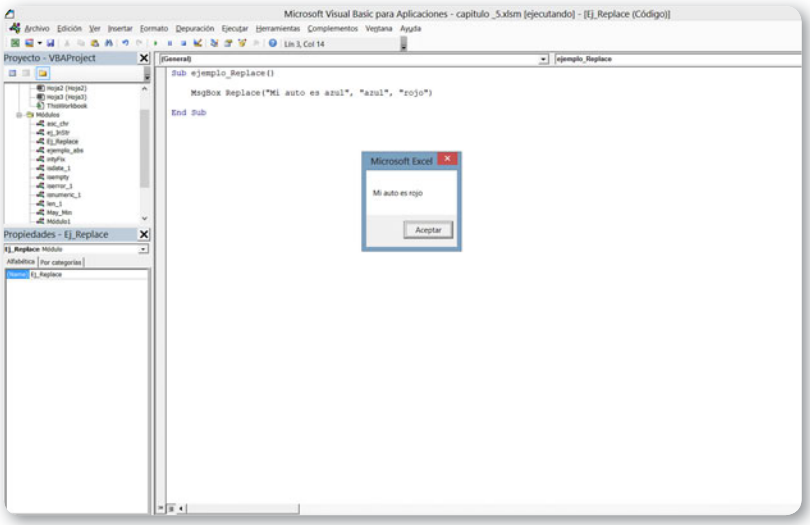


Figura 30. En este caso, **MsgBox Replace("Mi auto es azul", "azul", "rojo")** devuelve la cadena **"Mi auto es rojo"**.

Donde

- **expresión**: es la cadena o expresión de cadena en la que se busca.
- **encontrar**: es la subcadena buscada.

UNICODE

Unicode es un esquema de codificación de caracteres que utiliza 2 bytes por cada carácter. **ISO** (**Internacional Standards Organization**) define un número dentro del intervalo 0 a 65.535 por cada carácter y símbolo de cada idioma, más algunos espacios vacíos para futuras ampliaciones.

- **remplazarCon**: es la cadena que reemplazará a la anterior.
- **inicio**: indica la posición dentro del texto original (expresión) para comenzar a reemplazar
- **comparar**: como en la función **IntStr**, es la forma de comparar.

Funciones de fecha y hora

VBA ofrece varias funciones para obtener la fecha y la hora del reloj del sistema, que podemos utilizar en diferentes procedimientos. A continuación, describiremos las principales de esta clase.

Función Date

La función **Date** devuelve la fecha actual del sistema operativo de nuestra computadora en el formato que tengamos definido para la representación de fechas del sistema. Su sintaxis es **Date**.

Función Now

La función **Now** devuelve la fecha y la hora actual del sistema operativo en el formato que tengamos definido para la representación de fechas del sistema. Su sintaxis es **Now**.

Función Time

La función **Time** devuelve la hora actual del sistema operativo de nuestra computadora en el formato que tengamos definido para la representación de fechas del sistema. Su sintaxis es **Time**.

CUADRO INPUTBOX

Cuando utilizamos la función **InputBox**, tenemos pocas posibilidades de modificar los componentes del cuadro de diálogo. Solo podremos cambiar el texto de la barra de título, la solicitud de datos que se presenta al usuario, la posición del cuadro de diálogo en la pantalla y si tendrá un botón de ayuda o no.

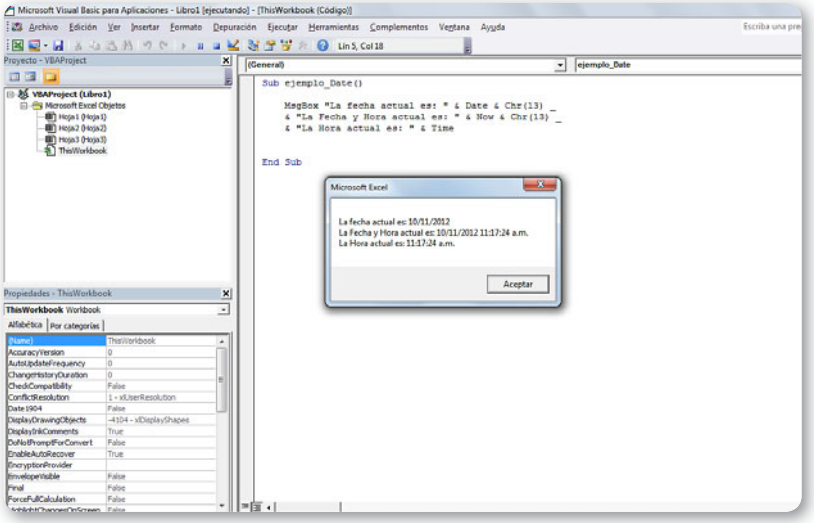


Figura 31. VBA tiene muchas funciones integradas que nos permiten trabajar con fechas y horas.

Función DateDiff

La función **DateDiff** determina la cantidad de tiempo entre dos fechas diferentes. Esta función puede devolver cualquier intervalo de tiempo entre los valores de fecha especificados, como meses, días, horas, minutos o incluso segundos. Su sintaxis es:

DateDiff(intervalo, fecha1, fecha2[, primerdíasemana[, primerasemanaaño]]).

Donde:

- **intervalo**: es el intervalo de tiempo utilizado para calcular la diferencia entre **fecha1** y **fecha2**. Hay diez diferentes valores que se pueden

FORMATO

La función **Date** devuelve la fecha en el idioma del sistema, si es en castellano es **dd/mm/aaaa**, si es inglés es **mm/dd/aaaa**. Para cambiarlo, se puede utilizar la función **Format**. Esta función devuelve una cadena formateada de acuerdo con las instrucciones contenidas en la cadena **formato**, por ejemplo, **Format(date,"dd/mm/aaaa")**

especificar para este argumento, los detallamos en la **Tabla 8**.

- **fecha1 y fecha2**: especifica las dos fechas que se utilizarán en el cálculo. Puede ser una cadena de fecha, el valor devuelto por una función o el contenido de una celda, mientras sea una fecha válida.
- **primerdíasemana**: es una constante que indica el primer día de la semana. Si se omite, se asume que es domingo. Veamos la **Tabla 9**.
- **primerasemanaaño**: determina la primera semana del año. Si se omite, considera como primera semana aquella en la que se encuentre el 1 de enero, incluso si cae en sábado. Para esto, veamos la **Tabla 10**.

VALORES INTERVALO

Intervalo	Descripción	Intervalo	Descripción
yyyy	Año	w	Día de la semana
q	Trimestre	ww	Semana
m	Mes	h	Hora
y	Día del año	n	Minuto
d	Día	s	Segundo

Tabla 8. Opciones que puede tomar el argumento intervalo de la función **`DateDiff`**.

PRIMER DÍA DE LA SEMANA (PRIMERDÍASEMANA)

Intervalo	Descripción
yyyy	Año. Solo compara el año de ambas fechas 31/12/11 y 1/1/12, devuelve el valor de un año.
q	Trimestre. Divide el año en cuatro trimestres y devuelve el número de trimestre entre fechas.

RANDOMIZE

El generador de números aleatorios comienza la generación de números aleatorios con una semilla determinada. Cada semilla genera una serie diferente de números aleatorios; eso significa que las series de números aleatorios se repitan cada vez que el procedimiento se ejecute, a menos que el generador de números aleatorios se reinicialice con un nuevo valor semilla.

m	Mes. Solo compara la porción mes de ambas fechas 31/12/11 y 1/12/12, devuelve el valor de un mes.
y	Día del año. Mismo resultado que usando d.
d	Día. Número de día entre dos fechas.
w	Día de la semana. Determina el día de la semana de la primera fecha, por ejemplo, miércoles, y cuenta la cantidad de miércoles entre las fechas.
ww	Semana. Se basa en el valor especificado como el argumento firstdayofwek para determinar el número de semana entre dos fechas.
h	Hora. Si un horario no está especificado, usa medianoche o 00:00.
n	Minuto. Número de minuto entre dos horarios.
s	Segundo. Número de segundo entre dos horarios.

Tabla 9. Opciones que puede tomar el argumento **`primerdiasemana`** de la función **`DateDiff`**.

PRIMERA SEMANA DEL AÑO (PRIMERASEMANAAÑO)

Constante	Valor	Descripción
vbUseSystem	0	Utiliza la configuración de la API de NLS.
vbFirstJan1	1	Empieza con la semana en la que se encuentra el 1 de enero (predeterminado).
vbFirstFourDays	2	Empieza con la primera semana que tenga al menos cuatro días en el nuevo año.
vbFirstFullWeek	3	Empieza con la primera semana que esté completamente incluida en el nuevo año.

Tabla 10. Opciones que puede tomar el argumento **`primerasemanaaño`** de la función **`DateDiff`**.

CUADROS DE DIÁLOGO NO MODALES

Los cuadros de diálogo no modales nos permiten cambiar el enfoque entre el cuadro de diálogo y otro formulario sin tener que cerrar el cuadro de diálogo. Es decir, podemos continuar trabajando en cualquier parte de la aplicación actual y mientras el cuadro de diálogo se encuentra presente.

Por ejemplo, el siguiente procedimiento muestra el número de meses transcurridos entre dos fechas dadas:

```
Sub ejemplo_DateDiff()

  Dim fecha1 As Date, fecha2 As Date

   fecha1 = #10/10/2012#
   fecha2 = #10/11/2012#

   meses = DateDiff("d", fecha1, fecha2)

   MsgBox "Transcurrieron: " & meses & " meses"

End Sub
```

RESUMEN

En este capítulo, conocimos las funciones InputBox y MsgBox que nos permiten interactuar con el usuario. También describimos las funciones integradas de Visual Basic para Aplicaciones (matemáticas, de comprobación, de cadenas, de fecha y hora), que podemos emplear tanto en los procedimientos Sub como en los procedimientos Function. Explicamos las funciones que permiten transformar información a un formato determinado, y otras para manipular cadenas de caracteres y para realizar cálculos matemáticos.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Cuál es la diferencia entre un **InputBox** y un **MsgBox**?
2. ¿Qué nos permite hacer el argumento **buttons** de la función **MsgBox**?
3. ¿Para qué se emplean las funciones de conversión de tipo?
4. ¿Para qué se emplea la función **CDate**?
5. ¿Para qué se emplea la función **Val**?
6. ¿Para qué se utilizan las funciones de comprobación?
7. ¿Cuál es la utilidad de la función **IsEmpty**?
8. ¿Cuál es la utilidad de la función **Rnd**?
9. ¿Cuál es la utilidad de la función **Trim**?
10. ¿Para qué se emplean las fuciones **UCase** y **LCase**?

EJERCICIOS PRÁCTICOS

1. Inserte un nuevo módulo.
2. Cámbiele la propiedad **Name** por **practico_3**.
3. Escriba el procedimiento del archivo **Cap5_actividad03.doc**, que se encuentra en el sitio **http://www.redusers.com/premium/notas_contenidos/macrosexcel2013/**.
4. Ejecute el procedimiento creado desde la ventana de VBE.
5. Cree un botón en la hoja de cálculo y asigne a este botón el procedimiento creado. Ejecute el procedimiento a través del botón creado

*06

Estructuras de programación

Las estructuras de programación permiten controlar el flujo de ejecución de un programa, es decir, el orden en que se ejecutan sus acciones individuales. En este capítulo, explicaremos las estructuras condicionales que nos permitirán tomar decisiones cuando nos encontremos con dos o más alternativas.

Estructuras condicionales

En todos los ejemplos de los procedimientos que hemos realizado hasta el momento, cada instrucción se ejecutaba una única vez en el orden en el que aparecía. Sin embargo, con frecuencia, nos encontraremos con situaciones en las que se deben proporcionar instrucciones alternativas que pueden ejecutarse o no, dependiendo de los datos de entrada, y reflejando el cumplimiento o no de una determinada condición establecida.

Las **estructuras condicionales**, llamadas también **estructuras de decisión** o **selectivas**, se utilizan para tomar decisiones lógicas. En ellas se evalúa una condición y, dependiendo del resultado obtenido, se realizan diferentes operaciones. Podemos utilizarlas para comparar tanto números como cadenas.

Entre las estructuras de decisión que acepta Visual Basic para Aplicaciones, se incluyen las siguientes:

- **If...Then**
- **If...Then...Else**
- **Select Case**

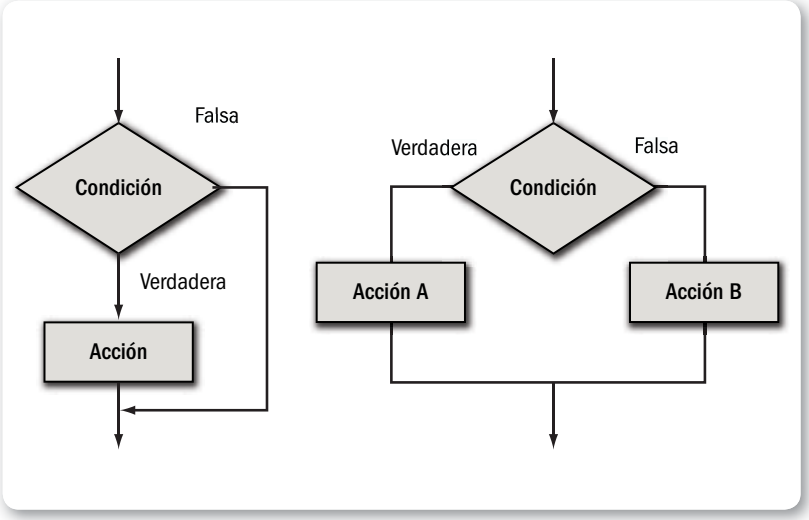


Figura 1. Diagrama de flujo de las estructuras condicionales simple y doble.

If...Then

En muchos casos, necesitaremos ejecutar una o más instrucciones **solo si una cierta condición es verdadera** y no tendremos que realizar ninguna acción si la condición es falsa, es cuando deberemos usar la estructura **If...Then** (si... entonces).

Si solo tenemos que realizar una sola instrucción en el caso de que la condición sea verdadera, usamos la sintaxis de una línea:

```
If condición Then instrucciones
```

En el caso de que necesitemos escribir varias instrucciones cuando la condición es verdadera, usamos la siguiente sintaxis:

```
If condición Then

  Instrucción_1
  Instrucción_2
  Instrucción_3
  ......

End If
```

Donde:

- **condición**: normalmente es una comparación, pero puede ser cualquier expresión numérica o de cadena que dé como resultado un valor de tipo numérico. VBA interpreta este valor como **True** o **False**. Cualquier valor numérico distinto de cero es considerado **True**, mientras que el cero es **False**, como así también una condición **Null**.

FLOWCHART

Los diagramas de flujo (**flowchart**), como, por ejemplo, el que podemos observar en la **Figura 1**, se emplean para representar gráficamente los distintos procesos que se tienen que realizar para resolver un problema, y se construyen utilizando símbolos estándares. Estos símbolos están conectados entre sí por flechas que indican el orden de ejecución de los procesos.

- **instrucciones**: es el conjunto de órdenes que debe realizar el compilador. En el formato en bloque es opcional, pero es requerido en el formato de línea que no tenga una cláusula **Else**. Si la condición es **True** y hay varias condiciones, estas se separan por dos puntos (**:**).

SI LA EXPRESIÓN LÓGICA DE LA SENTENCIA IF... THEN ES FALSA, VBA NO HACE NADA

Cuando VBA encuentra una sentencia **If...Then**, evalúa si la expresión lógica es verdadera (**True**); en caso afirmativo, ejecutará las sentencias que siguen a la cláusula **Then**. Si la expresión es falsa (**False**), entonces no hace nada.

Veamos algunos casos prácticos. Se le pide al usuario que ingrese la fecha de cierre, y, en el caso de que lo haga, entonces se ejecute el procedimiento **cierre**. Para esto escribimos el siguiente código:

```
Sub ejemplo1()

   Dim Fecha_V As Date

   Fecha_V = InputBox("Ingrese fecha de cierre:")

   If Fecha_V = Date Then cierre

End Sub
```

Ahora vamos a realizar un procedimiento que le pregunte al usuario si desea salir de la aplicación, y, si su respuesta es afirmativa, entonces se muestre un saludo y se ejecute el procedimiento **salir**.

EXPRESIONES LÓGICAS

La realización de acciones o decisiones alternativas se especifican utilizando condiciones que son verdaderas o falsas. Estas condiciones se llaman **expresiones lógicas** y se forman al comparar los valores de las expresiones mediante operadores relacionales o de comparación.

```
Sub ejemplo2()

   R = MsgBox("¿Desea salir de la aplicacion?" _
      , vbQuestion + vbYesNo)

   If R = vbYes Then
      MsgBox "Hasta Pronto"
      salir

   End If

End Sub
```

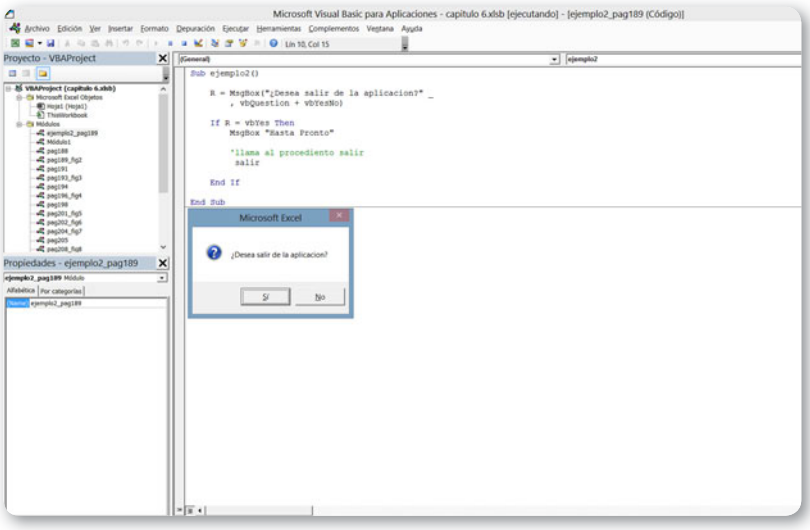


Figura 2. Si la condición es verdadera, se sale de la aplicación. En caso contrario, no se hace nada.

Hasta el momento, vimos ejemplos con la estructura **If** para evaluar una condición simple; no obstante, también es posible crear **condiciones más complejas** utilizando los operadores lógicos: **And**, **Or** y **Not**.

Estos operadores se utilizan con constantes lógicas de forma similar a la manera en que los operadores aritméticos se usan con las constantes numéricas. La operación **And** (y) combina dos condiciones

simples y produce un resultado verdadero (**True**) si los dos operandos son verdaderos. Es decir: **condición_1 And condición_2**. Por ejemplo:

```
If nro >= 0 And nro <= 100 Then MsgBox "Correcto"
```

La operación **Or** (o) es verdadera si uno de los dos operandos es verdadero. Es decir, **condición_1 Or condición_2**. Por ejemplo:

```
If edad >= 15 Or altura <= 1,60 Then MsgBox "Tenis"
```

La operación **Not** (no) actúa sobre una sola condición simple u operando, y simplemente niega (o invierte) su valor. Es decir, **condición_1 Not condición_2**. Por ejemplo:

```
If Not (cat = "A") Then Premio = sueldo * adicional
```

TABLA DE VERDAD

OPERADOR AND		
Operador 1	**Operador 2**	**Operador 1 And Operador 2**
True	True	True
True	False	False
False	False	False
OPERADOR OR		
Operador 1	**Operador 2**	**Operador 1 Or Operador 2**
True	True	True
True	False	True
False	True	True
False	False	False
OPERADOR NOT		
Operador 1	**Not Operador 2**	
True	False	
False	True	

Tabla 1. Descripción del significado de los operadores lógicos.

Por ejemplo, si el monto por pagar es mayor a 5000 y el cliente paga al contado, se le aplicará un 10% de descuento; de lo contrario, no se le aplicará nada. En este caso, tenemos dos condiciones que se deben cumplir, usamos el operador lógico **And**:

```
Sub ejemplo3()

 Dim monto As Double
 Dim f_pago As String
 Dim descuento As Double

  monto = Val(InputBox("Ingrese monto a pagar"))
  f_pago = UCase(InputBox("Ingrese forma de pago" _
  & "Contado (C) Tarjeta (T)"))

 If monto > 5000 And f_pago = "C" Then

   descuento = monto * 10 / 100
 End If

 MsgBox "Total a pagar " & monto - descuento

End Sub
```

If...Then...Else

Como vimos, la estructura **If...Then** es muy limitada, porque permite ejecutar una o más instrucciones solo si una cierta condición es verdadera, y no realiza ninguna acción en caso contrario.

ORDEN DE PRIORIDAD

Cuando la expresión lógica contiene diferentes tipos de operadores lógicos, es preciso seguir un orden de prioridad o precedencia para obtener el valor final de la expresión. El operador **Not** tiene la prioridad más alta, seguida del operador **And**, el operador **Or** y, por último, los operadores de relación.

Normalmente, necesitaremos realizar una serie de instrucciones si la condición es verdadera y otro conjunto de ellas si la condición es falsa. Para poder hacerlo, VBA cuenta con la estructura **If...Then...Else** (si... entonces...sino).

```
If condición Then

   Instrucción_1
   Instrucción_2
   Instrucción_3
   ......
Else

   Instrucción_4
   Instrucción_5
   Instrucción_6
   ......

End If
```

Esta estructura permite que el flujo del programa se bifurque por dos caminos distintos según si la condición se cumple o no. Si la condición resulta verdadera (**True**), se ejecutará el bloque de instrucciones correspondientes, y, si resulta falsa (**False**), se ejecutará el bloque de instrucciones comprendidos entre **Else** y **End If**. Cuando termina las operaciones, el ciclo del programa vuelve a la secuencia normal.

En el ejemplo que presentamos a continuación, definimos dos variables: **sexo** y **nombre**. Luego, le solicitamos al usuario que ingrese

PROMEDIO PONDERADO

El **promedio ponderado** es un tipo de promedio en el cual el resultado no surge de sumar todos los valores y dividirlos por el número total de valores, sino de asignarle un peso a cada valor para que algunos valores influyan más en el resultado que otros. Se obtiene del cociente entre la suma de los productos de cada dato por su peso y la suma de los pesos.

el nombre y el sexo de una persona. En función del último dato introducido, se escribirá: **"Hola Sra. xxxx"**, si la variable sexo es igual a **F** y **"Hola Sr. xxxx"**, si es distinta de **F**.

```
Sub ejemplo3()

Dim nombre As String
Dim sexo As String

nombre = UCase(InputBox("Ingrese su nombre: "))
sexo = UCase(InputBox("Ingrese su sexo: " & _
    "(F: femenino, M: Masculino)"))

   If sexo = "F" Then
      MsgBox "Hola Sra. " & nombre
   Else
      MsgBox "Hola Sr. " & nombre
   End If

End Sub
```

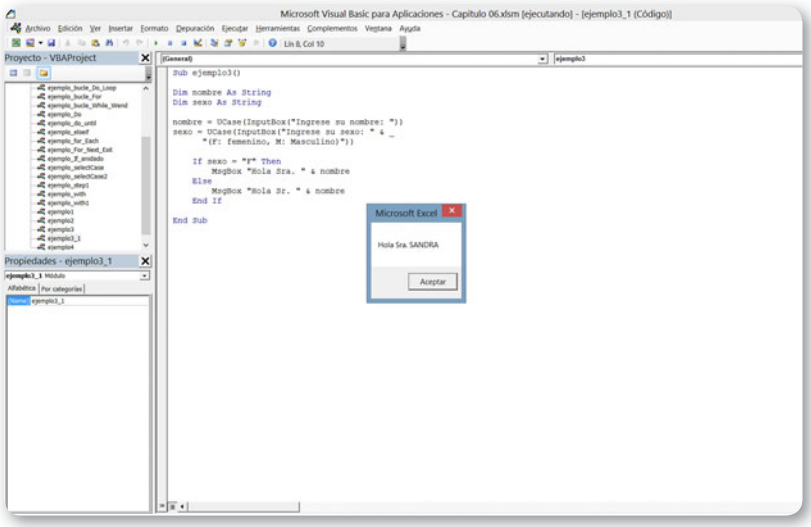


Figura 3. Usamos **`If...Then...Else`** para definir varios bloques de instrucciones, uno de los cuales se ejecutará.

Veamos otro ejemplo. Vamos a realizar un procedimiento que le solicite al usuario que ingrese tres notas de los cursos que ha realizado un alumno. El curso A tiene 2 créditos, el curso B tiene 3 créditos y el curso C tiene 4 créditos. El alumno puede llevar más de 20 créditos si su promedio ponderado fue 8 o más.

```
Sub ejemplo4()

Dim n1 As Byte, n2 As Byte, n3 As Byte
Dim P As Double


n1 = Val(InputBox("Ingrese nota curso A: "))
n2 = Val(InputBox("Ingrese nota curso B: "))
n3 = Val(InputBox("Ingrese nota curso C: "))

   P = (n1 * 2 + n2 * 3 + n3 * 4) / (2 + 3 + 4)

   If P >= 8 Then
      MsgBox "Promedio ponderado = " & P & _
      " Puede llevar más de 20 créditos)"

   Else
      MsgBox "Promedio ponderado = " & P & _
      " Sólo puede llevar hasta 20 créditos)"
   End If

End Sub
```

ELSEIF O SELECT CASE

Para que el uso de las estructuras sea algo simple y eficiente, debemos tomar en cuenta la siguiente recomendación. Si bien podemos agregar más cláusulas **ElseIf** a la estructura **If...Then**, esta sintaxis puede resultarnos confusa y molesta de escribir cuando en cada **ElseIf** comparamos la misma expresión con un valor distinto. Para estas situaciones, podemos emplear la estructura **Select Case**.

If...Then...ElseIf

La estructura **If...Then...ElseIf** es un caso especial de **If...Then...Else**. Usamos esta estructura en aquellas situaciones en que necesitamos comprobar más de una condición al mismo tiempo.

```
If condición_1 Then

  Instrucción_1
  Instrucción_2
    ......
ElseIf condición_2 Then

  Instrucción_3
  Instrucción_4
  ......
ElseIf condición_3 Then

  Instrucción_5
  Instrucción_6
  ......
......

Else
  Instrucción_7
  Instrucción_8
  ......

End If
```

SENTENCIAS DE CONTROL

Llamadas también estructuras de bifurcaciones y bucles, son las encargadas de controlar el flujo de un programa según sus requerimientos. Es decir, permitiendo tomar decisiones y realizar un proceso repetidas veces. Este tipo de estructuras son comunes en la mayoría de los lenguajes de programación, aunque su sintaxis puede variar de un lenguaje de programación a otro.

LA CONDICIÓN_1 SE EVALÚA EN PRIMER LUGAR; SI ES FALSA, SE EXAMINA LA CONDICIÓN_2

En este tipo de estructura, la **condición_1** es la que se examina en primer lugar. Si resulta verdadera (**True**), entonces se ejecutará el bloque de instrucciones correspondientes; en cambio, si es falsa, recién se evaluará la segunda condición (**condición_2**). Si la segunda condición es verdadera (**True**), se ejecutará el bloque de instrucciones **ElseIf**, y así sucesivamente.

Por último, si ninguna de las condiciones evaluadas resultan verdaderas, entonces al llegar a este punto, se deberán ejecutar las instrucciones que se encuentran comprendidas entre el bloque **Else** y **End If**.

Veamos el siguiente ejemplo en el que se le pide al usuario que ingrese tres números distintos. Dados estos números, el programa mostrará cuál de ellos es el mayor.

```
Sub ejemplo_ElseIf()

Dim n1 As Integer, n2 As Integer, n3 As Integer

n1 = Val(InputBox("Ingrese el primer nro: "))
n2 = Val(InputBox("Ingrese el segundo nro: "))
n3 = Val(InputBox("Ingrese el tercer nro: "))

   If n1 > n2 Then
      MsgBox "El nro " & n1 & _
            " es mayor que: " & n2
   ElseIf n1 > n3 Then
     MsgBox "El nro " & n1 & " es menor que: " & _
            n2 & " pero mayor que " & n3
   Else
    MsgBox "El nro " & n1 & " es menor que " & _
            n2 & " y " & n3
   End If

End Sub
```

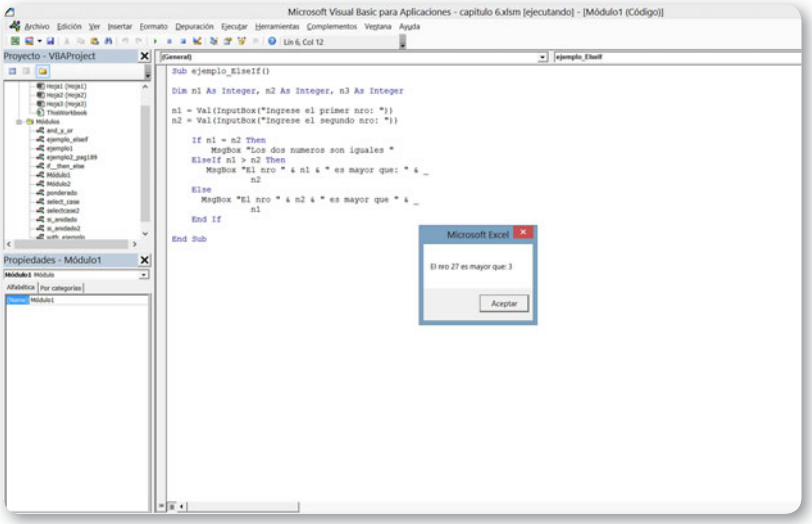


Figura 4. La cláusula **`ElseIf`** al igual que la cláusula **`Else`** son opcionales.

Estructuras If anidadas

Como hemos visto en la sintaxis de la estructura **If**, dentro del bloque de instrucciones por ejecutar podemos usar cualquier instrucción, incluso otra sentencia **If...Then...Else**. Cuando una o ambas bifurcaciones de una sentencia **If...Then...Else** contienen también una sentencia **If...Then...Else**, se dice que dichas sentencias están **anidadas**, y el proceso recibe el nombre de **anidamiento**.

Podemos usar una sentencia **If** anidada para construir decisiones con alternativas. Veamos un ejemplo. Un negocio ha programado una serie de ofertas basadas en un porcentaje de descuento sobre el total de la compra, que varía según el monto. Si es mayor o igual a 500, se hará un descuento del 30%; si es menor que 500 pero mayor o igual a 200, se

LEGIBILIDAD

Recordar que sintácticamente el sangrado y la separación de líneas no son necesarios en la escritura del código, pero favorecen la legibilidad de los procedimientos cuando estamos usando estructuras de repetición o estructuras de selección y, sobre todo, cuando anidamos estructuras.

hará un descuento del 20%; si es menor a 200 pero mayor o igual a 100, se hará un descuento del 10%, y por montos menores a 100 no se hará ningún descuento. Cada **If** lleva su cláusula **End If** correspondiente.

```
Sub ejemplo_If_anidado()

Dim importe As Single
Dim total As Single

   importe = InputBox("Ingrese Importe: ")

   If importe >= 500 Then
      total = importe - importe * 30 / 100
      MsgBox "Importe a cobrar: " & total
   Else
      If importe >= 200 And importe < 500 Then
         total = importe - importe * 20 / 100
         MsgBox "Importe a cobrar: " & total
      Else
         If importe >= 100 And importe < 200 Then
            total = importe - importe * 10 / 100
            MsgBox "Importe a cobrar: " & total
         Else
            MsgBox "Importe a cobrar: " & importe
         End If
      End If
   End If

End Sub
```

ESCRIBIR EXPRESIONES CASE

Cuando escribamos las expresiones de los **Case**, debemos hacerlo de la forma más sencilla posible, evitando intervalos muy extraños y teniendo cuidado con los operadores relacionales para no dejar afuera ningún valor posible. Si se nos olvida la palabra clave **Is**, VBA la escribirá por nosotros.

Estructuras Select Case

Cuando tenemos muchas condiciones diferentes, puede ser difícil utilizar varios **If** anidados; por esta razón, VBA ofrece la estructura **Select Case** como alternativa de la estructura **If...Then...Else**.

Con la estructura **Select Case**, se evalúa una expresión que puede tomar un número indeterminado de valores y realizar acciones según el valor de esta. Su sintaxis es la siguiente:

```
  Select Case expresión
Case Valor1
        Instrucción_1
        Instrucción_2
        ......
    Case Valor2
        Instrucción_1
        Instrucción_2
        ......
    Case Valor3
        Instrucción_1
        Instrucción_2
        ......

    .......

    Case Else
        Instrucción_1
        Instrucción_2
        ......


  End Select
```

La estructura **Select Case** funciona con una única expresión de prueba que se evalúa una sola vez al principio de la estructura. Visual Basic para Aplicaciones compara el resultado de esta expresión con los valores de cada **Case** de la estructura y, si hay coincidencia, ejecuta el bloque de instrucciones asociados a ese **Case**.

La palabra clave **Case**, normalmente, va seguida de una lista de expresiones de uno o más valores. Si hay más de un valor en una lista, se separan los valores por comas, por ejemplo:

```
Case 1, 2, 3
```

Para indicar un intervalo de valores, ambos inclusive, usamos la palabra clave **To**. Por ejemplo, si queremos considerar válidos los valores comprendidos entre 1 y 3:

```
Case 1 To 3
```

Si queremos usar operadores relacionales, utilizamos la palabra clave **Is**. Por ejemplo, para tomar como válidos todos los valores mayores que 3 utilizamos:

```
Case Is >3
```

Si los bloques de instrucciones van a llevar solo una instrucción sencilla, podemos ponerla a continuación de la expresión del **Case** separada con dos puntos, como, por ejemplo:

```
Case 1: MsgBox "Lunes"
```

Tengamos en cuenta que, si más de un **Case** coincide con la expresión de prueba, solamente se ejecutará el bloque de instrucciones asociado con la primera coincidencia. VBA ejecutará las instrucciones de la cláusula (opcional) **Case Else** si ningún valor de la lista de expresiones coincide con la expresión de prueba. Veamos el siguiente ejemplo.

Una casa de artículos para el hogar ofrece a sus clientes los siguientes planes de pago:

- **Plan 1**: si el cliente paga el 100% al contado, se hace un 10% de descuento sobre el precio contado.
- **Plan 2**: si el cliente paga el 50% al contado y el resto en 2 cuotas iguales, el precio se incrementa un 10%.
- **Plan 3**: si el cliente paga el 25% al contado y el resto en 3 cuotas iguales, el precio se incrementa un 15%.

Dado el precio del artículo y el tipo de plan, se muestra un mensaje con los valores por pagar. Si el plan ingresado no es correcto, se mostrará un mensaje. Para esto, escribimos el siguiente procedimiento:

```
Sub ejemplo_SelectCase()
Dim op As Integer
Dim monto As Single
Dim pcont As Single, pfin As Single

monto = InputBox("Ingrese monto a pagar")
op = InputBox("Ingrese el tipo de plan (1, 2 o 3)")

Select Case op
   Case 1
      pcont = monto - monto * 10 / 100
      MsgBox "Paga al contado: " & pcont

   Case 2
      pcont = monto + monto * 10 / 100
      pfin = pcont / 2
      cuota = pfin / 2
      MsgBox "Paga al contado: " & pfin & _
      Chr(13) & "y dos cuotas de: " & cuota

   Case 3
      pcont = monto + monto * 15 / 100
      pfin = pcont * 25 / 100
      cuota = (pcont - pfin) / 3
      MsgBox "Paga al contado: " & pfin & _
      Chr(13) & "y tres cuotas de: " & cuota

   Case Else
      MsgBox "Eligio mal el tipo de plan"

End Select

End Sub
```

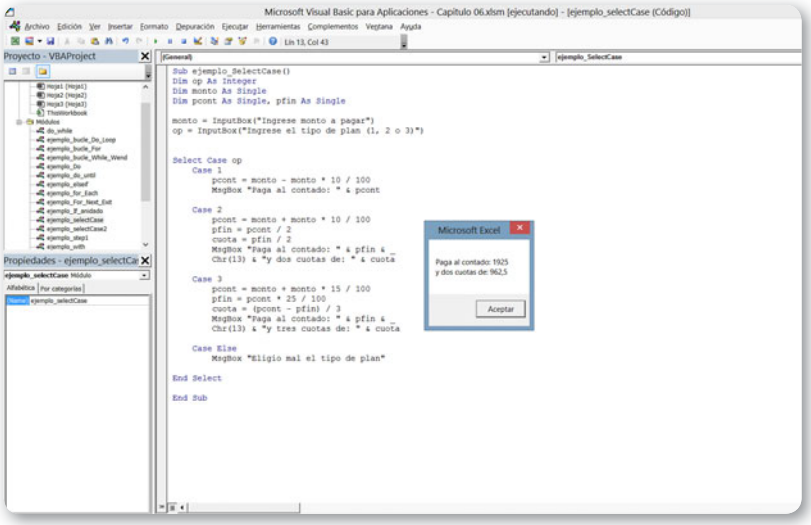


Figura 5. Con la estructura **Case** podemos evaluar una variable y realizar acciones dependiendo de su valor.

Veamos otro ejemplo, pero esta vez empleando operadores de comparación. Se pide al usuario que ingrese una longitud expresada en centímetros. Si la longitud es menor que 30,48 cm, se la debe mostrar en pulgadas; si es mayor o igual que 91,44 cm, se la debe mostrar en yardas, y, en caso contrario, se la debe mostrar en pies. (Sabemos que 1 yarda = 3 pies, 1 pie = 12 pulgadas y 1 pulgada = 2,54 cm).

Para esto, escribimos el siguiente procedimiento:

```
Sub ejemplo_SelectCase()
'L = longitud, inch = pulgada
'ft = pie, yd = yarda

Dim L As Single, inch As Single
Dim ft As Single, yd As Single

L = Val(InputBox("Ingrese Longitud en cm:"))

Select Case L
   Case Is < 30.48
```

```
        inch = L / 2.54

        MsgBox L & " cm equivalen a " & inch & _
        " pulgadas"

    Case 30.48 To 91.44
        ft = L / (12 * 2.54)

        MsgBox L & " cm equivalen a " & ft & " pies"

    Case Else
        yd = L / (3 * 12 * 2.54)

        MsgBox L & " cm equivalen a " & yd &
        " yardas"

End Select

End Sub
```

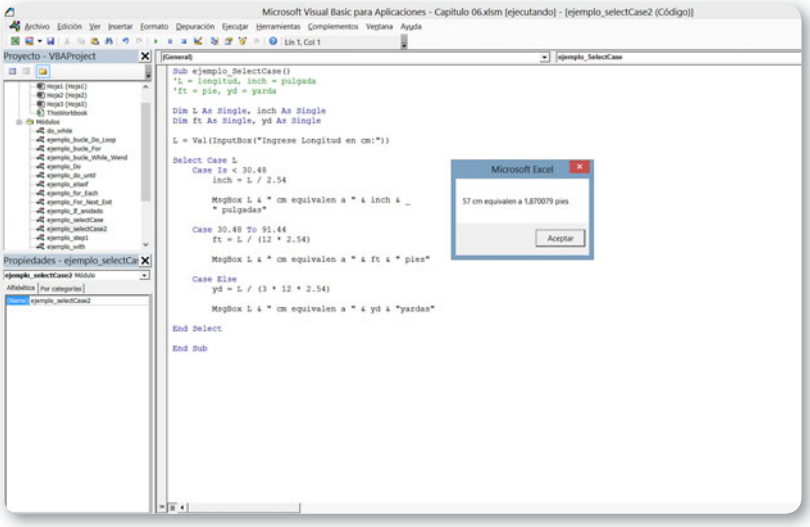


Figura 6. Incluimos las palabras clave **Is** y **To** en la expresión, para identificar la comparación que estamos realizando.

Estructura With...End With

Dado que la mayoría de los objetos tienen diversas propiedades, a veces necesitaremos realizar varia acciones sobre un mismo objeto, por lo cual repetiremos una y otra vez el mismo objeto para realizar esa serie de operaciones. Podemos evitar esto, empleando la estructura **With...End With**, que permite nombrar un objeto y realizar varias instrucciones en él, sin tener que nombrarlo varias veces. De esta manera, se facilita la escritura del código cuando hacemos referencia a los mismos miembros de un objeto, ya que solo debemos indicar sus propiedades y métodos.

Su sintaxis es la siguiente:

```
With objeto
      [instrucciones]
End With
```

En el ejemplo que detallamos a continuación, veremos cómo poner una fila entera en negrita, fuente Comic Sans Ms, tamaño 14, color verde:

```
Sub ejemplo_with()

   With Range("A1:E1").Font
      .Bold = True
      .Name = "Comic Sans MS"
      .Size = 14
      .ColorIndex = 4
   End With

End Sub
```

OBJETOS DE EXCEL

En el **Capítulo 1**, explicamos la jerarquía de objetos de VBA Excel y cómo hacer referencia a ellos, a sus propiedades, métodos y eventos. En el **Capítulo 7**, hablaremos de los principales objetos de Excel y trabajaremos con sus propiedades y métodos, donde la estructura **With...End With** nos será de gran utilidad.

Si no hubiésemos utilizado la estructura **Wiht...End With**, deberíamos haber escrito el siguiente procedimiento:

```
Sub ejemplo_with()

   Range("A1:E1").Select
   Selection.Font.Bold = True
   Selection.Font.Name = "Comic Sans MS"
   Selection.Font.Size = 14
   Selection.Font.ColorIndex = 4

End Sub
```

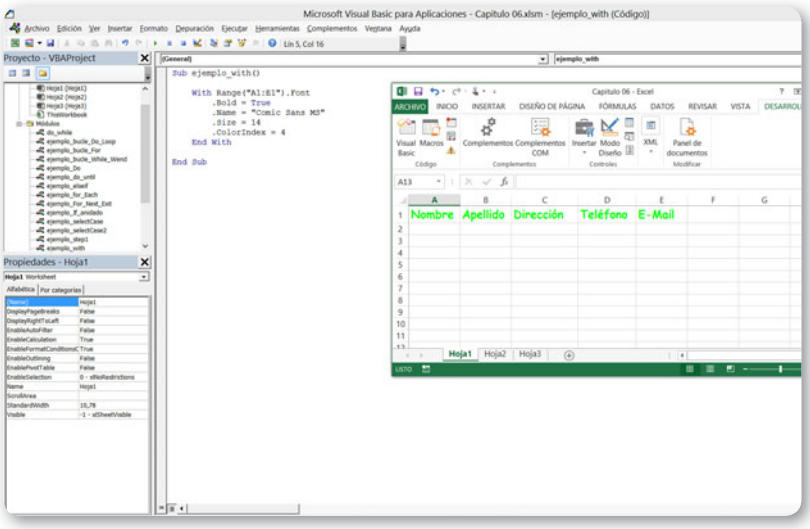


Figura 7. Como podemos apreciar, utilizando este tipo de estructura, el código es más limpio y más fácil de mantener

INICIALIZAR CONTADORES O ACUMULADORES

Si vamos a sumar o restar, inicializaremos los contadores en cero al principio del programa para que no empiece con valores residuales. En cambio, si vamos a multiplicar, los inicializaremos en 1 porque, si valen cero, todo lo que multipliquemos seguirá valiendo cero.

Estructuras de ciclo

Las **estructuras de ciclo**, llamadas también estructuras de bucle, nos permiten repetir uno o varios bloques de código un número determinado de veces de acuerdo con una condición. En tanto esta condición se cumpla, es decir, devuelva un valor verdadero (**True**), el ciclo seguirá ejecutándose. En el momento en que la condición devuelva una respuesta falsa (**False**), el ciclo terminará y el programa ejecutará el resto del código que se encuentra luego del ciclo.

> EN EL MOMENTO EN QUE LA CONDICIÓN DEVUELVE UNA RESPUESTA FALSA, EL CICLO TERMINA

Por ejemplo, podemos usar estas estructuras para validar el ingreso de un usuario a un determinado sistema. El usuario ingresa la contraseña y entra en un ciclo; si esta es incorrecta, devuelve falso, y el ciclo seguirá ejecutándose hasta que escriba la contraseña correcta. En caso contrario, devolverá verdadero y se saldrá del ciclo.

Entre las estructuras de bucle que acepta Visual Basic para Aplicaciones, se incluyen las que nombramos a continuación:

- **For...Next**
- **For...Each...Next**
- **Do...Loop**
- **While...Wend**

Estructura For...Next

Cuando necesitamos repetir una operación un número fijo de veces dentro de un procedimiento, utilizamos la estructura **For...Next**.

La estructura **For** usa una variable llamada **contador**, que incrementa o reduce su valor en cada repetición del bucle. Su sintaxis es la siguiente:

```
For contador = inicio To final [Step incremento]
        Bloque de instrucciones
Next
```

Donde:

- **contador**: es la variable que usaremos como contador, debe ser de tipo numérico. Esta variable incrementa o decrementa su valor de manera constante, por ejemplo de 1 en 1
- **inicio**: esta es una expresión de tipo numérica cuyo valor tomará el contador la primera vez.
- **final:** es una expresión numérica cuyo valor usará **For** de manera que solo entrará si el contador no supera el valor de esta expresión.
- **incremento**: es opcional. Es una expresión numérica que puede ser positiva o negativa. Si el incremento es positivo, **inicio** debe ser menor o igual que **final**, de lo contrario no se ejecutarán las instrucciones del bucle. Si incremento es negativo, **inicio** debe ser mayor o igual que **finalizar** para que se ejecute el bloque de instrucciones. Si no se establece el **Step**, el valor predeterminado de incremento es 1.
- **Bloque de instrucciones**: es el conjunto de sentencias que se ejecutarán con cada ejecución del bucle.
- **Next**: esta palabra clave sirve para delimitar el final del bucle. Cuando el bucle se encuentra con el **Next**, se vuelve otra vez al principio del **For**, y así hasta realizar el número de ejecuciones determinadas.

> UTILIZAMOS LA ESTRUCTURA FOR... NEXT PARA REPETIR UNA OPERACIÓN UN NÚMERO DE VECES

En el siguiente ejemplo, vamos a hacer un procedimiento que permita realizar la sumatoria de los números enteros múltiplos de 5, comprendidos entre el 1 y el 100, es decir, 5 + 10 + 15 + ... + 100, y muestre el resultado en un cuadro de mensaje.

ACUMULADORES/CONTADORES

La diferencia entre acumuladores y contadores es que los primeros se incrementan con cualquier número, como, por ejemplo, el total de una factura; mientras que los segundos se incrementan siempre con el mismo número, normalmente **1**. Los contadores se usan generalmente en un ciclo **For...Next** en donde toma un valor inicial y se incrementa hasta un valor final.

```
Sub ejemplo_bucle_For_Next()

Dim i As Integer
Dim suma As Integer

   For i = 1 To 100

      If (i Mod 5) = 0 Then

         suma = suma + i

      End If

   Next

   MsgBox "La suma de los múltiplos de 5 entre 1 y 100 es: "  &suma

End Sub
```

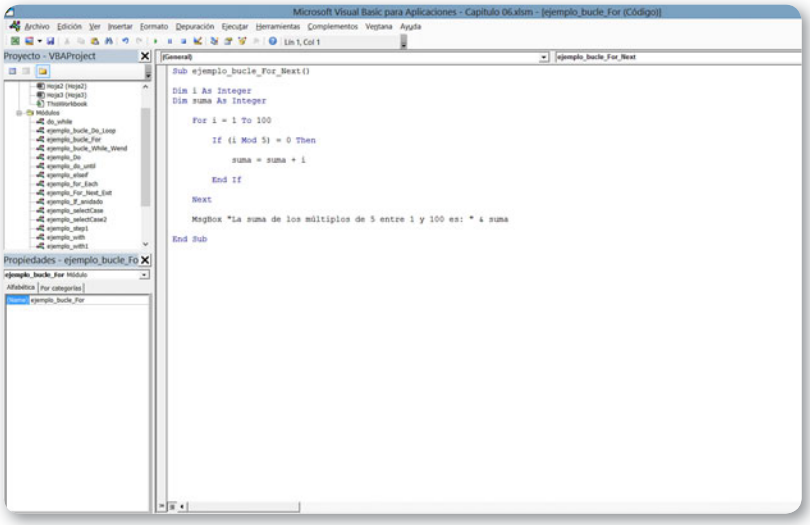


Figura 8. Este tipo de bucles es el más usado para recorrer los vectores de índices numéricos secuenciales.

Al entrar al bucle **For**, la variable **i** toma el valor **1**; a continuación, se ejecuta el bloque de sentencias y, luego, se incrementa la variable **i** en **1**. Es decir, el valor de la variable **i** se convertirá en **2** en la primera vuelta, y así sucesivamente hasta que la variable tome el valor **101**. Con esto, la condición ya no será igual a 100, y se saldrá del bucle, siguiendo la ejecución del código que venga a continuación del final del **Next**.

> PODEMOS MODIFICAR EL VALOR DE INCREMENTO USANDO LA PALABRA CLAVE NEXT

Como podemos ver, la variable que usamos como contador en un bucle **For...Next** se incrementa en **1** (uno) cada vez que se ejecuta el bucle. Sin embargo, es posible modificar este incremento utilizando la palabra clave **Step** seguida de una expresión numérica.

Por ejemplo, el siguiente procedimiento calcula la potencia de dos de los números naturales impares que se encuentran comprendidos entre 1 y 10, es decir 1^2, 3^2, 5^2, 7^2, 9^2.

```
Sub ejemplo_step()

    Dim i As Integer
    Dim suma As Integer
    Dim potencia As Integer

    For i = 1 To 10 Step 2

        potencia = i ^ 2

        Range("A" & i).Value = i
        Range("B" & i).Value = "="
        Range("C" & i).Value = potencia

    Next

End Sub
```

En este caso, al entrar al bucle **For**, la variable **i** toma el valor **1**; a continuación, se ejecuta el bloque de sentencias y, luego, se incrementa la variable **i** en **2**. Es decir, el valor de la variable de **i** se convertirá en **3** en la primera vuelta, y así sucesivamente hasta que la variable tome el valor **11**. De esta manera, la condición ya no será igual a 10, y se saldrá del bucle, siguiendo la ejecución del código que se encuentre a continuación del final del **Next**.

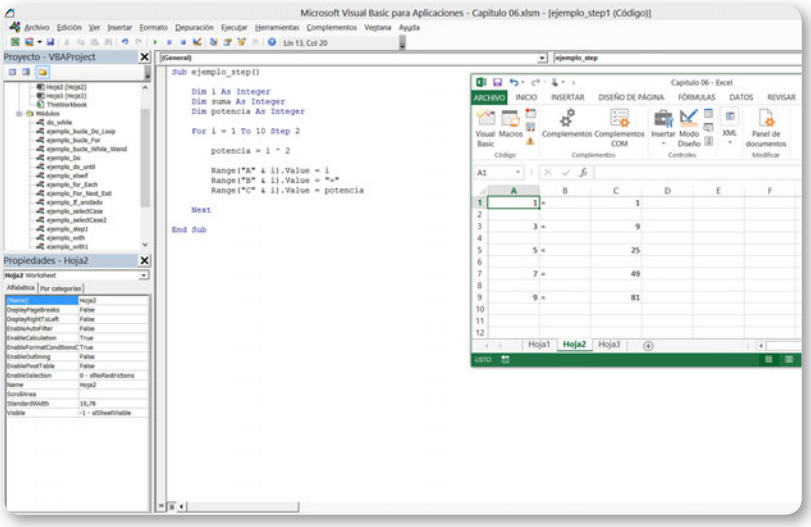


Figura 9. En un bucle, también podemos indicar valores decimales para el incremento y las variables **`inicio`** y **`fin`**.

Estructuras For Each...Next

Como explicamos en el **Capítulo 1**, VBA es un lenguaje orientado a objetos. Por lo tanto, es común que tengamos que trabajar con una colección de objetos, como, por ejemplo, hojas de cálculo (**Worksheets**).

PRECAUCIÓN

Es importante tener en cuenta que, cuando en los bucles **For...Next** usamos la sentencia **Step** fraccionaria, estos se ejecutarán más lentamente que los que tienen valores enteros. Lo mismo sucede con los bucles que emplean variables variantes para el contador, aunque sean enteros.

La estructura **For...Each...Next** nos permite recorrer todos los elementos de una colección o matriz y realizar acciones para cada uno de los elementos. Su sintaxis es la siguiente:

```
For Each elemento In grupo
        [Instrucciones]
        [Exit For]
        [Instrucciones]
Next [elemento]
```

Donde:

- **elemento**: es la variable de tipo objeto o **Variant** que se emplea para recorrer los elementos de la colección o matriz. Si es una colección, esta puede ser una variable del tipo **Variant** o un objeto genérico o especifico, como por ejemplo, un objeto **Worksheet**. En cambio, si es una matriz, esta variable solo puede ser del tipo **Variant**.
- **grupo**: es el nombre del conjunto de objetos o de una matriz.
- **Instrucciones**: representa el grupo de sentencias por realizar.
- **Next**: indica el final del bucle.

En este ejemplo incluimos un encabezado de página alineado a la derecha en todas las hojas de un libro de trabajo.

```
Sub ejemplo_For_Each()

Dim Pagina As Object

For Each Pagina In ThisWorkbook.Sheets

   Pagina.PageSetup.RightHeader = "Empresa ABC"

Next Pagina

End Sub
```

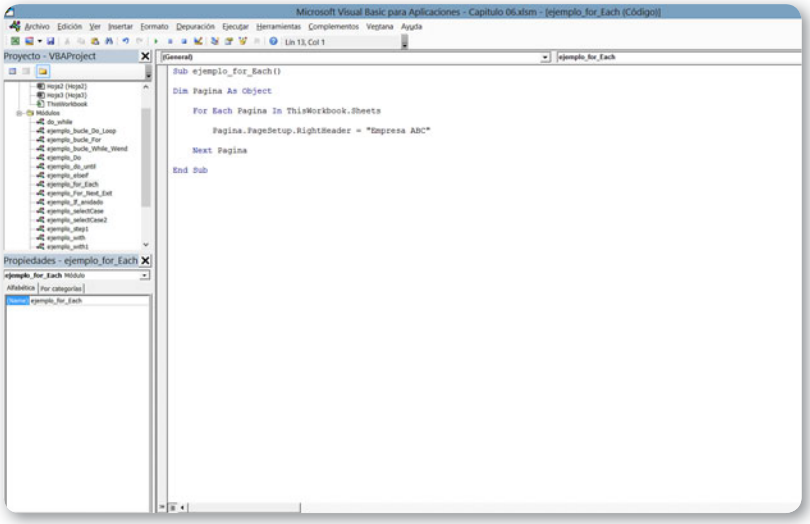


Figura 10. La grabadora de macros no puede registrar este tipo de bucles.

Salir de las estructuras For...Next y For Each...Next

Algunas veces, necesitamos salir de un bucle **For...Next** antes de que se haya ejecutado el número de veces que hemos previsto, para esto contamos con la sentencia **Exit For.**

En el siguiente código, se le solicita al usuario que ingrese su contraseña; si la contraseña es correcta, se muestra un mensaje de bienvenida y se sale del bucle. En caso contrario, se le solicita nuevamente el ingreso de su contraseña, y tiene hasta cinco intentos de ingresarla correctamente. Si después de cinco intentos no la ingresa en forma correcta, sale del bucle y muestra un mensaje de error.

```
Sub ejemplo_For_Next_Exit()

Dim contrasenia As String
Dim i As Integer

   For i = 1 To 5
   contrasenia = InputBox("Ingrese contraseña")
```

```
        If contrasenia = "123" Then
            mensaje = 1
            Exit For
        Else
            mensaje = 2
        End If
    Next

    If mensaje = 1 Then
        MsgBox "Bienvenido al sistema"
    Else
    MsgBox "Ha olvidado su contraseña, " & _
    "comuníquese con el administrador", _
     vbOKOnly + vbCritical

End If

End Sub
```

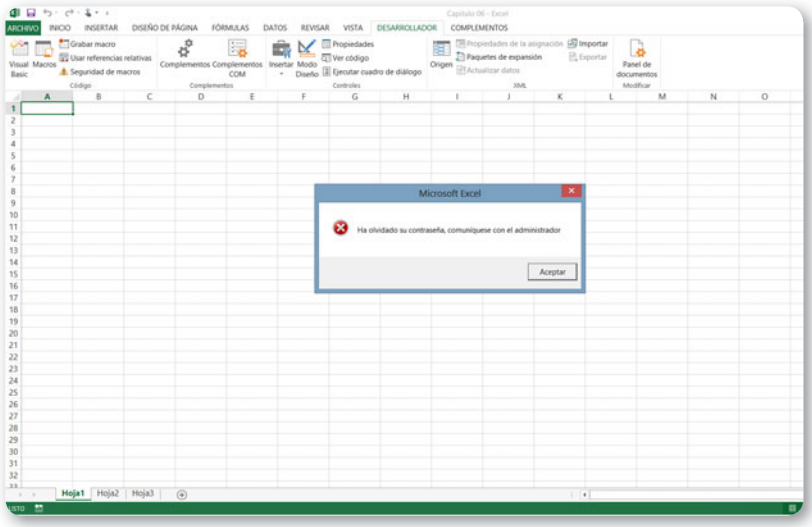


Figura 11. Al no ingresar la contraseña correcta, aparecerá un **MsgBox** donde se muestra solo el botón **Aceptar**.

Estructuras Do...Loop

Muchas veces, necesitamos seguir repitiendo una operación o no, dependiendo de los resultados obtenidos dentro del bucle. La estructura **Do...Loop** permite ejecutar un bloque de instrucciones un número indefinido de veces, hasta que se cumpla una condición. Este tipo de bucle hace posible probar una condición al comienzo o al final de una estructura de bucle. También podemos especificar si el bucle se repite mientras (**While**) la condición sigue siendo verdadera (**True**) o hasta que se convierta en verdadera (**Until**). Según esto, tenemos las variantes de la estructura **Do...Loop** que veremos a continuación.

Do...Loop While

Este bucle se repite mientras la condición se cumple.

```
Do
        Instrucciones
Loop While condición
```

Do...Loop Until

Este bucle se repite hasta que la condición se cumple.

```
Do
        Instrucciones
Loop Until condición
```

TECLA F8

Podemos ejecutar bucles en la modalidad paso a paso desde la ventana del editor de Visual Basic para Aplicaciones. Si presionamos la tecla **F8**, podremos comprobar el funcionamientos de un bucle **For... Next**, **Do...Loop**, **Do While...Loop**, por ejemplo, y así poder depurar los posibles errores en un código.

Tanto en la estructura **Do...Loop While** como en la estructura **Do...Loop Until**, las instrucciones se ejecutan primero, y se prueba la condición después de cada ejecución.

En el siguiente ejemplo, se le solicita al usuario que ingrese la cantidad de números para realizar la sumatoria de los múltiplos de 5, entre 1 y la cantidad ingresada.

```
Sub ejemplo_bucle_Do_Loop()

Dim i As Integer
Dim n As Integer
Dim suma As Integer

n = InputBox("ingrese cantidad de números")

   Do
      i = i + 1

      If (i Mod 5) = 0 Then

         suma = suma + i

      End If

   Loop While i < n

   MsgBox "La suma de los múltiplos de 5 entre" & _
        "1 y " & n & " es: " & suma
End Sub
```

ESTRUCTURAS DO...LOOP

Un error muy frecuente que solemos cometer con las estructuras de repetición **Do...Loop** consiste en olvidarnos de incrementar el contador. Esto da lugar a bucles infinitos, es decir, bucles que no acaban nunca. Otro error común es poner mal la condición de finalización del bucle.

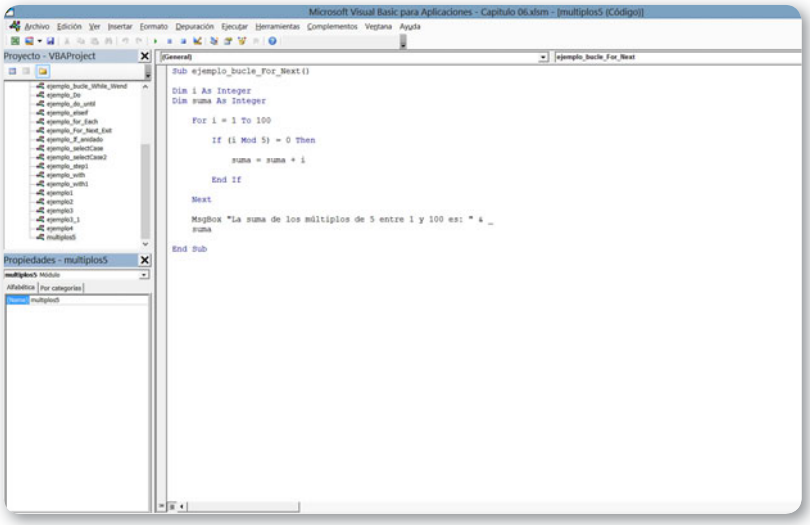


Figura 12. Usando la condición al final del bucle, se garantiza al menos una ejecución de instrucciones.

Do While...Loop

En este tipo de bucle, VBA evalúa primero la condición. Si esta es verdadera (**True**), se ejecutan las instrucciones y, luego, vuelve a la instrucción **Do While** y prueba la condición otra vez. Si la condición es falsa (**False**), se salta todas las instrucciones.

```
Do While condición
        Instrucciones
Loop
```

El procedimiento que presentamos a continuación calcula la suma de los primeros 10 números enteros.

¿QUÉ CICLO USAR?

Podemos usar cualquier ciclo para resolver un determinado problema. El ciclo que elijamos dependerá de nuestra necesidad. No debemos olvidar que cuanto menos código tengamos que escribir y menos variables sea necesario controlar mejor será la interpretación de una macro.

```
Sub ejemplo_do_while()

   Do While i <= 10

  suma = suma + i

   i = i + 1

   Loop

MsgBox "La suma de los primeros 10 " & _
      "nros enteros es: " & suma

End Sub
```

Do Until...Loop

Esta estructura es una variante del bucle **Do While...Loop**. En este caso, el bucle se repite siempre y cuando la condición sea falsa (**False**) en vez de verdadera (**True**), es decir, el bucle se repite mientras no se cumpla una determinada condición.

```
Do Until condición
       Instrucciones
Loop
```

El procedimiento que presentamos a continuación muestra cómo podemos calcular la suma de los primeros 10 números enteros empleando la estructura **Do Until...Loop**.

```
Sub ejemplo_do_until()

   Do Until i > 10

  suma = suma + i
```

```
    i = i + 1

    Loop

MsgBox "La suma de los primeros 10 " & _
       "nros enteros es: " & suma

End Sub
```

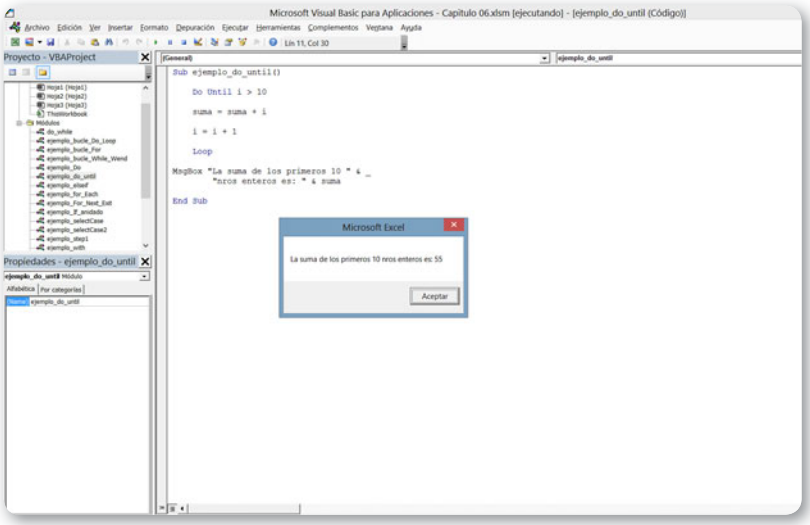


Figura 13. El bucle se ejecuta hasta que la variable **i** (contador) sea mayor a 10.

Salir de una estructura Do

Al igual que con las estructuras **For...Next** y **For Each...Next**, es posible que tengamos situaciones en las que deseamos salir de un bucle **Do... Loop** antes de ejecutar las sentencias restantes del bucle. Para ello, contamos con la instrucción **Exit Do**. Cuando VBA encuentra una instrucción **Exit Do**, el control se transfiere directamente a la siguiente instrucción que se encuentra fuera del bucle.

El código que presentamos a continuación muestra un ejemplo de cómo salir de un bucle **Do...Loop**.

```
Sub ejemplo_Do()
Dim contrasenia As String
Dim i As Integer
i = 0
Do
i = i + 1

   If i > 5 Then
      MsgBox "Ha olvidado su contraseña, " & _
            "comuníquese con el administrador"
      Exit Sub
   End If

   contrasenia = InputBox("Ingrese su contraseña")

   Loop Until contrasenia = "123"

   MsgBox "Bienvenido al sistema"

End Sub
```

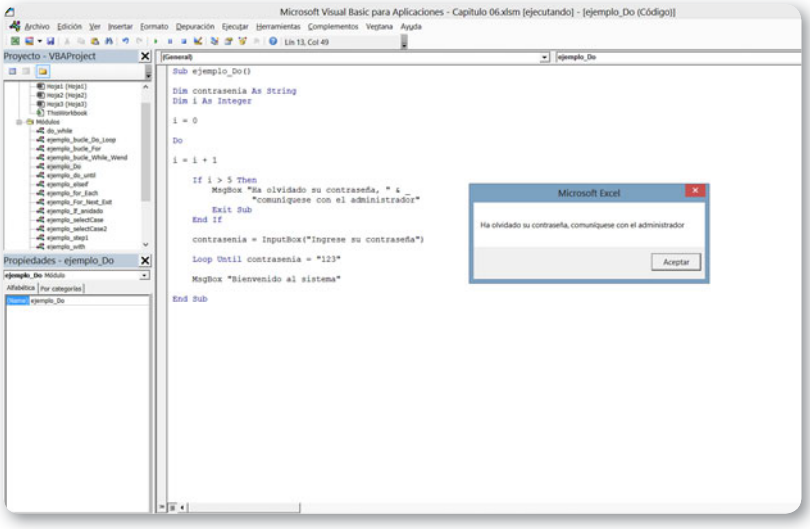


Figura 14. Si el usuario ingresa mal la contraseña, luego de 5 intentos se sale del bucle mostrándole un mensaje.

Estructuras While...Wend

VBA permite otra variante del bucle **Do While...Loop**, es decir, con la comprobación al principio del bucle. La estructura **While...Wend** podemos utilizarla cuando no conocemos por anticipado el número de iteraciones. Su sintaxis es la siguiente:

```
While condición
        Instrucciones
Wend
```

Cuando se ejecuta este bucle, VBA evalúa primero la condición; si esta es verdadera (**True**), ejecuta el bloque de instrucciones que siguen hasta la sentencia **Wend**, en caso contrario sale del bucle.

Por ejemplo, podemos usar la estructura **While...Wend** para calcular la suma de **n** números naturales.

```
Sub ejemplo_bucle_While_Wend()

Dim contador As Integer
Dim n As Integer
Dim suma As Integer

n = InputBox("ingrese cantidad de números")

   While contador <= n

      contador = contador + 1

      If (contador Mod 5) = 0 Then

         suma = suma + contador

      End If

   Wend
```

```
MsgBox "La suma de los múltiplos de 5 " & _
       "entre 1 y " & n & " es: " & suma
End Sub
```

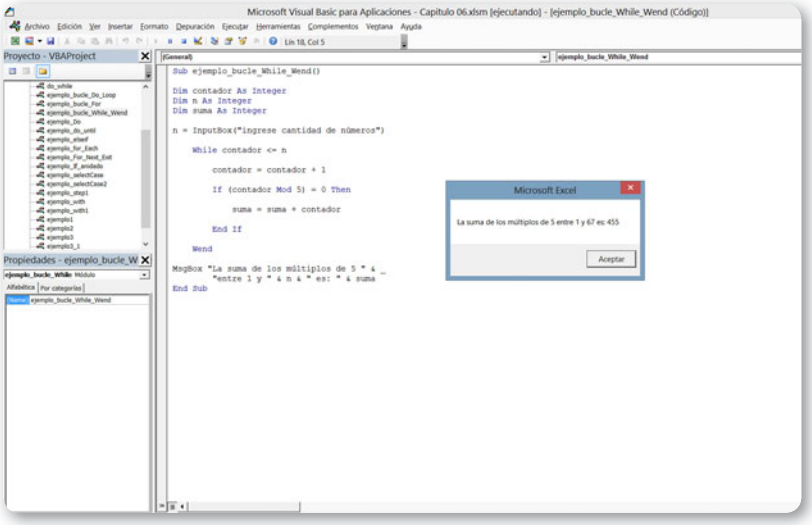


Figura 15. Los bucles **`While`...`Wend`** se incluyen en VBA para la compatibilidad con versiones anteriores.

RESUMEN

Las estructuras o sentencias de control se clasifican en sentencias selectivas o condicionales, y repetitivas o bucles. En este capítulo, describimos los principales tipos de estructuras en VBA. Dentro de las sentencias selectivas, se destacan las estructuras If...Then...Else y Select Case, que permiten ejecutar o no un conjunto de instrucciones dependiendo de la evaluación de ciertas condiciones. Dentro de las estructuras de repetición, se distinguen los bucles For...Next y Do...Loop, que permiten repetir una tarea varias veces a una gran velocidad.

Actividades

TEST DE AUTOEVALUACIÓN

1. Defina las estructuras condicionales y enuncie su clasificación.
2. ¿Qué diferencia hay entre **ElseIf** y **Else**?
3. ¿Cuándo es conveniente usar la instrucción **Select Case**?
4. Defina las estructuras repetitivas y enuncie su clasificación.
5. ¿Para qué se emplea un contador?
6. ¿Para qué se emplea un acumulador?
7. ¿Cuál es la sentencia que nos permite salir de un ciclo **For…Next**, antes que este termine su ejecución?
8. ¿Para qué se emplea la instrucción **With End With**?
9. ¿Qué diferencia existe entre la estructura **Do While…Loop** y **Do…Loop While**?
10. ¿Cuál es la sentencia que nos permite salir de un ciclo **DO**?

EJERCICIOS PRÁCTICOS

1. Desarrolle un procedimiento que permita realizar la suma de los primeros 10 números enteros.
2. Desarrolle un procedimiento que le permita ingresar un número al usuario (del 1 al 12) y le devuelva el nombre del mes correspondiente.
3. Realice un procedimiento que permita ingresar dos números por el teclado y determinar cuál de los dos valores es el menor. Mostrarlo en un cuadro de mensaje.
4. Efectuar un procedimiento que lea los primeros 300 números enteros y determine cuántos de ellos son impares; al final, se deberá mostrar su sumatoria.
5. Desarrolle un procedimiento que permita calcular el promedio de notas de un grupo de alumnos. Este finaliza cuando **N = 0**.

*07

Principales objetos de Excel

VBA es un lenguaje de programación que permite manipular los objetos de Excel: las hojas de cálculo, los rangos de celdas, los libros y la propia aplicación. En este capítulo, nos introduciremos en los conceptos de programación específicos de VBA en Microsoft Excel, que nos permitirán acceder y controlar las propiedades, los objetos y sus funcionalidades.

Modelo de Objetos de Excel

Como explicamos en el **Capítulo 1**, el modelo de objetos de Excel contiene una gran cantidad de elementos ordenados en forma jerárquica. Aunque disponemos de muchos objetos, casi siempre utilizaremos estos cuatro:

- **Application**: aplicación
- **Workbook**: libro de trabajo
- **Worksheet**: hoja de cálculo
- **Range**: celda

A continuación, explicaremos las características de cada uno de estos objetos, como así también conoceremos sus principales propiedades y métodos, y la manera de personalizarlos.

Application

Application es el objeto de nivel superior en el modelo de objetos de Excel; representa al programa y al libro de trabajo.

Brinda acceso a las opciones y las configuraciones a nivel de la aplicación, como, por ejemplo, de la cinta de opciones, de la impresora activa, el tamaño de la fuente por defecto, entre otras.

Entonces, al modificar las propiedades y los métodos de este objeto, estaremos realizando cambios que se van a reflejar en la interfaz de Excel y en sus diferentes herramientas.

JERARQUÍA DE LOS OBJETOS

Como **Application** es el objeto principal dentro del modelo de objetos de Excel y todos los demás objetos derivan de él, la mayoría de las veces no será necesario referirnos explícitamente a este objeto, ya que se supone que todos los demás provienen de él. Conocer este detalle nos permite simplificar y abreviar la escritura del código al trabajar con los distintos objetos de Excel.

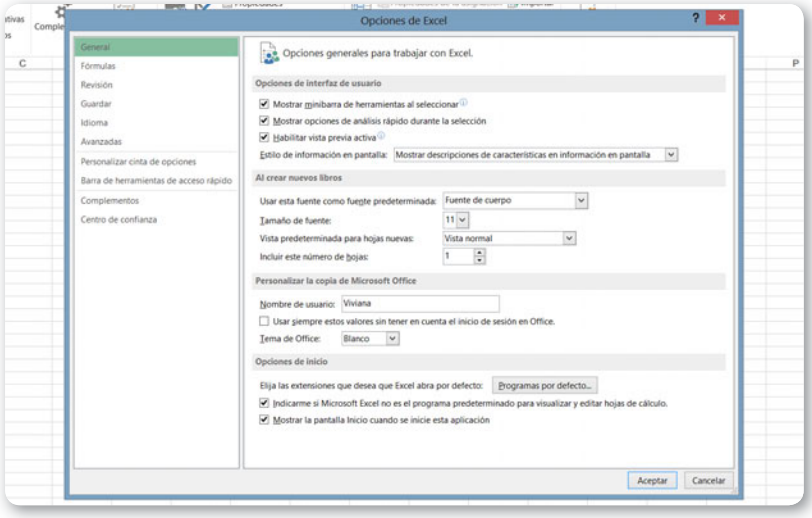


Figura 1. Muchas de las opciones que podemos modificar con el objeto **`Application`** son las que encontramos en **`Archivo/Opciones`**.

Propiedades del objeto Application

Application es un objeto que tiene una gran cantidad de propiedades, algunas definen el ambiente donde se ejecuta Microsoft Excel, otras controlan la presentación de la interfaz (como por ejemplo el aspecto del puntero del mouse en Excel), y otras devuelven objetos.

A continuación, describiremos algunas de las propiedades más importantes.

> APPLICATION ES EL OBJETO DE NIVEL SUPERIOR EN EL MODELO DE OBJETOS DE EXCEL

Caption

Esta propiedad devuelve o establece el nombre que aparece en la **Barra de título** de la ventana de la aplicación principal (Microsoft Excel).

Por ejemplo, si queremos cambiar el título Microsoft Excel, que es el que aparece por defecto, por Liquidación de Haberes, utilizamos la sentencia que mostramos a continuación:

```
Application.Caption = "Liquidación de Haberes"
```

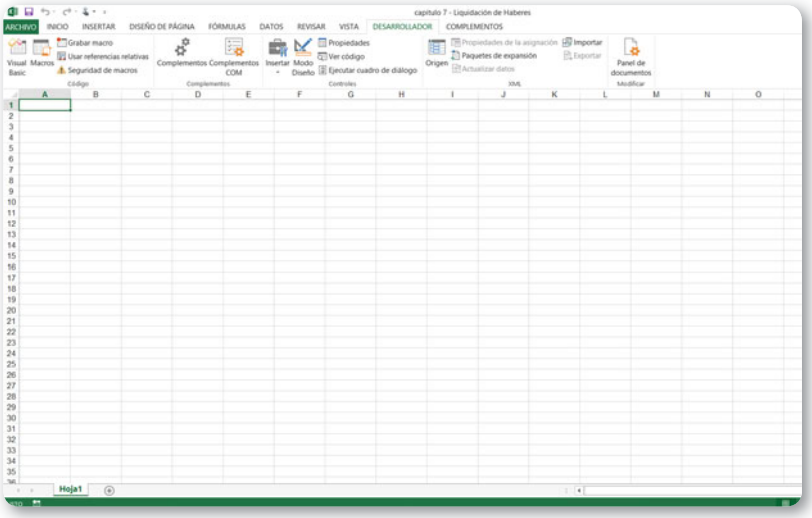


Figura 2. Si establecemos el valor **`Empty`** para el nombre, la propiedad **`Application.Caption`** devuelve Microsoft Excel.

Path

Esta propiedad devuelve la unidad y la carpeta donde está instalado Microsoft Excel, por ejemplo, **C:\Program Files (86)\Microsoft Office\Office14**. Para que nos muestre en un cuadro de mensaje la ubicación de Microsoft Excel, podemos usar la siguiente sentencia:

```
MsgBox "Microsoft Excel está instalado en" & _
    Application.Path
```

DefaultFilePath

Esta propiedad podemos usarla para establecer la ubicación predeterminada que utiliza Microsoft Excel para abrir sus archivos. En este caso, usamos la siguiente sentencia:

```
MsgBox "La ruta de los archivos por defecto es " & _
    Application.DefaultFilePath
```

En cambio, si queremos cambiar la ubicación predeterminada, podemos utilizar la siguiente sentencia:

```
Application.DefaultFilePath = "C:\Users\Planillas"
```

StatusBar

Esta propiedad la utilizamos para mostrar mensajes de texto personalizados en la barra de estado mientras se está ejecutando una macro. Por ejemplo, si estamos ejecutando un proceso de actualización de datos, podemos mostrar en la barra de estado el mensaje **Actualizando**.... Para esto escribimos la siguiente sentencia:

> STATUSBAR MUESTRA MENSAJES PERSONALIZADOS EN LA BARRA DE ESTADO

```
Application.StatusBar = "Actualizando....."
```

Este mensaje va a permanecer en la barra de estado hasta que otro mensaje se escriba en ella, ya sea porque le asignamos a la propiedad **StatusBar** otro valor o porque Microsoft Excel lo hace.

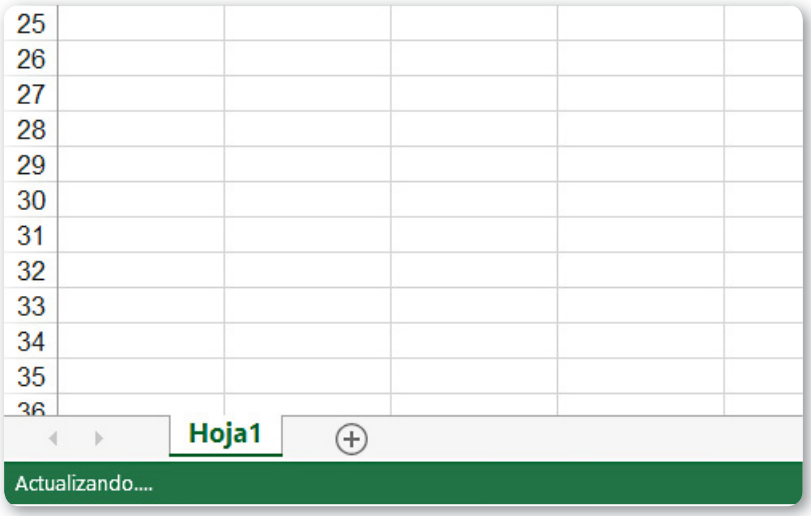


Figura 3. Para borrar los mensajes de la barra de estado, le asignamos a la propiedad **`StatusBar`** el valor **`False`** o una cadena nula (**`""`**).

DisplayFormulaBar

Esta propiedad permite mostrar u ocultar la **Barra de fórmulas**. Por ejemplo, si queremos ocultar la **Barra de fórmulas** para ganar espacio en la pantalla, podemos utilizar la siguiente sentencia:

```
Application.DisplayFormulaBar = False
```

Para restablecerla, asignamos el valor **True** a la propiedad **DisplayFormulaBar**.

DisplayAlerts

Esta propiedad determina si la aplicación muestra los cuadros de diálogo de alerta. Por ejemplo, si el código de una macro elimina una hoja de cálculo, Excel normalmente nos mostrará un cuadro de alerta donde se nos pide que confirmemos la eliminación. Para suprimir estos mensajes, establecemos la propiedad **DisplayAlerts** en **False**.

Por ejemplo, el siguiente procedimiento elimina la hoja **auxiliar** y no le pregunta al usuario si desea confirmar esta acción. Luego, volvemos a activar los mensajes de alerta, estableciendo la propiedad en **True**.

```
Sub ejemplo_elimina_hoja()

   Application.DisplayAlerts = False
   Worksheets("auxiliar").Delete
   Application.DisplayAlerts = True

End Sub
```

CUÁNDO USAR STATUSBAR

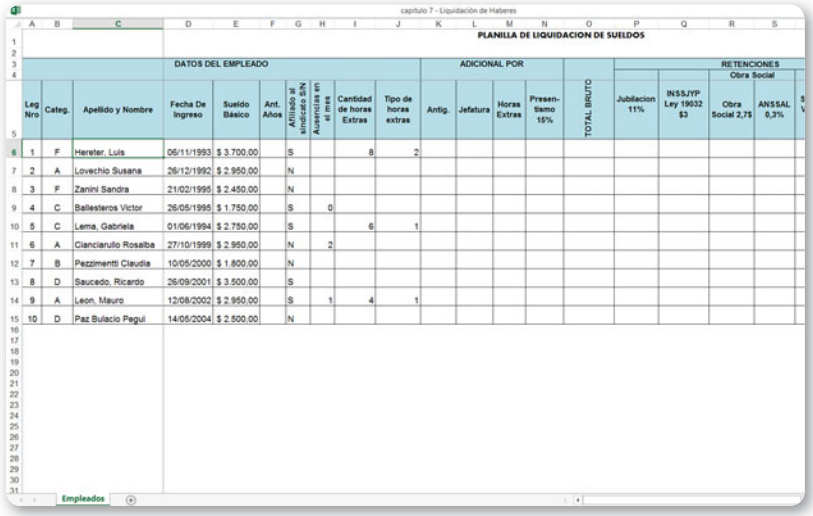
El uso de la propiedad **StatusBar** es muy útil, siempre y cuando la barra de estado esté visible. Recordemos que la interfaz de Excel puede ser personalizada por el usuario, de manera que la barra de estado quede oculta. Antes de mostrar un mensaje, debemos asegurarnos de que la barra de estado esté visible; para esto utilizamos la propiedad **Application.DisplayStatusBar = True.**

DisplayFullScreen

Esta propiedad permite establecer si Microsoft Excel se ejecuta en modo pantalla completa. Además de maximizar la ventana de la aplicación, oculta la barra de título, la cinta de opciones y la barra de estado. Esto nos permite ampliar la visualización de la hoja de cálculo.

Para esto, escribimos la siguiente sentencia:

```
Application.DisplayFullScreen = True
```



capitulo 7 - Liquidación de Haberes

PLANILLA DE LIQUIDACION DE SUELDOS

			DATOS DEL EMPLEADO							ADICIONAL POR						RETENCIONES				
																		Obra Social		
Leg Nro	Categ.	Apellido y Nombre	Fecha De Ingreso	Sueldo Básico	Ant. Años	Afiliado al sindicato S/N	Ausencias en el mes	Cantidad de horas Extras	Tipo de horas extras	Antig.	Jefatura	Horas Extras	Presen-tismo 15%	TOTAL BRUTO	Jubilacion 11%	INSSJYP Ley 19032 $3	Obra Social 2,7$	ANSSAL 0,3%	
1	F	Hereter, Luis	06/11/1993	$ 3.700,00		S		8	2										
2	A	Lovechio Susana	26/12/1992	$ 2.950,00		N													
3	F	Zanini Sandra	21/02/1995	$ 2.450,00		N													
4	C	Ballesteros Victor	26/05/1995	$ 1.750,00		S	0												
5	C	Lema, Gabriela	01/06/1994	$ 2.750,00		S		6	1										
6	A	Cianciarullo Rosalba	27/10/1999	$ 2.950,00		N	2												
7	B	Pezzimentti Claudia	10/05/2000	$ 1.800,00		N													
8	D	Saucedo, Ricardo	26/09/2001	$ 3.500,00		S													
9	A	Leon, Mauro	12/08/2002	$ 2.950,00		S	1	4	1										
10	D	Paz Bulacio Pegui	14/05/2004	$ 2.500,00		N													

Empleados

Figura 4. Para volver al modo de pantalla normal, establecemos en **`False`** la propiedad **`DisplayFullScreen`**.

Métodos del objeto Application

Como vimos en capítulos anteriores, un **método** es un conjunto de comportamientos o acciones que puede realizarse en el objeto. A continuación, describiremos algunos de los métodos que podemos utilizar con el objeto **Application**.

OnTime

Este método se puede utilizar para programar los procedimientos que se ejecuten en momentos específicos del día o en intervalos de tiempo determinados. Su sintaxis es la siguiente:

Application.OnTime (hora,nombre,hora_lim,programar)

Donde:

- **hora**: es la hora a la que se desea ejecutar el procedimiento.
- **nombre**: es el nombre del procedimiento que se va a ejecutar.
- **hora_lim**: es la hora límite a partir de la cual ya no se ejecutará el procedimiento.
- **programar**: puede ser **True** para programar un nuevo procedimiento **OnTime** o **False** para anular un procedimiento establecido previamente.

ONTIME PERMITE PROGRAMAR PROCEDIMIENTOS EN MOMENTOS ESPECÍFICOS DEL DÍA

Por ejemplo, si queremos ejecutar un procedimiento todos los días a las 13:00 para mostrarle un cuadro de mensaje al usuario donde se le informe que es el horario del almuerzo, podemos escribir la sentencia que mostramos a continuación:

Application.OnTime TimeValue("13:00:00"), "almuerzo"

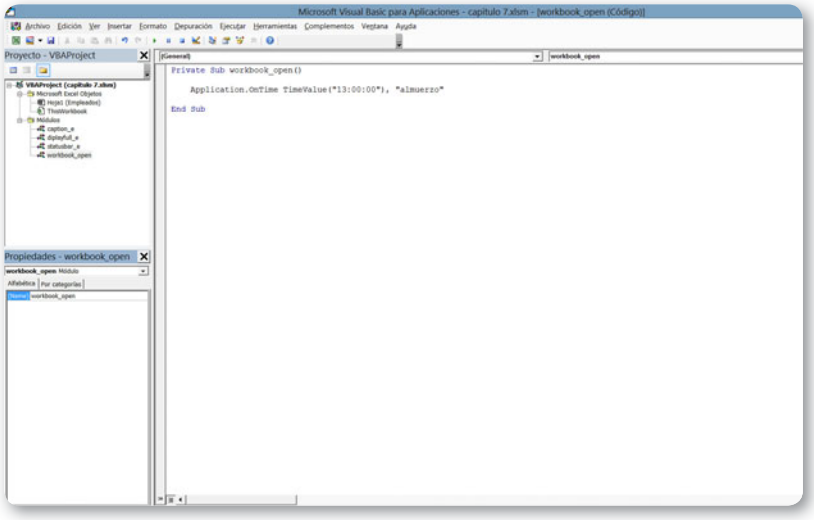


Figura 5. Podemos usar también la expresión **Now+TimeValue(hs)** para ejecutar un procedimiento a partir de un tiempo determinado.

OnKey

Este método ejecuta un procedimiento determinado cuando el usuario presiona una tecla o una combinación de ellas.

Su sintaxis es la siguiente:

Application.Onkey(tecla, procedimiento)

Donde

- **tecla**: es una cadena que indica la tecla que se debe presionar.
- **procedimiento**: es una cadena que indica el nombre del procedimiento que se va a ejecutar.

Por ejemplo, si queremos asignarle al procedimiento **saludo** la combinación de teclas **CTRL + -** (signo menos), usamos la sentencia:

Application.OnKey "^{-}","saludo"

Si deseamos borrar la asignación de teclas realizadas con el método **OnKey** del ejemplo anterior, utilizaremos la siguiente sentencia:

Application.OnKey "^{-}",""

> ONKEY EJECUTA UN PROCEDIMIENTO DETERMINADO CUANDO EL USUARIO PULSA UNA TECLA

Wait

Este método realiza una pausa hasta que transcurra un momento especificado, en una macro que se está ejecutando. Cuando llega a la hora determinada, devuelve el valor **True**.

Su sintaxis es la siguiente: **Application.Wait(hora)**

ARGUMENTOS PARA ONKEY

Como argumento para el método **OnKey**, podemos combinar cualquier tecla individual con las teclas **ALT**, **CTRL**, **MAYÚS**, incluso combinar estas últimas entre sí. Para mayor información, podemos consultar las tablas de códigos que se muestran en la ayuda de Visual Basic para Aplicaciones, presionando la tecla **F1**.

Donde **hora** es cuando se desea que se reanude la macro. El siguiente ejemplo muestra un mensaje luego de haber transcurrido 1 minuto.

```
If Application.Wait(Now + TimeValue("0:01:00")) Then
    MsgBox "Finalizo el proceso de actualización"
End If
```

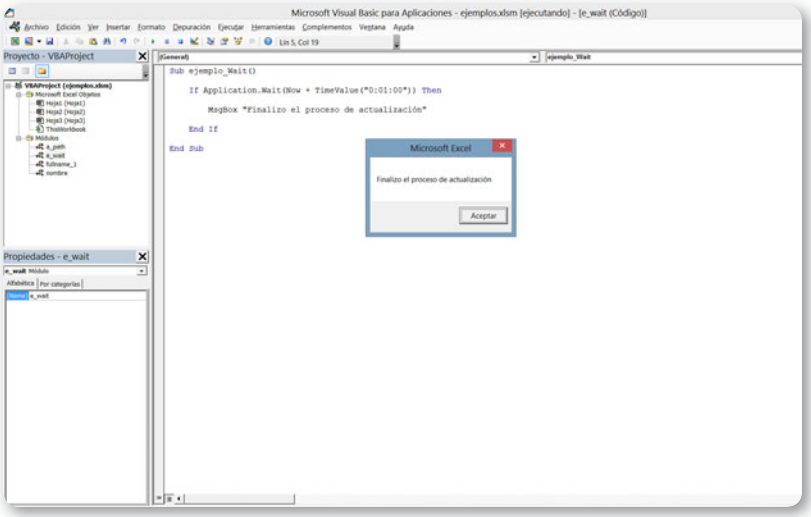


Figura 6. El método **`Wait`** suspende todas actividades de Microsoft Excel, exceptuando los procesos de segundo plano.

Quit

Este método nos permite salir de Microsoft Excel. Si tenemos algún libro abierto y si no hemos guardado los cambios, mostrará un cuadro de diálogo que pregunta si deseamos guardar los cambios. La sentencia que nos permite salir de la aplicación es la siguiente:

```
Application.Quit
```

Si establecimos la propiedad **DisplayAlerts** en **False** o la propiedad **Save** de un libro como **True** (esta propiedad la veremos más adelante), al ejecutar la sentencia **Application.Quit**, se cerrará la aplicación sin preguntarnos si deseamos guardar los cambios realizados en los libros.

Workbooks

El objeto **Workbooks** se encuentra debajo del objeto **Application** en la jerarquía del modelo de objetos de Microsoft Excel. El objeto **Workbook** forma parte de la colección **Workbooks**. Este objeto es devuelto por las siguientes propiedades del objeto **Application**:

- **Application.Workbooks**: devuelve la colección **Workbooks**.
- **Application.ActiveWorkbook**: devuelve el libro activo. Si no hay libros abiertos, esta propiedad devuelve **Nothing**.
- **Application.ThisWorkbook**: esta propiedad devuelve el libro que contiene la macro que se ejecuta.

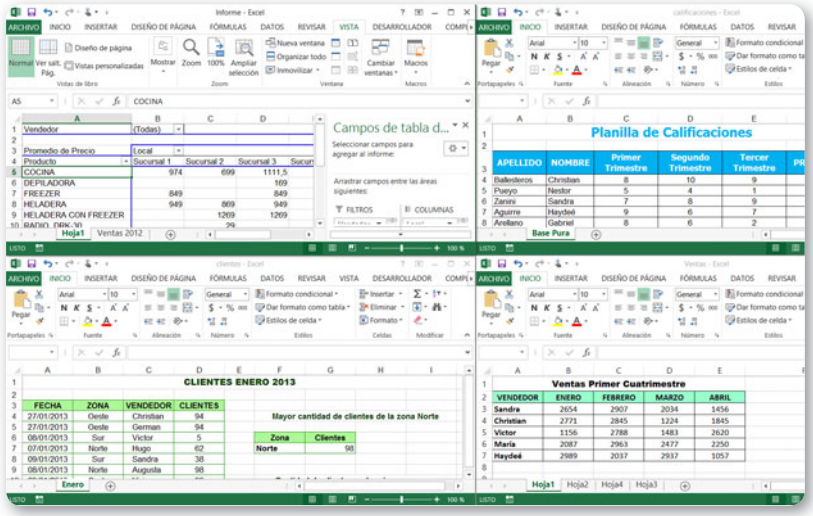


Figura 7. Excel 2013 soporta la Interfaz de Documento Único (SDI). Cada libro tiene su propia ventana de aplicación de nivel superior.

MÁS MÉTODOS

El objeto **Application** posee métodos que nos permiten realizar operaciones relacionadas con la aplicación activa, actuar sobre las fórmulas y cálculos, y sobre las celdas, entre otros. Para conocer más sobre ellos, podemos abrir el **Examinador de objetos** presionando la tecla **F2**.

A continuación, describiremos algunas de las propiedades y métodos de los objetos **Workbooks** y **Workbook**.

Propiedades de los objetos Workbooks y Workbook

Tengamos en cuenta, cuando trabajamos con objetos y colecciones, que muchos objetos de la colección comparten las mismas propiedades y métodos que sus miembros. Dependiendo del objeto que utilicemos, los parámetros que son necesarios o que estén disponibles para estos miembros pueden variar.

Item

Como mencionamos anteriormente, el objeto **Workbook** forma parte de la colección **Workbooks**. Cada libro de la colección puede ser identificado por la propiedad **Item**, que hace referencia a un **único elemento** de la colección. Por lo tanto, podemos acceder a un libro en particular usando un número de índice (**Item**). Una vez que hemos hecho alusión a un objeto **Workbooks** mediante la propiedad **Item**, podemos trabajar con él a través de sus propiedades y métodos. Por ejemplo, para referirnos al tercer libro de la colección, podemos usar la siguiente sentencia:

```
Workbooks.Item(3)
```

O simplemente:

```
Workbooks.(3)
```

ELIMINAR CARPETAS

La instrucción **Kill** solo permite eliminar archivos, no carpetas. Para borrar carpetas, usamos la instrucción **RmDir**. Su sintaxis es **RmDir Path**, donde **Path**, es la cadena de acceso de la carpeta que queremos eliminar. Por ejemplo **RmDir ("C:\Ventas")**. Si omitimos la ruta, VBA intentará eliminar la carpeta actual. Esta instrucción solo elimina carpetas vacías, de lo contrario dará error.

También es posible referirnos a un libro específico de una colección **Workbooks** a través del nombre del archivo, el cual debe estar entre comillas. Por ejemplo, si queremos referirnos al archivo llamado **sueldos.xlsm** tenemos que hacerlo de la siguiente manera:

```
Workbooks("sueldos.xlsm").Activate
```

> LA PROPIEDAD ITEM HACE REFERENCIA A UN ÚNICO ELEMENTO DE LA COLECCIÓN WORKBOOKS

Otra manera de hacer referencia a un libro de Excel es a través de la propiedad **ActiveWorkbook**, del objeto **Application**, que devuelve el libro activo.

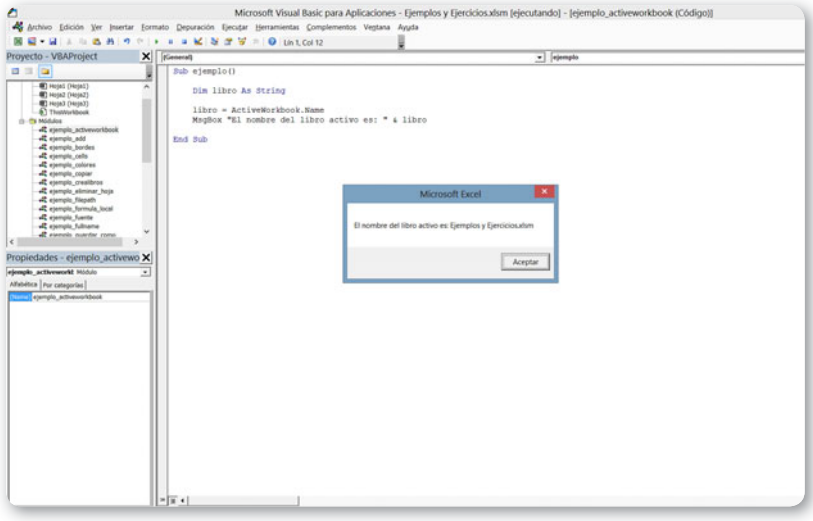


Figura 8. Podemos referirnos al libro activo de Excel a través de la propiedad **`ActiveWorkbook`** del objeto **`Application`**.

Name

Esta propiedad devuelve el nombre de un libro. Por ejemplo, el siguiente procedimiento muestra el nombre del libro activo:

```
Sub ejemplo()
    Dim libro As String
    libro = ActiveWorkbook.Name
    MsgBox "El nombre del libro activo es :" & libro
End Sub
```

Este otro ejemplo muestra el nombre del tercer libro de la colección **Workbooks**.

```
Sub ejemplo()

    Dim libro As String
    libro = Workbooks(3).Name
    MsgBox "El nombre del tercer libro es :" & libro

End SubEnd Sub
```

FullName

Esta propiedad devuelve el nombre y la ruta completa de un libro especificado. Por ejemplo, el siguiente procedimiento muestra el nombre y la ruta del documento activo.

MÁS PROPIEDADES

Los objetos **Workbooks** y **Workbook** poseen otras propiedades como **CreateBackup**, que indica si creamos una copia de seguridad cuando grabamos el archivo, y **Password** que devuelve o define la contraseña necesaria para abrir un libro. Para conocer más acerca de estas y de otras propiedades, podemos consultar la ayuda de VBA presionando la tecla **F1**.

```
Sub ejemplo_fullname()

a = ActiveWorkbook.FullName
MsgBox "El nombre y la ruta del archivo es: " & a

End Sub
```

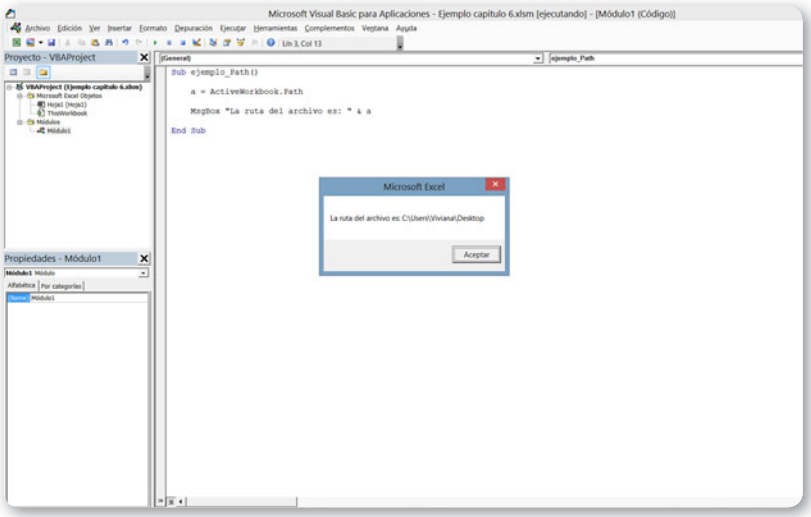


Figura 9. A diferencia de la propiedad **`FullName`**, la propiedad **`Path`** solo devuelve la ruta completa del archivo.

Saved

Esta propiedad devuelve el valor **False** para indicar que se han realizado cambios desde la última vez que se guardó el archivo. En el caso contrario, devuelve **True**. Por ejemplo, el siguiente procedimiento

MSDN

Podemos investigar y conocer aún más acerca de las propiedades, métodos y eventos de los objetos de Microsoft Excel consultando la referencia del modelo de objetos en **Microsoft Developer Network** (**MSDN**) en el sitio de: **http://msdn.microsoft.com/es-es/library/bb149081.aspx.**

muestra un mensaje que indica si se han guardado o no los cambios realizados en el libro de trabajo.

```
Sub ejemplo_Saved()

cambios = ThisWorkbook.Saved

If cambios = False Then

   MsgBox "No se guardaron los cambios en el libro"
Else

   MsgBox "Se guardaron los cambios en el libro"

End If

End Sub
```

Métodos de los objetos Workbooks y Workbook

A continuación, veremos algunos de los métodos que podemos utilizar con los objetos **Workbooks** y **Workbook**.

Add

Este método pertenece a la colección **Workbooks** y permite crear un nuevo libro de Microsoft Excel. Cuando se genera un nuevo libro, VBA crea un nuevo objeto **Workbook** y lo suma a la colección **Workbooks**.

Su sintaxis es:

Workbooks.Add(Template)

Donde **Template** es un argumento opcional que determina cómo se creará el nuevo libro. Este argumento puede ser:

- Una **cadena** que especifica el nombre de un archivo existente de Microsoft Excel para usarlo de plantilla. Cuando utilizamos un archivo de plantilla, lo que hace Excel es copiar todos los elementos de la hoja de cálculo especificada, incluyendo los textos y las macros, en el nuevo libro.
- Una constante **XLWBATemplate** que permite especificar el tipo de libro que se va a crear. Hay cuatro tipos de esta constante:
  - **xlWBATWorksheet**: para crear un libro con una sola hoja de cálculo.
  - **xlWBATChart**: para crear un libro que contiene un gráfico.
  - **xlWBATExcel4IntlMacroSheet**: para crear una hoja de macro Excel 4.0.
  - **xlWBATExcel4MacroSheet**: para crear una hoja internacional de macros.

> ADD ES UN MÉTODO DE WORKBOOKS QUE CREA UN NUEVO LIBRO DE EXCEL

Si no especificamos este argumento, Microsoft Excel crea un libro nuevo con varias hojas en blanco.

El siguiente ejemplo crea una variable (libro) del tipo **Workbook** y luego a esta variable le asignamos el método **Add**.

```
Sub nuevo_libro()

  Dim libro As Workbook

  Set libro = Workbooks.Add

End Sub
```

VENTANA INMEDIATO

La **Ventana Inmediato** dentro del Editor de Visual Basic para Aplicaciones es una herramienta muy útil para explorar objetos, propiedades y métodos. En ella, podemos ejecutar instrucciones VBA y, de esta manera, vemos el resultado en forma inmediata. Para abrirla, simplemente presionamos la combinación de teclas **CRTL + G** o la seleccionamos desde el menú **Ver**.

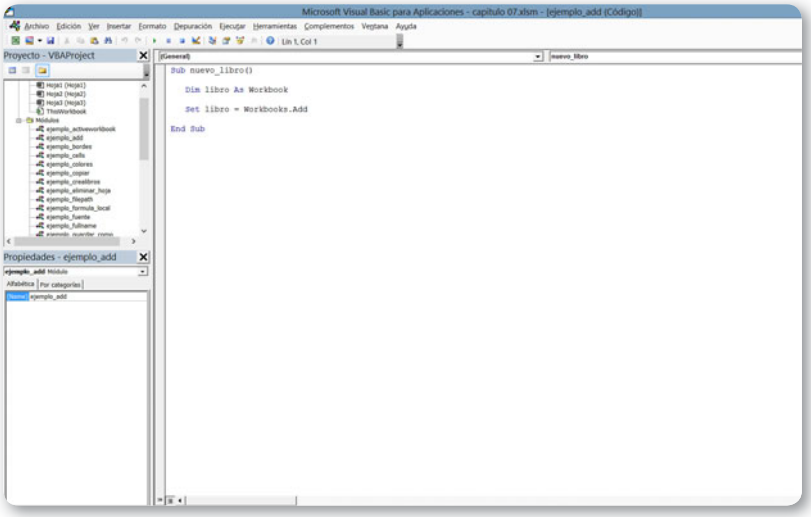


Figura 10. Una vez que le asignamos a una variable el método **Add**, podemos usarla para cambiar las propiedades del libro.

En este ejemplo, creamos tres libros de trabajo:

```
Sub crea_libros()

Dim x As Integer
   For x = 1 To 3
      Workbooks.Add
   Next x

End Sub
```

ÁREA DE TRABAJO

Cuando trabajamos con muchos libros en simultáneo, podemos guardar un **área de trabajo**. En ese caso, Microsoft Excel tendrá en cuenta la posición de todos los libros que están abiertos, al igual que muchas de sus características. Las áreas de trabajo almacenadas tienen la extensión .XLW. En VBA, para hacerlo usamos el método **SaveWorkspace**.

En este ejemplo, creamos un nuevo libro con una hoja de gráfico.

```
Sub nuevo_libro()

  Dim libro As Workbook

  Set libro = Workbooks.Add(xlWBATChart)

End Sub
```

SaveAs

Después de crear un libro, es probable que deseemos guardarlo. Para esto, contamos con el método **Save As**. Su sintaxis más básica es:

Workbook.SaveAs(FileName)

Donde **FileName** es el único parámetro requerido, y se utiliza para especificar el nombre y la ubicación del archivo. En el siguiente ejemplo, creamos un procedimiento que agrega un nuevo libro y lo guarda con el nombre **Ejemplo.xlsm**.

```
Sub GuardarComo1()

  Workbooks.Add

  ThisWorkbook.SaveAs ("Ejemplo.xlsm")

End Sub
```

Su sintaxis completa es la siguiente:

```
Workbook.SaveAs(FileName FileFormat, Password, WriteResPassword,
ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru,
Local).
```

Donde:

- **FileFormat**: contiene la constante **XFileFormat** que indica el formato para guardar el archivo, por omisión es el libro de Excel (**xlNormal**).
- **Password**: como contraseña para abrir el archivo podemos escribir hasta quince caracteres. Distingue entre mayúsculas y minúsculas.
- **WriteResPassword**: contiene la contraseña de restricción de escritura de un archivo.
- **ReadOnlyRecommended**: si esta es **True**, aparece un mensaje que recomienda que el archivo se abrirá como de solo lectura.
- **CreateBackup**: crea una copia de seguridad del archivo.
- **AccessMode**: contiene un valor constante **xlNoChange**, **XlExclusive** o **slShared**, que indica el modo de acceso.
- **ConflictResolution**: contiene una constante que indica la forma de resolver conflictos, **xlLocalSessionChanges** acepta cambios del usuario local, **xlOtherSessionChanges** acepta cambios de otros usuarios.
- **AddToMru**: predeterminado como **False**. Si se establece como **True**, se agrega el libro a la lista de archivos usados recientemente.
- **Local**: guarda los archivos en el idioma de Microsoft Excel si es **True**. Si es **False**, los archivos se guardan en el idioma VBA.

En este ejemplo, el procedimiento crea un nuevo libro y lo guarda como **Ventas.xlsm** en **Mis Documentos**. Le asignamos una contraseña:

```
Sub Ejemplo_GuardaComo()

Dim ruta As String, nombre As String

ruta = "C:\Users\Pepe\Documents\"
nombre = "Ventas.xlsm"

ActiveWorkbook.SaveAs ruta & nombre, _
    FileFormat:= xlOpenXMLWorkbookMacroEnabled, _
    Password:="123", _
    ReadOnlyRecommended:=False

End Sub
```

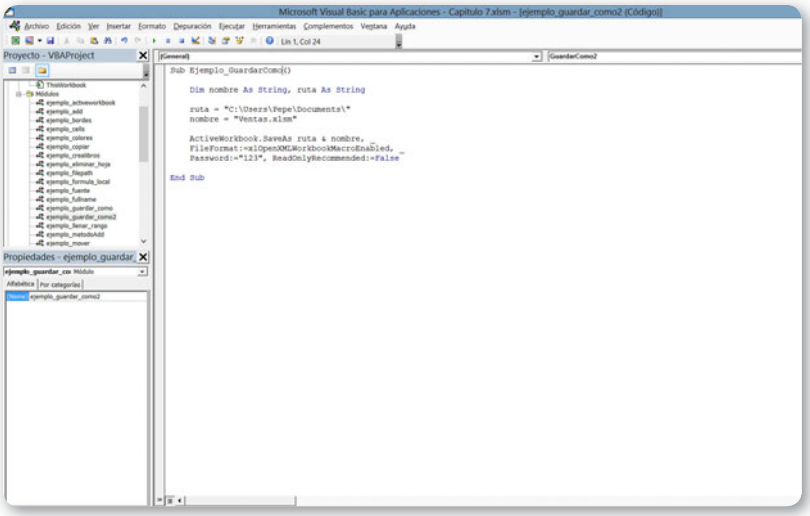


Figura 11. El formato **`xlOpenXMLWorkbookMacroEnabled`** especifica que el archivo se guardará en el formato **`Libro de Open XML`** con macros habilitadas.

Save

Utilizamos el método **Save** para guardar los diferentes cambios que hemos realizado en un libro después de haberlo guardado por primera vez. Su sintaxis es la siguiente:

```
Workbook.Save
```

La siguiente sentencia guarda el libro activo:

```
ThisWorkbook.Save
```

OPCIONES PARA CERRAR

El método **Close** cierra todos los libros de trabajo, pero Microsoft Excel sigue abierto. En cambio, si usamos el método **Quit** del objeto **Application**, este también cerrará Excel. Si tenemos libros abiertos, **se** cerrarán primero los libros, y, si alguno contiene cambios, la aplicación nos pedirá que los guardemos. Siempre y cuando no hayamos usado la propiedad **DisplayAlerts**.

Si conocemos el nombre del archivo, por ejemplo **Ventas.xlsm**, podemos usar la siguiente sintaxis:

```
Workbooks("Ventas.xlsm").Save
```

Close

Este método aplicado al objeto **Workbooks** permite cerrar un libro, y no acepta ningún argumento. Su sintaxis es:

```
Workbooks.Close
```

Podemos hacer referencia a un libro de la colección **Workbooks**, a través de su nombre o su índice. Por ejemplo, si tenemos tres libros abiertos y queremos cerrar el segundo, empleamos la sentencia:

```
Workbooks(2).Close
```

El siguiente ejemplo cierra el libro **Ventas.xlsm**:

```
Workbooks("Ventas.xlsm").Close
```

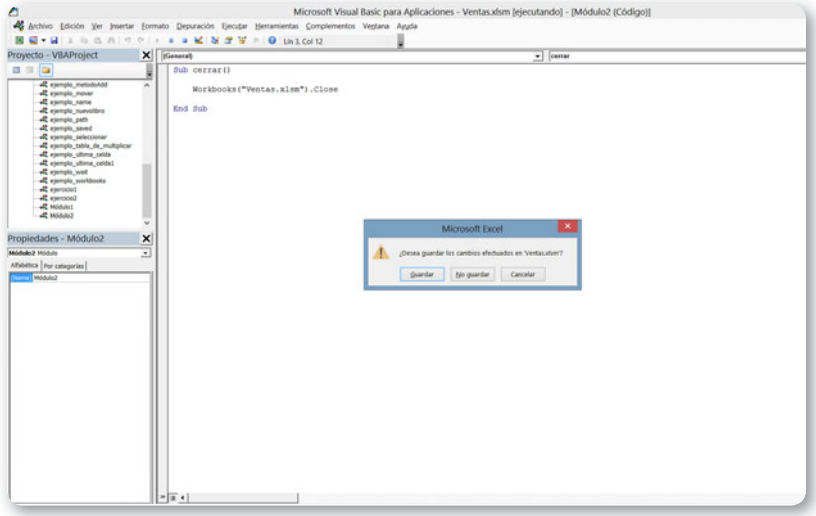


Figura 12. Si no guardamos los cambios en el libro, al ejecutar esta sentencia nos preguntará si deseamos hacerlo.

En el caso de que no especifiquemos ningún libro, todos los libros que tengamos abiertos se cerrarán.

En cambio, el método **Close** del objeto **Workbook** puede aceptar hasta tres argumentos, por lo tanto, nos va a permitir cerrar un libro en particular. Su sintaxis es la siguiente:

```
Workbook. Close(SaveChanges, Filename,_
        RouteWorkbook)
```

Donde:

- **SaveChanges**: guarda los cambios del libro que se cierra, si se especifica el valor **True**. En cambio, si se indica el valor **False**, el libro se cierra sin guardar los cambios realizados.
- **Filename**: podemos especificar un nombre para guardar el archivo, siempre que le hayamos indicado el valor **True** a **SaveCanges**.
- **RouteWorkbook**: este parámetro se emplea cuando necesitamos distribuir el libro entre otros usuarios para trabajar en grupo.

EL MÉTODO CLOSE DEL OBJETO WORKBOOKS PERMITE CERRAR UN LIBRO

La sintaxis que presentamos a continuación cierra el libro que se encuentra activo. Si, con anterioridad, no se han guardado los cambios realizados, se mostrará un cuadro de diálogo que nos pregunta si deseamos hacerlo antes de cerrar el libro.

ThisWorkbook.Close

BORRAR UN ARCHIVO

La instrucción **Kill** elimina un archivo de Excel o de cualquier otro tipo. Su sintaxis es **Kill Path&Filename**. Por ejemplo: **KillFile = "C:\Planilla\Ventas.xlsx"**. Podemos eliminar uno o varios archivos a la vez usando los símbolos de comodín: el asterisco (*****) para reemplazar una secuencia de caracteres o el signo de interrogación (**?**) que reemplaza un solo carácter.

La siguiente sintaxis permite cerrar el libro activo y abre el cuadro de diálogo **Guardar como**:

```
ThisWorkbook.Close SaveChanges:=True
```

Tengamos en cuenta que, si no especificamos el parámetro **SaveChange**, Microsoft Excel comprobará cada libro de trabajo para asegurarse de que se han guardado desde su última modificación. Si el libro contiene cambios, Excel nos pedirá que los guardemos.

Open

Empleamos este método para abrir un archivo de Microsoft Excel. Cada vez que abrimos un libro, este se agregará a la colección **Workbooks**. Al igual de lo que sucede con el comando **Archivo/Abrir** de la cinta de opciones de Excel, el libro que abramos será el libro activo. Su sintaxis más básica es la siguiente:

> EL LIBRO QUE ABRAMOS CON EL MÉTODO OPEN PASARÁ A SER EL LIBRO ACTIVO

```
Workbooks.Open(FileName)
```

Donde **FileName** es el único parámetro requerido, e indica el nombre y la ruta del archivo por abrir. Si solo proporcionamos el nombre del archivo, Excel lo buscará en la carpeta actual.

Por ejemplo, para abrir el archivo **Ventas.xlsm** que se encuentra en la carpeta actual, escribimos la siguiente sentencia:

```
Workbooks.Open ("Ventas.xlsm")
```

GETOPENFILE

En lugar de especificar en el código el archivo que deseamos abrir, podemos mostrar el cuadro de diálogo **Abrir** a través del método **GetOpenFilename**. Al utilizar este método, cuando el usuario hace clic en el botón **Aceptar**, el nombre del archivo seleccionado se almacena en una variable, y esta variable se le asigna al método **Open**.

Si en cambio queremos abrir el archivo **Liquidacion.xlsm** que se encuentra en la carpeta **Planillas** del disco **C**, escribimos la sentencia:

Workbooks.Open ("C:\Planillas\Liquidacion.xlsm")

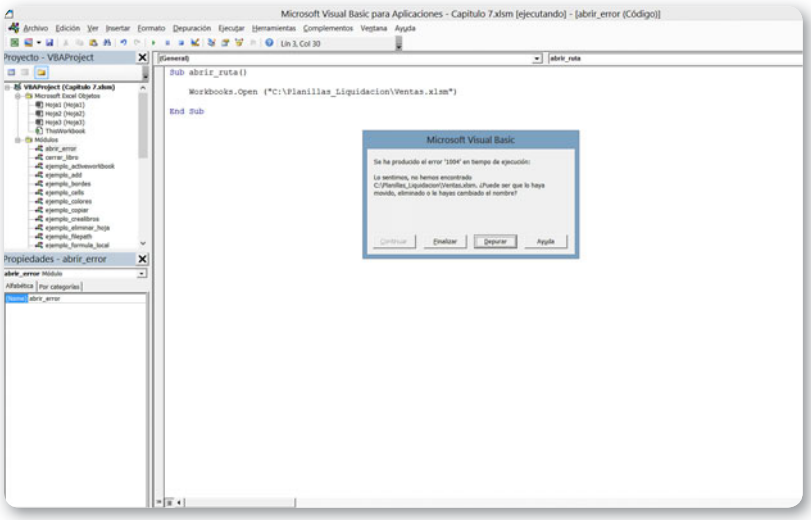


Figura 13. Cuando Excel no encuentra el archivo que queremos abrir, presenta un mensaje de error.

Este método posee dieciséis parámetros diferentes que determinan la manera en que Microsoft Excel va a abrir el libro. La sintaxis completa de este método es la siguiente:

```
Workbooks.Open(FileName, UpdateLinks, ReadOnly, Format, Password,
WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable,
Notify, Converter, AddToMru, Local, CorruptLoad)
```

Describiremos a través de ejemplos solo alguno de ellos. Si queremos abrir un archivo como de solo lectura, usaremos el argumento **ReadOnly**, asignándole un valor **True**.

```
Workbooks.Open Filename:="Ventas.xlsm", _
        ReadOnly:=True
```

Para abrir un archivo que está protegido con una contraseña, utilizamos el argumento **Password**, como muestra la siguiente sentencia:

```
Workbooks.Open Filename:="Ventas.xlsm", _
        Password:="clave"
```

Para agregar el archivo a la lista de usados recientemente, tenemos el argumento **AddToMru**, como vemos en la sentencia:

```
Workbooks.Open Filename:="Ventas.xlsm", _
        Addtomru:=True
```


Worksheet

Como sabemos, un libro de Excel contiene hojas de cálculo que podemos insertar, borrar, copiar o desplazar a cualquier lugar que deseemos. Como así también, renombrarlas, seleccionarlas y editarlas en conjunto, es decir, introducir un dato y aplicar formatos al mismo tiempo a todas las celdas de varias hojas de cálculo.

En Visual Basic para Aplicaciones, una hoja de cálculo es un objeto del tipo **Worksheet**, y las diferentes hojas de cálculo que vamos a utilizar se almacenan en la colección **Worksheets**.

Propiedades del objeto Worksheet

El objeto **Worksheet** se encuentra debajo del objeto **Workbook** en la jerarquía del modelo de objetos de Excel. A continuación veremos algunas de sus propiedades.

Item

Cuando abrimos un archivo de Microsoft Excel, por defecto se crea un libro de trabajo que contiene una hoja de cálculo. Para hacer referencia a una determinada hoja de la colección, utilizamos la

propiedad **Item**. Por ejemplo, para referirnos a la segunda hoja del libro activo, escribimos la siguiente sentencia:

Worksheets.Item(2).Activate

O simplemente:

Worksheets(2).Activate

PARA HACER REFERENCIA A UNA HOJA DE UN LIBRO, USAMOS LA PROPIEDAD ITEM

También podemos referirnos a una hoja de cálculo en concreto de una colección **Worksheets** a través de su nombre, el cual debe estar entre comillas. Por ejemplo, hacemos referencia a la hoja **Norte** de la siguiente manera:

Worksheets("Norte").Activate

En el caso de que necesitemos hacer referencia a la hoja activa, usamos la siguiente sentencia:

ActiveSheet

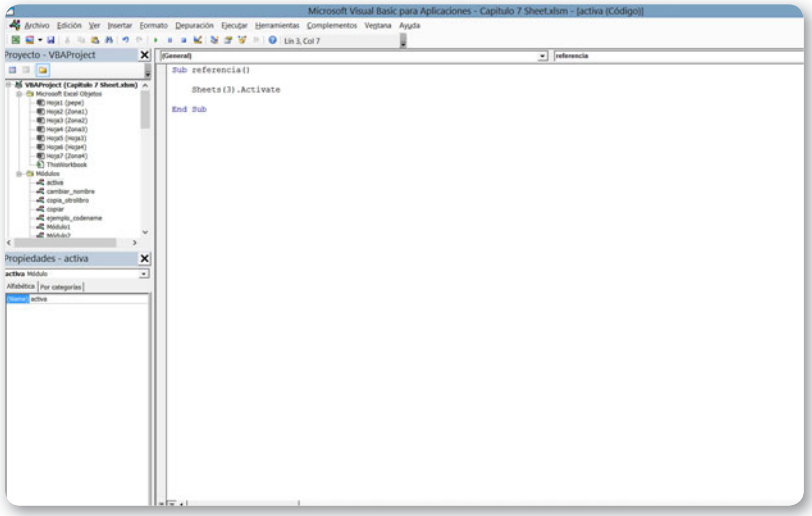


Figura 14. La colección **Worksheets** también recibe el nombre de **Sheets**.

Name

Esta propiedad se emplea para establecer o devolver el nombre de una hoja de cálculo. Por ejemplo, tenemos el siguiente procedimiento que muestra el nombre de la **Hoja3**.

```
Sub nombre()

    MsgBox Worksheets(3).Name

End Sub
```

Si queremos cambiar el nombre de la **Hoja1** por **Enero**, podemos utilizar la siguiente sentencia:

Worksheets("hoja1").Name = "Enero"

También podríamos usar la siguiente sentencia:

Sheets("hoja1").Name = "Enero"

CodeName

Cuando en el código de una macro hacemos referencia a la hoja de cálculo por el nombre de la pestaña o por su índice, corremos el riesgo de generar un error, porque el nombre de las hojas o su posición en el libro pueden variar. Una forma de salvaguardar el código que se refiere a hojas específicas es usando la propiedad **CodeName**. Esta devuelve el nombre clave que Microsoft Excel le asigna a la hoja de cálculo cuando

REFERENCIA A UNA HOJA DE CÁLCULO

Podemos declarar una variable del tipo **Worksheet** para obtener la referencia a una hoja de cálculo. La inicializamos mediante la propiedad **Item** y la asignamos a la variable mediante el operador **Set**. Por ejemplo, para asignar a la variable **Ene** la hoja **Enero** del libro **Ventas.xlsm**, escribimos lo siguiente: **Set Ene = Workbooks("Ventas.xlsm").Worksheets("Enero").**

la crea (**Hoja1**, **Hoja2**, **Hoja3**, etcétera). La siguiente sentencia devuelve el nombre clave de la **Hoja1** y activa la **Hoja3**.

```
MsgBox Worksheets(1).CodeName
Hoja3.Activate
```

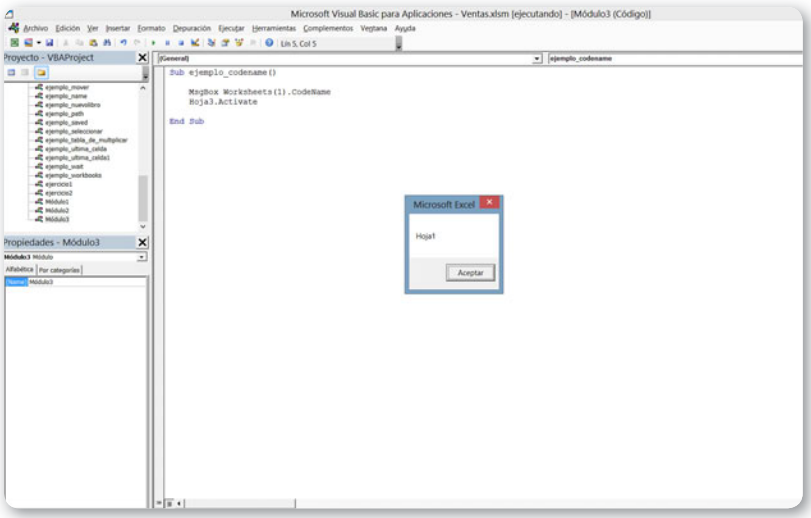


Figura 15. En la ventana del **Explorador de proyectos**, los **CodeName** son los que figuran fuera de los paréntesis.

Visible

Esta propiedad permite mostrar u ocultar una hoja de cálculo. Por defecto, tiene el valor **True**; si le asignamos el valor **False** ocultamos la hoja de cálculo. Por ejemplo, si queremos ocultar la hoja **Enero**, escribimos la siguiente sentencia:

```
Worksheets("Enero").Visible = False
```

LA PROPIEDAD VISIBLE PERMITE MOSTRA U OCULTAR UNA HOJA DE CÁLCULO

Con esta sentencia, el usuario puede volver a visualizar esta hoja empleando el comando **Ocultar y mostrar** del grupo **Celdas/Formato** que se encuentra en la ficha **Inicio** de la interfaz de Microsoft Excel.

Si en cambio, queremos ocultar la hoja para que la única forma de hacerla visible sea a través del código, empleamos la sentencia:

```
Worksheets("Enero").Visible = xlSheetVeryHidden
```

En este caso, para volver a mostrar esta hoja, tenemos que asignarle el valor **True** a la propiedad **Visible**.

Count

Esta propiedad de la colección **Worksheets** y **Sheets** devuelve el número de elementos que hay en la colección actual. Por ejemplo, si queremos saber cuántas hojas tiene un libro, utilizamos la siguiente sentencia:

```
MsgBox Worksheets.Count
```

UsedRange

Cuando necesitamos saber cuál es el rango utilizado en una hoja de cálculo, usamos la propiedad **UsedRange** del objeto **Worksheet**. La siguiente sentencia selecciona el rango utilizado en la hoja **Enero**.

```
Worksheets("Enero").UsedRange.Select
```

Métodos de los objetos Worksheets y Worksheet

A continuación, explicaremos algunos de los diferentes métodos de los objetos **Worsheets** y **Worksheet**.

NOMBRES DE LAS HOJAS

Si bien podemos ponerles cualquier nombre a las hojas, el que utilicemos determinará el ancho de la pestaña, en la parte inferior de la ventana del libro de trabajo. Por lo tanto, debemos dar a las hojas nombres concisos, de tal forma que podamos ver más de dos o tres solapas a la vez.

Add

Este método del objeto **Worksheets** o **Sheets** permite agregar hojas de cálculo, de gráfico o de macros a un libro de Excel.

Su sintaxis es la siguiente:

Objeto.Add(Before, After, Count, Type)

Donde:

- **Objeto**: puede ser un objeto **Worksheets** o **Sheets**.
- **Before**: indica que la hoja se agrega antes de la hoja activa. Si se omite este parámetro o el parámetro **After**, la nueva hoja se agregará antes de la hoja que se encuentra activa.
- **After**: indica que la hoja se agrega después de la hoja activa.
- **Count**: permite indicar la cantidad de hojas que vamos a agregar. Si omitimos este parámetro, solamente se agregará una hoja.
- **Type**: especifica el tipo de hoja que queremos crear. Si se trata de una hoja de cálculo, usamos el valor **xlWorksheet**; si en cambio queremos crear una hoja de gráfico, usamos el valor **xlChart**. Para crear una hoja de macros, usamos el valor **xlExcel4MacroSheet** o **xlExcel4IntlMacroSheet**. Si se omite este argumento se creará una hoja de cálculo.

> EL MÉTODO ADD DE WORKSHEETS AGREGA HOJAS DE CÁLCULO, DE GRÁFICO O DE MARCROS

Por ejemplo, el procedimiento que presentamos a continuación agrega una nueva hoja de cálculo después de la hoja **Febrero** y modifica el nombre de la nueva hoja por **Marzo**.

```
Sub metodoAdd()

   Worksheets.Add after:=Worksheets("Febrero")
   ActiveSheet.Name = "Marzo"

End Sub
```

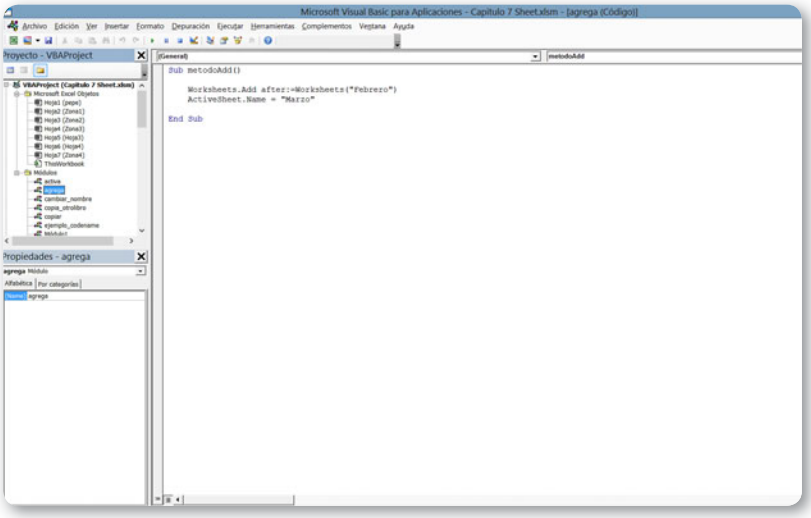


Figura 16. La cantidad de hojas de un libro está condicionada por nuestra necesidad y la memoria de la computadora.

Move

Podemos organizar las hojas de un libro de Excel usando el método **Move** del objeto **Sheets** o **Worksheets**. Cuando movemos una hoja, debemos indicar la nueva ubicación especificando el nombre de la hoja que deseamos colocar antes o después de la hoja activa.

Su sintaxis es la siguiente:

```
Objeto.Move(Before, After)
```

Donde:

- **Objeto**: puede ser un objeto **Worksheets** o **Sheets**.
- **Before**: se utiliza para indicar la hoja delante de la cual se va a mover la hoja seleccionada.
- **After**: permite indicar la hoja después de la cual se desea mover la hoja seleccionada.

Por ejemplo, si queremos mover la hoja denominada **Abril** para ubicarla después de la última hoja del libro, podemos utilizar la sentencia que mostramos a continuación:

```
Sub mover()
  ultima = Worksheets.Count
  Worksheets("Abril").Move After:=Worksheets(ultima)

End Sub
```

Si no especificamos los argumentos **Before** o **After**, Excel creará un libro nuevo y moverá la hoja a ese libro.

Copy

Este método permite copiar una hoja de cálculo especificada a otra ubicación dentro del mismo libro o en otro libro.
Su sintaxis es la siguiente:

Objeto.Copy(Before, After)

Donde:

- **Objeto**: es la hoja que queremos copiar.
- **Before**: indica la hoja delante de la cual vamos a copiar la hoja seleccionada.
- **After**: se utiliza para indicar la hoja después de la cual se desea copiar la hoja seleccionada.

> EL MÉTODO COPY COPIA UNA HOJA DENTRO DE UN MISMO LIBRO O EN OTRO DISTINTO

El siguiente procedimiento crea tres copias de la hoja **Zona1**, las ubica a continuación de esta (la hoja **Zona1** tiene índice 1) y las renombra como **Zona2**, **Zona3** y **Zona4**.

COUNT

Tengamos en cuenta que, si movemos una hoja de cálculo antes o después de una hoja inexistente, se provocará un error. Para evitar estos errores, especialmente cuando usamos índices para hacer referencia a una hoja específica, debemos usar la propiedad **Count** que determina la cantidad de hojas del libro. De esta manera, nos aseguramos de que el índice que usamos es válido.

```
Sub copiar()

For i = 2 To 4

 Worksheets("Zona1").Copy After:=Worksheets(i - 1)
 Worksheets(i).Name = "Zona" & i

Next

End Sub
```

Si omitimos los parámetros **Before** o **After**, VBA creará un nuevo libro con la hoja copiada. Por ejemplo, la siguiente sentencia copia la hoja **Zona1** a un nuevo libro:

```
Worksheets ("Zona1").Copy
```

Delete

Este método elimina hojas de cálculo, de gráfico y de macros. Si la hoja de cálculo contiene datos, Excel mostrará un cuadro de diálogo que solicita que confirmemos si realmente queremos eliminar la hoja.

Su sintaxis es la siguiente:

```
Objeto.Delete
```

Donde **Objeto** puede ser un objeto **Worksheets** o **Sheets**.

Por ejemplo, si queremos eliminar la hoja **Norte** del libro activo, que ocupa la segunda posición en la colección, podemos usar una de las siguientes sintaxis:

```
Worksheets("Norte").Delete
Worksheets(2).Delete
```

Si **Norte** es la hoja activa, podemos usar la siguiente sentencia:

```
ActiveSheet.Delete
```

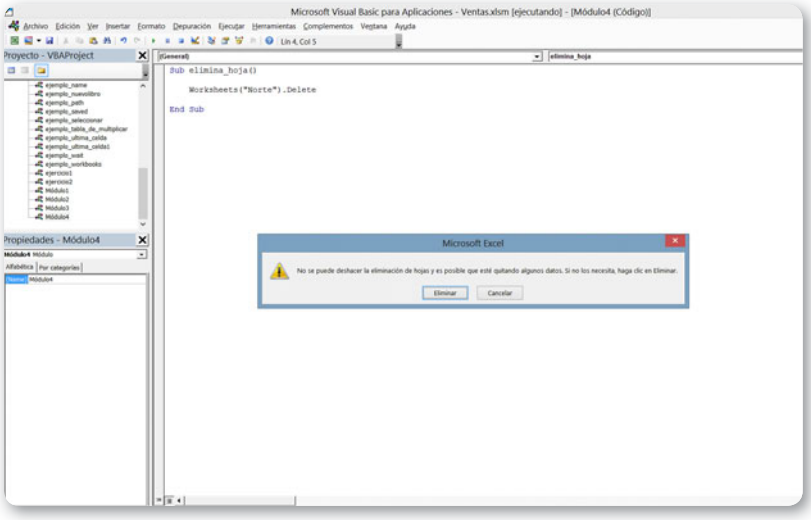


Figura 17. Cuando eliminamos una hoja, esta acción se realiza de manera permanente.

Range

El objeto **Range** se encuentra debajo del objeto **Worksheet** y puede ser una celda, una fila, una columna o un conjunto de celdas de una misma hoja o de otras hojas (rango 3D).

Propiedades de los objetos Range

El objeto **Range** tiene una amplia variedad de propiedades con las cuales podemos trabajar; a continuación, describiremos algunas de ellas.

PROTEGER HOJAS

Las hojas de cálculo pueden contener información confidencial o datos que no deseamos que sean modificados. VBA, además de la protección mediante contraseñas para los libros, permite proteger los datos de una determinada hoja. Usamos el método **Protect** para proteger una hoja y **Unprotec** para desprotegerla. Si queremos proteger la hoja activa, usamos la sentencia **ActiveSheet.Protect.**

ActiveCell

Esta propiedad del objeto **Application** y **Windows** devuelve la celda activa de la ventana activa o seleccionada.

Su sintaxis es la siguiente:

```
Objeto.ActiveCell
```

Donde **Objeto** puede ser **Application** o **Windows**. Por ejemplo, la siguiente sentencia muestra el contenido de la celda activa.

```
MsgBox Application.ActiveCell
```

Range

Esta propiedad se utiliza para devolver un objeto **Range** que representa un rango de celdas. Su sintaxis es:

```
Objeto.Range (rango)
```

Donde

- **Objeto**: puede ser un objeto **Application**, **Range** o **Worksheet**.
- **rango**: es una referencia de rango o nombre introducido como texto (**string**).

Por ejemplo, tenemos la siguiente sentencia que selecciona el rango **A1:E1** de la hoja que se encuentra activa:

```
Range("A1:E1").Select
```

CELDA ACTIVA

Llamamos **celda activa** a la celda que ha sido seleccionada. Cuando seleccionamos una única celda, esta se vuelve activa. Si, en cambio, tenemos seleccionado un rango, la celda activa (**Activecell**) será la primera del extremo superior izquierdo o desde donde parta la selección. Podemos identificarla como la celda resaltada en la selección.

Si, en cambio, queremos indicar un rango no contiguo, utilizamos la coma como operador de unión. Por ejemplo, para el rango **A1:A5** y **C1:C5**, escribimos la siguiente sentencia:

```
Range("A1:A5, C1:C5").Select
```

También es posible hacer referencia a un rango por su nombre. Por ejemplo, si queremos hacer referencia al rango denominado **costo** utilizamos la siguiente sentencia:

```
Range("costo").Select
```

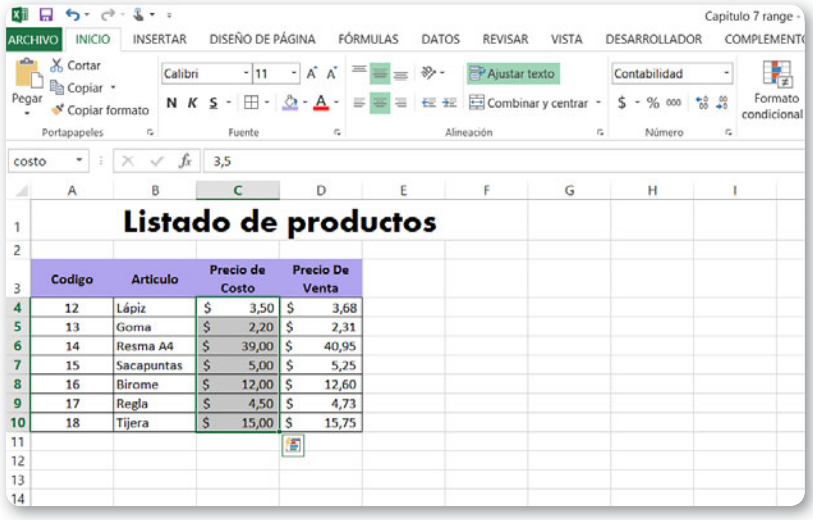


Figura 18. El nombre del rango en VBA debe ponerse entre comillas, a diferencia del uso de los rangos con nombre en las fórmulas dentro de una misma hoja.

Cells

También podemos hacer referencia a celdas específicas de una hoja de cálculo utilizando la propiedad **Cells**, que devuelve un objeto **Range**, el cual puede ser todas las celdas de la hoja de cálculo o un rango de celdas. Su sintaxis es la siguiente:

```
Objeto.Cells(índice_fila, índice_columna)
```

Donde:

- **Objeto**: puede ser un objeto **Application**, **Range** o **Worksheet**.
- **índice_fila**: es el número de fila de la celda. Si el objeto es una hoja de cálculo, **indice_fila** se refiere a la fila **1** de la hoja; en cambio, si el objeto es un rango, se refiere a la primera fila del rango.
- **índice_columna**: puede ser una letra o un número que representa la columna de la celda. Si el objeto es una hoja de cálculo, **indice_columna B** o **2** se refiere a la columna **B** de la hoja. En cambio, si el objeto es un rango, **B** o **2** se refiere a la segunda columna del rango.

Por ejemplo, para referirnos a la celda **D4**, podemos escribir las siguientes sentencias:

```
Cells(4,"D")
```

```
Cells (4,4)
```

En este ejemplo, creamos un procedimiento que muestra la tabla de multiplicar del 2 en el rango **A1:A10**:

```
Sub ejemplo_cells()
 For i = 1 To 10
Cells(i, 1).Value = "2 X " & i & " = " & 2 * i
 Next
End Sub
```

Rows

Esta propiedad devuelve un objeto **Range** que hace referencia a una fila de un objeto **Application**, **Range** o **Worksheet**. Su sintaxis es:

```
Objeto.Rows (indice)
```

Donde:

- **Objeto**: puede ser un objeto **Application**, **Range** o **Worksheet**.
- **indice**: es el número de fila. Si el objeto es una hoja de cálculo, índice **1** se refiere a la fila **1** de la hoja; en cambio, si el objeto es un rango de celdas, entonces índice **1** se refiere a la primera fila del rango. Este argumento es opcional.

Por ejemplo, si queremos seleccionar la fila **1** de una hoja de cálculo, escribimos la siguiente sentencia:

```
Rows("1").Select
```

Columns

Esta propiedad devuelve un objeto **Range** que hace referencia a una columna de un objeto **Application**, **Range** o **Worksheet**. Su sintaxis es:

```
Objeto.Rows (indice)
```

Donde

- **Objeto**: puede ser un objeto **Application**, **Range** o **Worksheet**.
- **indice**: es el número de fila. Si el objeto es una hoja de cálculo, índice **1** se refiere a la columna **A** de la hoja; en cambio, si es un rango de celdas, entonces índice **1** se refiere a la primera columna del rango. Este es un argumento opcional.

> LA PROPIEDAD COLUMNS DEVUELVE UN OBJETO QUE HACE REFERENCIA A UNA COLUMNA

OTRAS PROPIEDADES

Las propiedades **EntireRow**, **EntireColumn** y **CurrentRegion** son útiles para realizar operaciones que permiten ampliar un rango. **EntireRow** representa una o más filas del rango especificado, y **EntireColumn**, una o más columnas. **CurrentRegion** devuelve el rango delimitado por cualquier combinación de filas y columnas en blanco.

Por ejemplo, para seleccionar la columna **C** de una hoja de cálculo, escribimos la siguiente sentencia:

```
Columns("C").Select
```

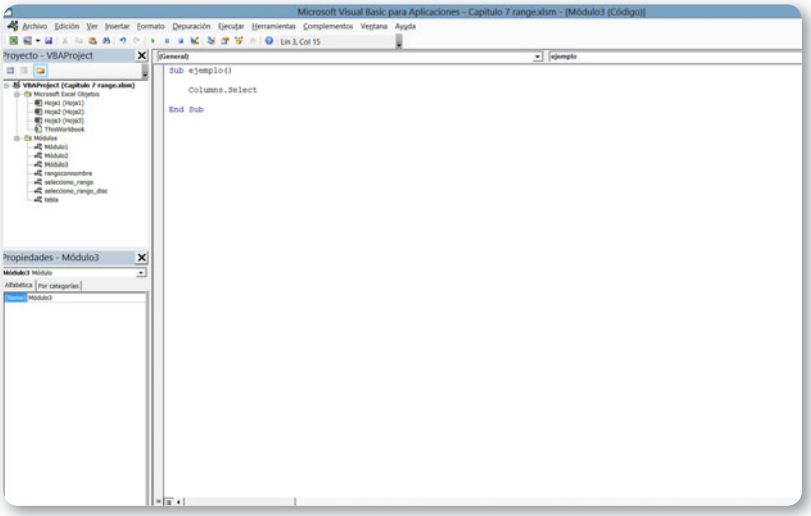


Figura 19. Si en la propiedad **Columns** omitimos el **indice**, devuelve todas las filas de la hoja de cálculo.

Offset

Esta propiedad devuelve un objeto **Range** que está desplazado de un rango especificado por un cierto número de filas y columnas. Su sintaxis es la siguiente:

```
Range.Offset ([fila_offset],[columna_offset])
```

Donde

- **fila_offset**: es el número de fila para desplazar. Si el número es positivo, desplaza el rango hacia abajo; si es negativo, lo desplaza hacia arriba y, si es cero, usa la misma fila.
- **columna_offset**: es el número de columna para desplazar. Si el número es positivo, desplaza el rango hacia la derecha; si es negativo, lo desplaza hacia la izquierda y, si es cero, usa la misma columna.

Por ejemplo, si queremos mover dos celdas a la izquierda de la celda activa, usamos la siguiente sentencia.

```
ActiveCell.Offset (0,-2).Select
```

Value

Esta propiedad permite establecer o devolver el valor que tiene una celda especificada dentro de una hoja de cálculo.

Si la celda elegida se encuentra vacía, entonces **Value** devuelve el valor **Empty**.

Su sintaxis es la siguiente:

```
Expresión.Value
```

> LA PROPIEDAD VALUE DEVUELVE EL VALOR QUE TIENE UNA CELDA ESPECIFICADA

Donde **Expresión** representa un objeto **Range**.

A continuación, veamos algunos ejemplos que permiten asignar un valor a una celda específica en la hoja activa. Por ejemplo, a la celda **A5** le asignamos el valor 56:

```
Range("A5").Value = 56
```

En este ejemplo, insertamos la fecha y hora del sistema a la celda activa de la hoja activa:

```
ActiveCell.Value = Now
```

Si queremos introducir el mismo valor en muchas celdas de la hoja activa, por ejemplo en el rango **C1:E5**, agregamos la palabra **Hola**:

```
Range("C1:E5").Value = "Hola"
```

Para introducir un valor en una celda de una hoja de trabajo específica, por ejemplo, en la celda **A2** de la hoja **Gastos**, ingresamos el texto **Comunes** de la siguiente manera:

```
Worksheets("Gastos").Range("A2").Value = "Comunes"
```

Si necesitamos introducir en una celda un texto entre comillas dobles, debemos triplicar las comillas dobles. Por ejemplo, en la celda **B1**, ingresamos el texto **Sandra** entre comillas.

Range("B1").Value = """Sandra"""

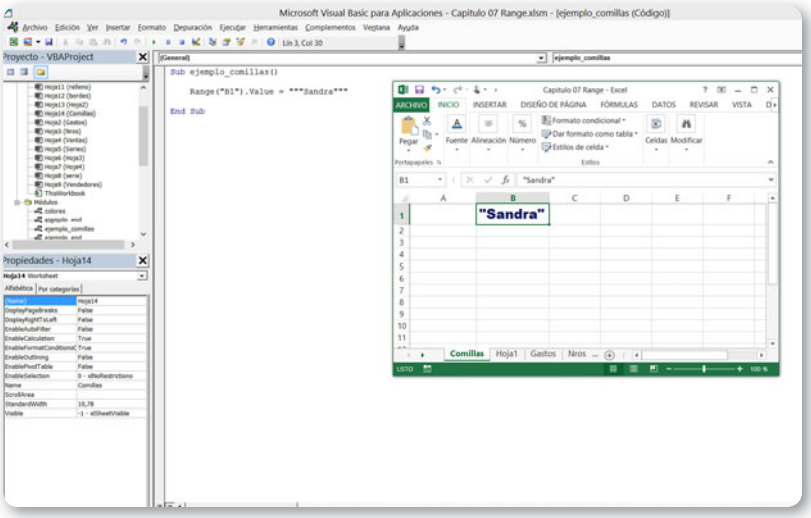


Figura 20. No es necesario activar una celda para introducir un valor en ella, desde cualquier parte de la hoja se puede escribir.

Si queremos llenar el rango **A1:J10** con valores consecutivos, podemos usar el siguiente procedimiento:

```
Sub ejemplo()
   n = 1
   For F = 1 To 10
      For c = 1 To 10
         Sheets("Nros").Cells(F, c).Value = n
         n = n + 1
      Next c
   Next F
End Sub
```

FormulaLocal

Esta propiedad permite ingresar fórmulas y funciones en nuestro propio idioma, cualquiera que este fuera, en las celdas de Excel mediante VBA, de la misma manera que la escribiríamos dentro de una celda de Excel

Su sintaxis es la siguiente:

Expresión.FormulaLocal = "formula"

Donde

> LA PROPIEDAD FORMULALOCAL PERMITE INGRESAR FUNCIONES EN NUESTRO IDIOMA

- **Expresión**: representa un objeto **Range**.
- **Formula**: es una fórmula o función ingresada como texto (**String**).

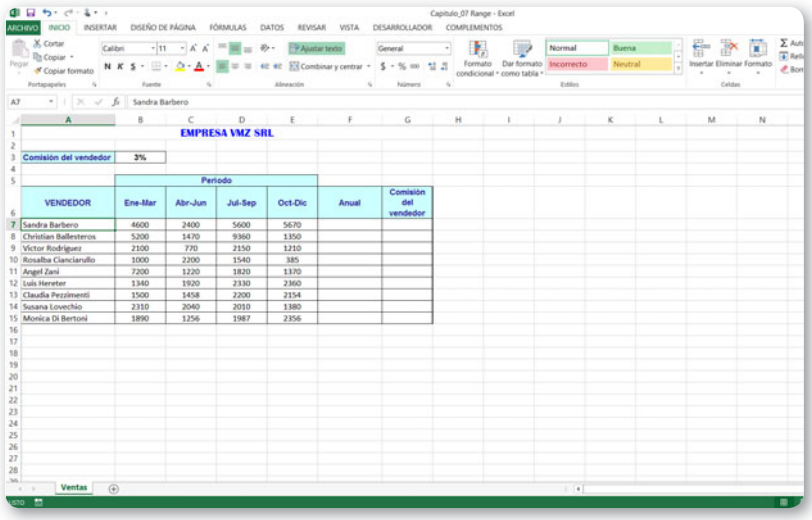


Figura 21. Al escribir la función en nuestro propio idioma, la macro solo será compatible con otro Excel en nuestro idioma.

Supongamos que tenemos una planilla de ventas, como se muestra en la **Figura 21**, en la que debemos calcular:

- **Total anual**: es igual a la suma de los períodos por cada vendedor.
- **Comisión del vendedor**: es igual al total anual por el porcentaje de comisión del vendedor.

Escribimos el siguiente procedimiento:

```
Sub ejemplo_formula_local()

    Range("F7:F15").FormulaLocal = "=Suma(B7:E7)"
    Range("G7:G15").FormulaLocal = "=F7*B$3"

End Sub
```

End

Esta propiedad del objeto **Range** sirve para determinar la dirección a la cual se extenderá una selección de rango. Su función es desplazar el cursor a la celda que esté, a partir de la celda activa, en dirección hacia arriba (**xlUp**), abajo (**xlDown**), derecha (**xlToRight**) o izquierda (**xlToLeft**), dentro del rango actual, es decir, aquel que está delimitado por una fila o columna en blanco. Su sintaxis es:

Objeto.End(Direction)

Donde **objeto** representa a un objeto **Range**. Por ejemplo, para desplazarnos desde la celda **A2** hasta la última celda con datos (no vacía) hacia abajo, usamos la siguiente sentencia:

Range("A7").End(xlDown).Select

Si, por ejemplo, queremos encontrar la primera fila libre a partir de la celda **A7**, usamos el siguiente procedimiento:

RANGE.FORMULA

Cuando usamos esta propiedad, debemos escribir el nombre de la función en inglés. De este modo, nuestra macro será compatible con cualquier versión de Excel aunque esté en un idioma diferente al que usamos. Por ejemplo, si queremos sumar el rango de celdas **B7:E7**, debemos ingresar la sentencia **Range ("F7:F15").Formula = "=Sum(B7:E7)"** .

```
Sub ultima_celda()
   Worksheets("Ventas").Select
   filalibre = Range("A7").End(xlDown).Offset(1, 0).Row
   MsgBox filalibre

End Sub
```

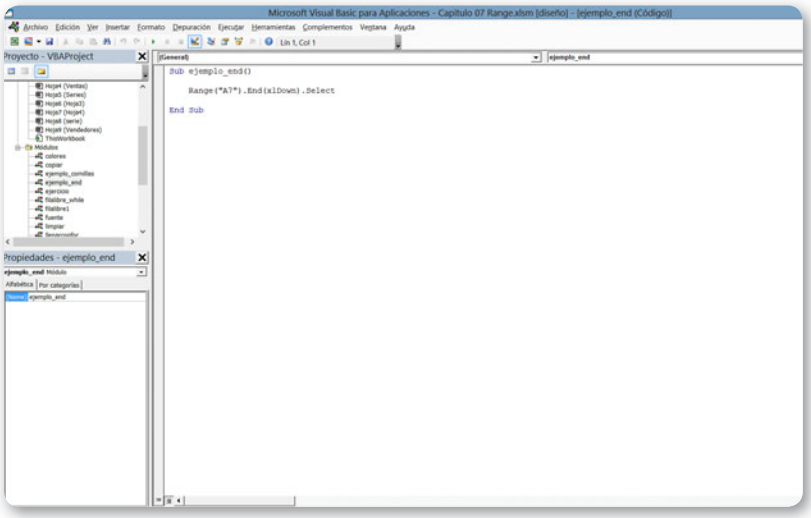


Figura 22. Esta propiedad equivale a presionar las teclas **FIN + FLECHA ARRIBA**, **FIN + FLECHA ABAJO**, **FIN + FLECHA IZQUIERDA** o **FIN + FLECHA DERECHA**.

Font

Esta propiedad del objeto **Range** sirve para establecer el tipo de fuente para los datos contenidos en una celda o un grupo de ellas.

Su sintaxis es la siguiente:

Objeto.Font

Donde **objeto** es la variable que representa al objeto **Range**.

Por ejemplo, si queremos cambiarle al rango **A1:A10** el tipo de letra a **Comic Sans Ms**, cuerpo **10** y **Negrita**, tenemos que escribir el procedimiento que mostramos a continuación:

```
Sub fuente()

    For i = 1 To 10

      With Cells(i, 1).Font
         .Name = "Comic Sans Ms"
         .Size = 10
         .Bold = True
       End With

 Next i

End Sub
```

Interior

Esta propiedad permite establecer el color del interior de una celda. Su sintaxis es la siguiente:

Objeto.Interior

Por ejemplo, el procedimiento que presentamos a continuación muestra en la columna **A** el valor **indice** correspondiente a la paleta **ColorIndex** y, en la columna **B**, pinta el interior de las celdas con todos los valores correspondientes a la paleta **ColorIndex**.

```
Sub colores()

   Dim i As Integer

   With Worksheets("relleno")

      For i = 1 To 56

         Select Case i
            Case Is < 15
```

```
            .Cells(i, 1).Interior.ColorIndex = i
            .Cells(i, 2).Value = " = " & i
        Case Is < 29
            .Cells(i - 14, 3).Interior.ColorIndex = i
            .Cells(i - 14, 4).Value = " = " & i
        Case Is < 43
            .Cells(i - 28, 5).Interior.ColorIndex = i
            .Cells(i - 28, 6).Value = " = " & i
        Case Is < 57
            .Cells(i - 42, 7).Interior.ColorIndex = i
            .Cells(i - 42, 8).Value = " = " & i
    End Select

  Next i

 End With

End Sub
```

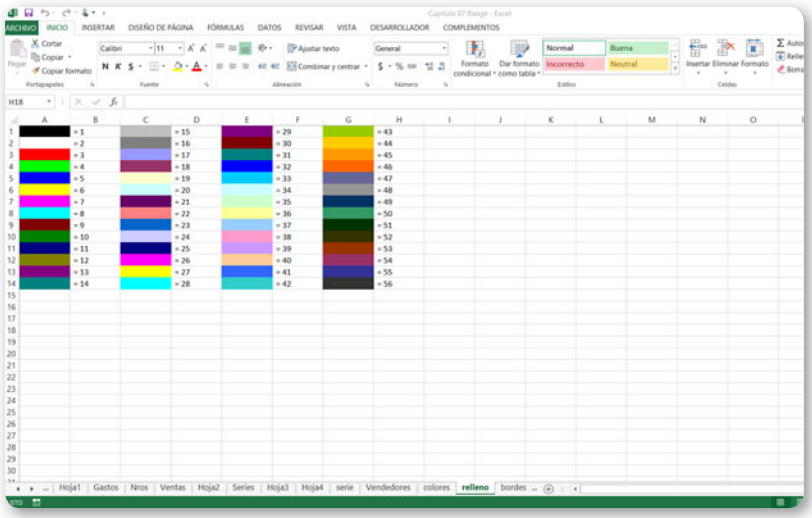


Figura 23. Al pintar el fondo de una celda en rojo, asignamos a la propiedad **`ColorIndex`** del interior un valor **3**.

Border

Esta propiedad permite aplicar bordes a una celda o un rango de celdas. Su sintaxis es la siguiente:

Objeto.Borders

Por ejemplo, este procedimiento aplica un borde grueso, estilo entrecortado y color azul al rango **A1:D5**.

```
Sub bordes()

Worksheets("bordes").Select

 With Range("A1:D5").Borders

    .LineStyle = xlDash
    .ColorIndex = 41
    .Weight = 3

  End With
End Sub
```

Métodos del objeto Range

A continuación, explicaremos algunos métodos del objeto **Range**.

Select

Este método permite seleccionar una celda o un conjunto de ellas. Su sintaxis es la siguiente:

Objeto.Select

Donde **Objeto** representa a un objeto **Range**. Por ejemplo, si queremos seleccionar una celda, usamos la siguiente sentencia:

Range("A1").Select

Para seleccionar un rango indicado por variables, podemos utilizar el procedimiento que mostramos a continuación:

```
Sub seleccionar()

   columna = 5
   rango = ("B2" & ":E" & columna)
   Range(rango).Select

End Sub
```

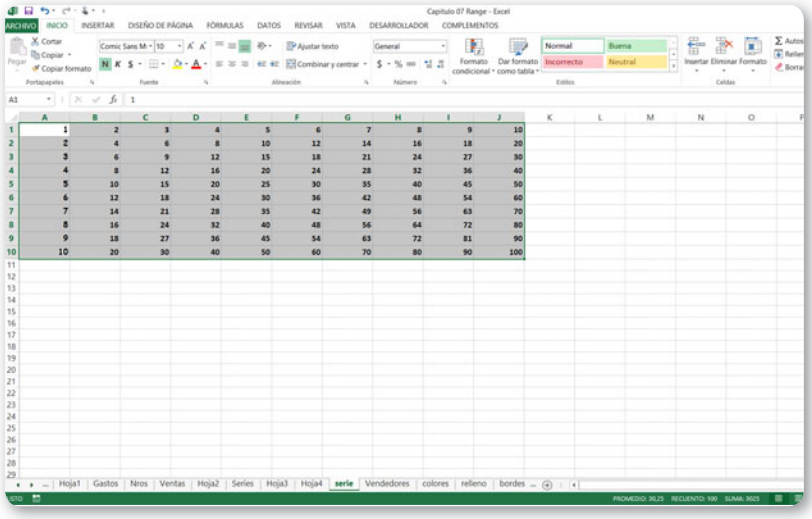


Figura 24. Mediante la propiedad **`CurrentRegion`**, podemos seleccionar el rango que contiene datos.

COLORES RGB

Excel emplea una paleta de 56 colores RGB predefinidos, que podemos usar como relleno de celdas, color de bordes y de fuente. El código **RGB** se basa en la mezcla de tres colores (red, green, blue) para conseguir toda la gama completa. El valor de cada uno de los 56 colores puede ser cualquiera de los 16 millones de colores disponibles, pero en Excel solo podemos usar 56 colores.

Para seleccionar la última celda con datos a partir de la celda **A7**, usamos el siguiente procedimiento:

```
Sub ultima_celda()
    Worksheets("Ventas").Select
    Range("A7").Select
    While ActiveCell.Value <> ""
        ActiveCell.Offset(1, 0).Select
    Wend
    filalibre = ActiveCell.Row
End Sub
```

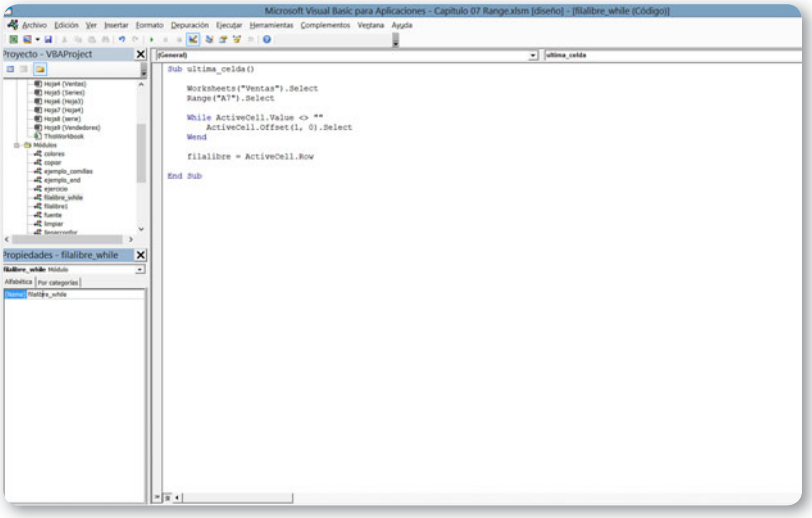


Figura 25. Usamos **`ActiveCell.Offset`** para desplazarnos una fila hacia abajo, si la celda activa contiene datos.

MOVER

El método **Cut** del objeto **Range** permite cortar el contenido de una celda o un conjunto de celdas y pegarlo en el Portapapeles, para luego utilizarlo, o en un destino especificado. Por ejemplo, si necesitamos mover el contenido del rango **A1:A6** al rango **C1:C6** de la hoja de cálculo activa, tenemos que escribir la sentencia **Range("A6:A15").Cut Range("C6:C15")**.

DataSeries

Este método permite introducir una serie de datos en un rango de celdas. Su sintaxis es la siguiente:

Objeto.DataSeries(Rowcol, Type, Date, Step, Stop, Trend)

Donde:

- **Objeto**: es el rango que vamos a usar para la serie de datos.
- **RowCol**: para introducir datos en una fila, usamos la constante **xlRows** y, para introducir datos en columnas, usamos la constante **xlColumns**. Si se omiten estas constantes, VBA utiliza el tamaño y la forma del rango.
- **Type**: es el tipo de serie. Por defecto es **xlLinear** (serie lineal), pero también puede ser: **xlGrowht** (serie geométrica), **xlChronological** (serie cronológica) o **xlAutoFill** (autorrelleno).
- **Date**: si hemos usado la constante **xlChronological** para el argumento **Type**, permite especificar diferentes intervalos para las series de fecha. Sus opciones son **xlDay**, **xlWeekday**, **xlMonth** o **xlYear**.
- **Step**: permite especificar el valor de incremento de la serie, por omisión este valor es 1.
- **Stop**: para especificar el valor final de la serie. Si este valor se omite, entonces Excel rellena el rango.
- **Trend**: indica la tendencia. Si el valor es **True**, crea una serie lineal o geométrica. Si el valor es **False**, crea una serie de datos estándares. Por omisión, este valor es **False**.

Por ejemplo, usando el método **DataSeries**, podemos crear una tabla de multiplicar como mostramos a continuación:

```
Sub TablasMultiplicar()
  Worksheets("serie").Activate
  For Columna = 1 To 10
     Cells(1, Columna) = Columna
     Set Rango = Range(Cells(1, Columna), Cells(10, Columna))
     Rango.DataSeries Step:=Columna
  Next
End Sub
```

Copy

Este método permite copiar celdas y rangos. Su sintaxis es la siguiente:

Objeto.Copy(Destination)

Donde **Objeto** es el rango que vamos a usar para la serie de datos. Por ejemplo, para copiar el contenido de la celda **A6** en la celda **B6** de la hoja activa, usamos la siguiente sentencia:

Range("A6").Copy Range("B6")

Para copiar un rango en una hoja de cálculo distinta de la hoja activa, tenemos esta sentencia:

```
Worksheets("Ventas").Range("A6:A15").Copy _
    Worksheets("Vendedores").Range("A6:A15")
```

ClearContents

Este método borra el contenido de las celdas, conservando el formato de ellas. Su sintaxis es la siguiente:

Objeto.ClearContents

Por ejemplo, para eliminar el contenido del rango de celdas **A7:G15**, usamos la siguiente sentencia:

Range("A7:G15").ClearContents

USEDRANGE

La propiedad **UsedRange** del objeto **Range** devuelve un rango rectangular que se extiende desde la celda superior izquierda del rango utilizado (la primera) hasta la celda inferior derecha usada (la última del rango). Por ejemplo, si la celda **A1** tiene la palabra **Ventas** y la celda **C6** contiene el número **40**, el rango utilizado que devolverá esta propiedad será el rango **A1:C6**.

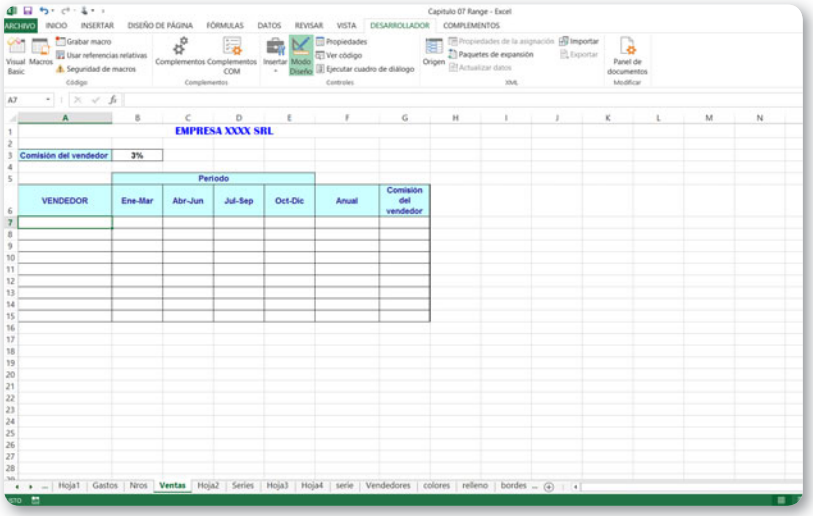


Figura 26. Para eliminar el contenido y el formato de un grupo de celdas, usamos el método **`Clear`**. Por ejemplo: **`Range("A7:G15").Clear`**.

RESUMEN

En este capítulo, realizamos una breve descripción de los principales objetos del modelo de objetos de Microsoft Excel: Application, Workbook, Worksheet y Range. Aprendimos a trabajar con algunas de sus propiedades y métodos a través de diversos ejemplos que nos ayudarán a crear nuestros propios procedimientos. Este es solo un comienzo, tengamos en cuenta que existen muchas otras propiedades y métodos relacionados con estos objetos.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Para qué sirve la propiedad **DefaultFilePath** del objeto **Application**?
2. ¿Para qué sirve la propiedad **DisplayAlerts** del objeto **Application**?
3. ¿Qué permite hacer el método **Quit** del objeto **Application**?
4. ¿Qué devuelve la propiedad **Thisworkbook**?
5. ¿Para qué sirve la propiedad **Name** del objeto **Workbook**?
6. ¿Cuál es el método del objeto **Workbook** que me permite crear un nuevo libro de Excel?
7. ¿Para qué sirven los métodos **Save** y **SaveAs** del objeto **Workbook**?
8. ¿Qué permite hacer el método **Close** del objeto **Workbook**?
9. ¿Para qué se puede emplear la propiedad **Visible** del objeto **Worksheets**?
10. ¿Qué objeto devuelve la propiedad **Range**?

EJERCICIOS PRÁCTICOS

1. Inserte un nuevo módulo y cámbiele la propiedad **Name** por **ejercicio_1**.
2. Escriba el procedimiento del archivo **Cap7_actividad02.doc**, que se encuentra en el sitio **http://www.redusers.com/premium/notas_contenidos/macrosexcel2013/**.
3. Cree un botón en la hoja de cálculo, asígnele el procedimiento creado y ejecútelo.
4. Inserte un nuevo módulo y cámbiele la propiedad **Name** por **ejercicio_2**.
5. Escriba el procedimiento del archivo **Cap7_actividad05.doc**, que se encuentra en el sitio **http://www.redusers.com/premium/notas_contenidos/macrosexcel2013/**.

*08

Formularios

En este capítulo, explicaremos cómo diseñar formularios usando el Editor de VBA para crear cuadros de diálogo personalizados, que nos permitirán generar una interfaz amigable para interactuar con el usuario. Además, conoceremos algunos controles, con sus propiedades y eventos, y aprenderemos a programarlos.

Formularios

Los **formularios**, llamados también cuadros de diálogos personalizados, son ventanas editables que contienen objetos de menor jerarquía, tales como botones, cuadros de textos, textos estáticos y lista de opciones, entre otros. Se los emplea para mostrar información y para permitirle al usuario introducir o seleccionar contenido. Se les puede asociar una combinación de código VBA y de datos para responder a las acciones del usuario. Un libro de trabajo puede tener cualquier cantidad de formularios.

Insertar un formulario

Para agregar un formulario a un proyecto, desde la ventana del Editor de Visual Basic para Aplicaciones hacemos clic en el menú **Insertar** y, luego, seleccionamos la opción denominada **UserForm**.

> A TRAVÉS DE UN FORMULARIO, EL USUARIO PUEDE INGRESAR O SELECCIONAR DATOS

Al insertar el primer formulario a un proyecto, veremos que en el **Explorador de proyectos** se crea una carpeta llamada **Formularios**, que contiene el elemento **UserForm1**. Esta carpeta tendrá todos los formularios que generemos para nuestro libro.

En el sector derecho de la ventana del Editor de VBA, aparecerá un formulario vacío y, como una ventana flotante, veremos el **Cuadro de herramientas** que agrupa los diferentes controles que podemos incorporar al formulario. Estos controles son los que permiten interactuar con los usuarios; más adelante los veremos en detalle.

USERFORMS EN OFFICE

Los **UserForms** no son exclusivos de Microsoft Excel, sino que forman parte de la biblioteca de objetos de Visual Basic para Aplicaciones, y, por lo tanto, los tenemos disponibles para usarlos en todas las aplicaciones del paquete Microsoft Office, como, por ejemplo, en Word o en PowerPoint.

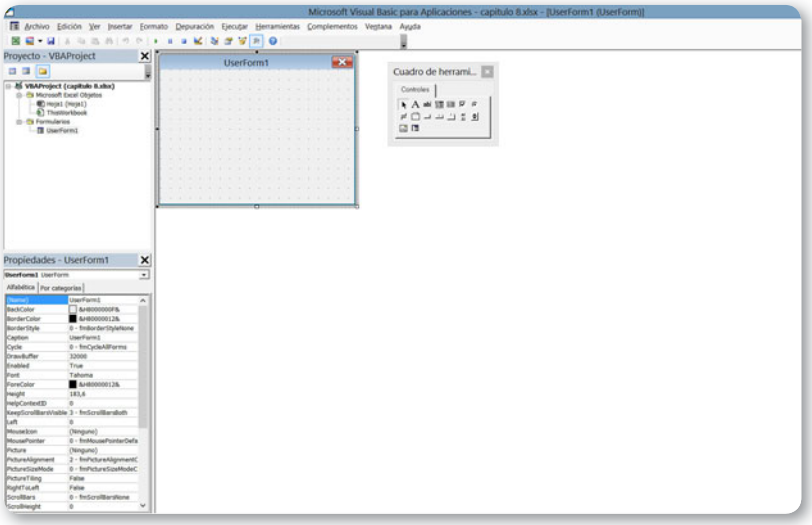


Figura 1. También podemos insertar un formulario pulsando el botón **`Insertar UserForm`** de la barra de herramientas Estándar.

El formulario es un objeto **UserForm** y, como cualquier otro objeto de VBA, tiene sus propios eventos, propiedades y métodos con los que podemos controlar su apariencia y su comportamiento.

Propiedades de los formularios

El primer paso para diseñar formularios consiste en establecer sus propiedades. Si la ventana **Propiedades** no se encuentra visible en el Editor de VBA, para acceder a ella pulsamos la opción **Ventana Propiedades** del menú **Ver** o hacemos un clic con el botón derecho del mouse sobre el formulario y, en el menú, seleccionamos **Propiedades**. Debemos tener presente que una sola propiedad puede pertenecer a

ESTABLECER PROPIEDADES

Es posible establecer las propiedades de un formulario tanto en tiempo de diseño, mientras lo estamos armando, desde la ventana **Propiedades** del Editor de VBA, como en tiempo de ejecución, al escribir el código, es decir, mientras se ejecuta realmente la aplicación y se interactúa con ella. No hay una forma mejor que la otra, sino que la elección dependerá de nuestro gusto.

diferentes objetos. Por ejemplo, es posible definir un tipo y tamaño de letra para los cuadros de texto, los botones de comando y las etiquetas, entre otros elementos disponibles.

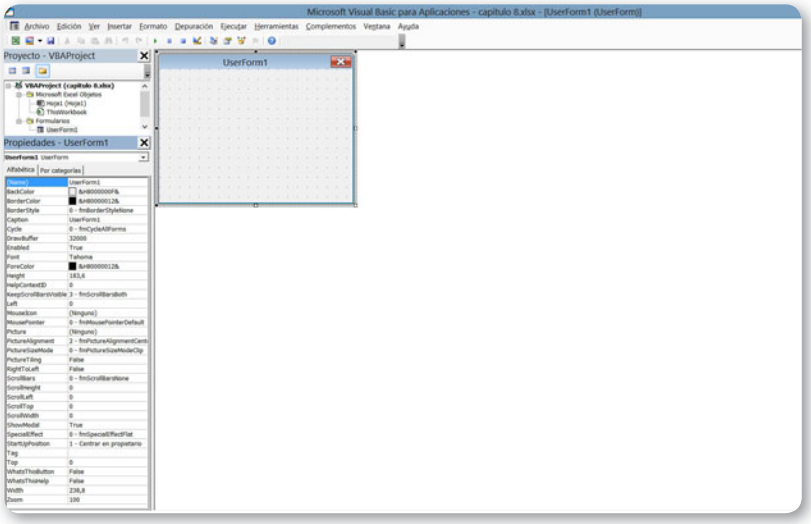


Figura 2. La ventana **Propiedades** tiene dos fichas que presentan las propiedades por categoría o en orden alfabético.

A continuación, describiremos algunas de las propiedades más comunes de los formularios.

- **Name**: permite establecer el nombre con el que haremos referencia al formulario en el código. Cuando agregamos un formulario, VBA, de manera predeterminada, le asigna el nombre **UserForm1**, **UserForm2**, **UserForm3**, y así sucesivamente. Es conveniente darle a esta propiedad un valor más significativo, como, por ejemplo, **frmEntrada**, para un formulario de entrada de datos.

COLORES

Cuando seleccionamos la propiedad **BackColor**, veremos el valor **&H8000000F&**, el cual está codificado. Esto se debe a que Visual Basic describe los códigos de color utilizando un código hexadecimal (base 16). El empleo de códigos hexadecimales, en teoría, permite la definición de hasta **16.772.216** colores diferentes.

- **Caption**: permite ingresar un texto personalizado que se mostrará en la barra de título del formulario.
- **Height**: permite establecer la altura del formulario en puntos.
- **Width**: permite determinar el ancho del formulario en puntos
- **BackColor**: permite elegir el color de fondo del formulario. Para seleccionar un color diferente al predeterminado, debemos desplegar el menú del cuadro de valores mediante un clic sobre su contenido y luego sobre la flecha de la derecha. Veremos que este menú contiene dos fichas: **Sistema** y **Paleta**. La ficha **Sistema** muestra una paleta de colores que Windows generalmente utiliza para sus distintos objetos. En cambio, la ficha **Paleta** presenta todos los colores que se encuentran disponibles para el fond de los formularios. Esta propiedad también la podemos modificar desde el código. Su sintaxis es **Objeto.BackColor [=Long]**. Donde **Objeto** es cualquier objeto válido, y **Long** es un argumento opcional que determina el valor numérico que representa un color válido.

> LA PROPIEDAD BACKCOLOR PERMITE CAMBIAR EL COLOR DE FONDO DEL FORMULARIO

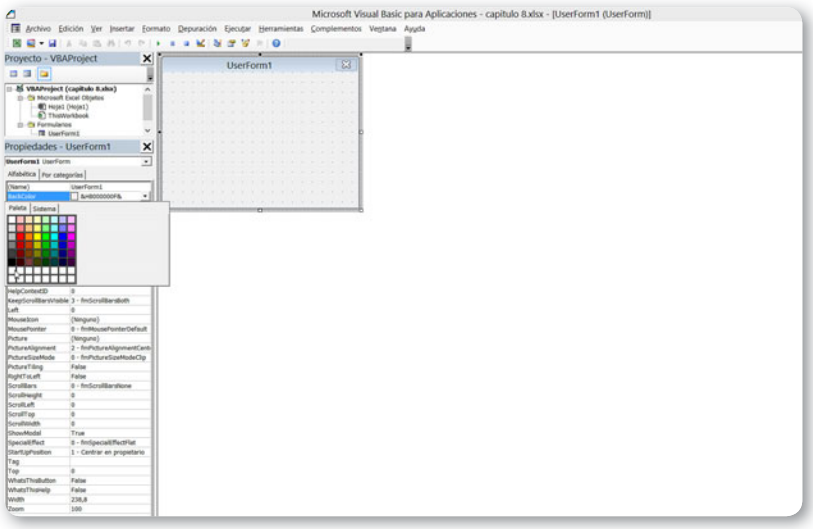


Figura 3. Haciendo un clic con el botón derecho del mouse sobre los casilleros en blanco de la ficha **Paleta**, podemos crear nuestros propios colores.

- **BorderStyle**: permite establecer un tipo de borde para el formulario. Tiene dos opciones: **0 - fmBorderStyleNone** para un control sin línea, y **1 - fmBorderStyleSingle** para un control con una línea. La segunda opción es la predeterminada para el objeto formulario.
- **BorderColor**: permite definir el color del borde; se emplea junto a la propiedad **BorderStyle.**
- **StartUpPosition**: permite determinar la ubicación exacta del formulario en la pantalla, como, por ejemplo, centrado.
- **Picture**: permite incluir una imagen de fondo para el formulario.

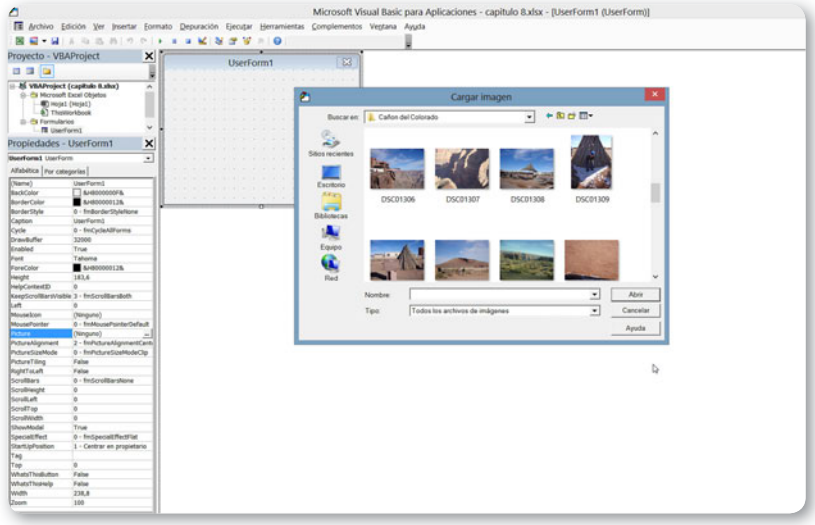


Figura 4. Haciendo un clic en el botón **...** del cuadro de valores de la propiedad **Picture**, podremos seleccionar un archivo de imagen.

- **PictureAlignment**: permite alinear la imagen dentro del formulario. Las opciones disponibles son: **0 - fmPictureAlignmentTopLeft** alinea la imagen en la parte superior izquierda del formulario; **1 - fmPictureAlignmentTopRight** alinea la imagen en la parte superior derecha del formulario; **2 - fmPictureAlignmentCenter** centra la imagen en el formulario (es la opción predeterminada); **3 - fmPictureAlignmentBottomLeft** alinea la imagen en la parte inferior izquierda del formulario y **4 - fmPictureAlignmentBottomRight** alinea la imagen en la parte inferior derecha del formulario.
- **PictureSizeMode**: permite ajustar el tamaño de la imagen. Tiene diferentes opciones disponibles: **0 - fmPictureSizeModeClip**

mantiene el tamaño original de la imagen. Si este supera el tamaño del formulario, solo se mostrará una parte de la imagen; **1 - fmPictureSizeModeStretch** autoajusta la imagen, en dirección vertical y horizontal, para que entre en el formulario. Esta opción distorsionará la imagen; **3 - fmPictureSizeModeZoom** autoajusta la imagen al tamaño del formulario, pero, a diferencia de la opción anterior, no distorsiona la imagen.

Métodos de los formularios

A continuación, vamos a conocer los distintos métodos que nos permiten manejar los formularios.

- **Show**: muestra el formulario en pantalla. Para esto, Visual Basic para Aplicaciones primero comprueba que el formulario esté cargado en la memoria y, si no lo está, procede a cargarlo. La sintaxis básica es **NombreFormulario.Show**. Por ejemplo, tenemos el siguiente procedimiento que muestra el formulario llamado **frmEjemplo**.

```
Sub mostrar_formulario()
   frmEjemplo.Show
End Sub
```

- **Load**: carga el formulario en memoria, pero no lo hace visible. Su sintaxis es **Load NombreFormulario**.
- **Hide**: oculta el formulario, sin descargarlo de la memoria. Si bien los controles no estarán al alcance del usuario, podemos seguir haciendo referencia a ellos a través del código. Esto quiere decir que los valores de las variables a nivel formulario no desaparecen, sino que permanecen ocultas. Su sintaxis es **NombreFormulario.Hide**. Por ejemplo, el procedimiento que presentamos a continuación oculta el formulario llamado **frmEjemplo**.

```
Sub mostrar_formulario()
   frmEjemplo.Hide
End Sub
```

- **Unload**: descarga el formulario de la memoria, y, de esta manera, todos los controles volverán a los valores que tenían por defecto. Su sintaxis es **Unload NombreFormulario**.

Eventos de los formularios

A continuación, veremos los principales eventos de un formulario.

- **Initialize**: se produce al cargar por primera vez el formulario, a través del método **Show** o **Load**. Esto significa que, si en una aplicación cargamos un formulario por primera vez, se va a realizar este evento, pero, si descargamos el formulario y luego lo volvemos a cargar, esta segunda vez y las sucesivas no se producirá este evento.
- **Activate**: ocurre cuando el formulario se vuelve activo, es decir, cada vez que se hace visible (método **Show**). Esto sucede múltiples veces si tenemos en la aplicación más de un formulario, pero solo uno de ellos puede ser el formulario activo.

Controles de un formulario

La gran mayoría de los formularios que diseñemos se utilizará para que los usuarios puedan ingresar y seleccionar datos de una manera fácil, por eso, es necesario que contengan **controles**. Microsoft Excel posee una gran variedad de controles que podemos agregar a un formulario. Para esto, usaremos el **Cuadro de herramientas** que aparece al insertar o abrir un formulario y que contiene los controles más utilizados. Primero elegimos el control que necesitamos, luego lo seleccionamos y lo arrastramos al formulario.

ME

Me es una palabra reservada de acceso directo que se refiere al objeto **UserForm** que contiene el código. Al escribirla seguida por un punto (**.**), veremos automáticamente todas las propiedades para el formulario. Por ejemplo, para ocultar el formulario activo podemos utilizar la siguiente sentencia **Me.Hide**.

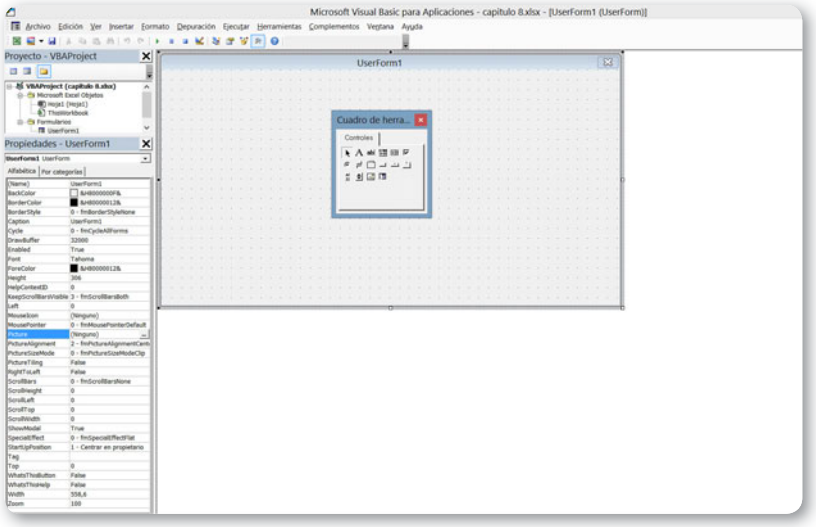


Figura 5. El editor de VBA carece de comandos de menú para agregar controles al formulario.

Si el **Cuadro de herramientas** no se encuentra visible, debemos ir al menú **Ver** y seleccionar la opción **Cuadro de herramientas** o presionar el botón **Cuadro de herramientas** de la barra de herramientas Estándar.

A continuación, describiremos las principales propiedades, métodos y eventos de los controles más comunes.

Etiquetas (Label)

El control **Etiqueta** (**Label**) se utiliza para mostrar información que los usuarios no pueden modificar y para identificar a otros controles, como, por ejemplo, los cuadros de texto y las barras de desplazamiento.

> EN EL CUADRO DE HERRAMIENTAS DE LOS FORMULARIOS ENCONTRAMOS LOS CONTROLES

Propiedades

Al igual que en los formularios, es posible establecer las propiedades de cualquier control en tiempo de diseño, a través de la ventana **Propiedades**, o en tiempo de ejecución, asignando la propiedad en el código. Las principales propiedades del control **Etiqueta** son:

- **Name**: permite establecer el nombre con el que haremos referencia a la etiqueta en el código. Por omisión, se le asigna el nombre **Label1.**
- **Caption**: el contenido que muestra una etiqueta es asignado por esta propiedad. Por ejemplo, si a la etiqueta llamada **lbl_titulo** le queremos asignar el texto **"Ejemplo Etiqueta"** en tiempo de ejecución, usamos la siguiente sentencia:
  lbl_titulo.Caption = "Ejemplo Etiqueta".

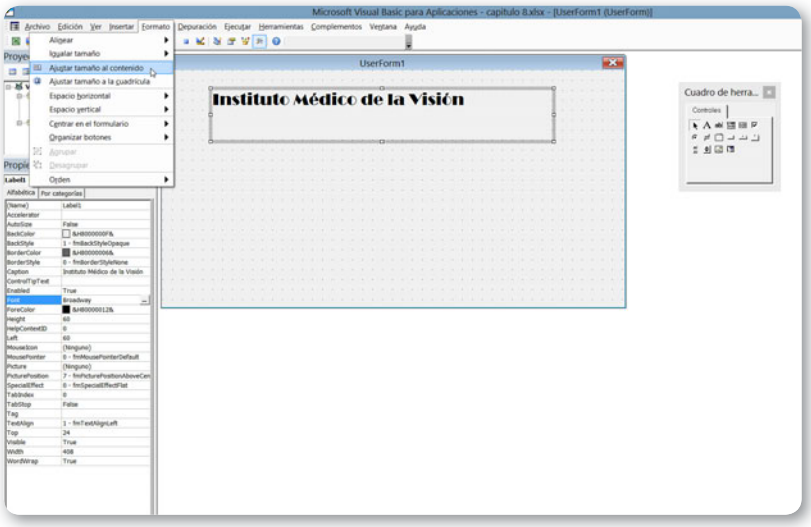


Figura 6. Podemos ajustar el tamaño de la **Etiqueta** desde el menú **Formato/Ajustar tamaño al contenido**.

- **TextAlign**: especifica la alineación del texto dentro del control. Las opciones disponibles son: **1 - fmTextAlignLeft** alinea el texto hacia la izquierda; **2 - fmTextAlignCenter** centra el texto y **3 - FmTextAlignRight** alinea el texto hacia la derecha. Por ejemplo, si queremos centrar el texto de la etiqueta **lbl_titulo** en tiempo de ejecución, utilizamos la siguiente sintaxis: **lbl_titulo.TextAlign = fmTextAlignCenter.**
- **AutoSize**: determina si el tamaño del control se ajusta de manera automática a su contenido. El valor que puede tomar es **True** o **False**.
- **Font**: establece la fuente, el estilo y el tamaño del texto del control.

Por ejemplo, si queremos que el texto de la etiqueta **lbl_titulo** tenga fuente **Arial**, estilo **Negrita**, tamaño **14** y color azul tenemos que usar la sentencia que presentamos a continuación:

```
With lbl_titulo
   .Font.Name = "Arial"
   .Font.Size = 14
   .Font.Bold = True
   .Font.Italic = True
   .ForeColor = RGB(0, 0, 255)
End With
```

- **BorderStyle**: al igual que en los formularios, esta propiedad establece el tipo de borde para la etiqueta.
- **BorderColor**: permite definir el color del borde de la etiqueta.

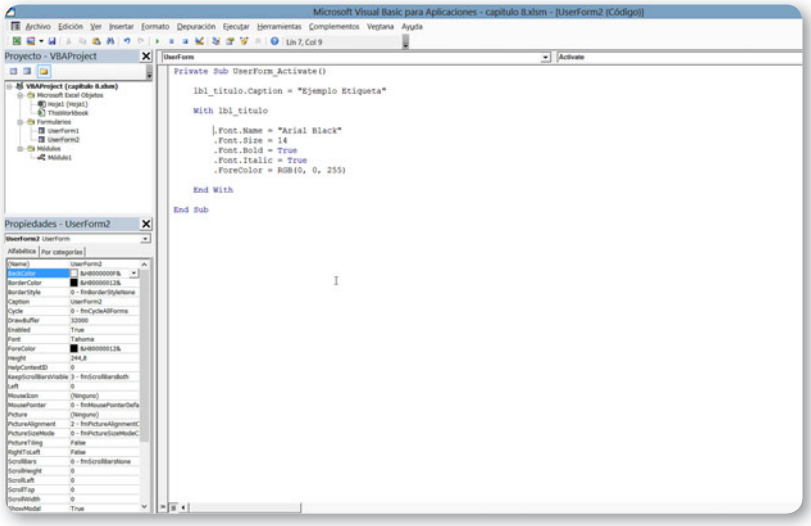


Figura 7. Por convención, como nombre de un **Label** se emplea el prefijo **lbl** seguido de un nombre significativo. Por ejemplo: **lbl_titulo**.

SELECCIONAR OBJETOS

Cuando agregamos controles a un formulario, en algún momento de nuestro trabajo vamos a tener que seleccionarlos, ya sea para cambiar su tamaño o bien para modificar su ubicación. Para esto, utilizaremos el **seleccionador de objetos**, que es el primer icono que aparece en el **Cuadro de herramientas**. Esta herramienta se encuentra seleccionada por defecto.

Eventos

Los eventos más empleados en las etiquetas son:

- **Click**: se produce cuando se pulsa el control.
- **DbClick**: ocurre cuando se hace doble clic sobre el control.

Cuadro de texto (TextBox)

El control **Cuadro de texto** (**TextBox**) se suele utilizar para mostrar información o para que el usuario ingrese datos. El contenido puede ser editado por el usuario.

Propiedades

Las propiedades más importantes de los cuadros de texto son:

- **Name**: permite establecer el nombre con el que haremos referencia al cuadro de texto en el código. De manera predeterminada, se le asigna el nombre **TextBox1**.
- **Text**: el contenido que muestra un cuadro de texto es controlado por esta propiedad. Se puede establecer de tres maneras diferentes. En tiempo de diseño, desde la ventana **Propiedades**. En tiempo de ejecución, mediante código, por ejemplo: **txt_mensaje.Text = "Ingrese sus datos aquí"**. Por último, a través del texto que escribe el usuario en tiempo de ejecución. El contenido se puede recuperar si se lee la propiedad **Text** en tiempo de ejecución.
- **Font**: esta propiedad establece la fuente, el estilo y el tamaño para el texto contenido en el control.
- **Enabled**: a partir de los valores **True** o **False**, determina si el control puede responder a eventos generados por el usuario. Esta propiedad

CAMBIAR LAS PROPIEDADES

Si seleccionamos varios controles, podremos cambiar de manera simultánea sus propiedades para acelerar nuestro trabajo. Pero debemos tener en cuenta que, cuando seleccionamos dos o más controles, en la ventana **Propiedades** únicamente veremos las propiedades compartidas por esos controles.

la utilizamos para habilitar y deshabilitar los controles. Cuando le asignamos el valor **False**, el usuario no podrá interactuar con el control. Es decir, el cuadro de texto quedará inhabilitado para responder al usuario hasta que cambiemos al valor **True**.

Por ejemplo, si queremos deshabilitar en tiempo de ejecución el cuadro de texto llamado **txt_Importe**, utilizamos la siguiente sentencia:

```
txt_Importe.Enabled = False.
```

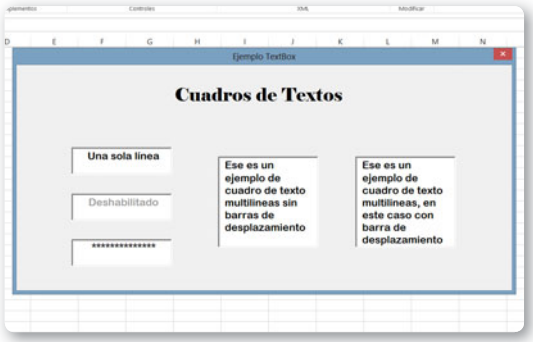


Figura 8. Por convención, como nombre de un **TextBox** se emplea el prefijo **txt** seguido de un nombre significativo. Por ejemplo: **txt_nombre**.

- **Locked**: especifica si el control puede ser editado. Puede adoptar dos valores: **True** o **False**. Si le asignamos el valor **True**, el usuario no podrá modificar el texto de este control.
- **MaxLength**: establece la longitud máxima de caracteres permitida para el cuadro de texto. Si no se establece o si se pone valor **0**, permite cualquier longitud de texto.

CONTROLSOURCE

En un cuadro de texto, es posible tomar el contenido de una celda determinada, utilizando la propiedad **ControlSource**. Esta propiedad es la que permite hacer referencia a una celda, por ejemplo, la **A1**. También podemos tomar el valor de una celda para usarlo en un cuadro de texto empleando el procedimiento de evento **TextBox_Change** del cuadro de texto.

- **MultiLine**: de manera predeterminada, el cuadro de texto presenta una única línea de texto y no muestra las barras de desplazamiento; por lo tanto, si un texto es más largo que el ancho de un cuadro de texto, solo veremos una parte de este. Si le asignamos el valor **True** a esta propiedad, el cuadro de texto aceptará múltiples líneas.
- **ScrollBars**: permite agregar barras de desplazamiento verticales (**fmScrollBarsVertical**), horizontales (**fmScrollBarsHorizontal**) o ambas (**fmScrollBarsBoth**). Por omisión, el cuadro de texto carece de barras de desplazamiento.
- **PasswordChar**: a veces, es conveniente que no se pueda leer lo que se escribe en el cuadro de texto, como, por ejemplo, la entrada de una contraseña. Esta propiedad permite especificar el carácter empleado para ocultar el texto que realmente contiene un control.

Figura 9. El contenido de la propiedad **Text** no cambia por el hecho de presentar en pantalla un carácter distinto.

Eventos

Los eventos más empleados en los cuadros de texto son:

- **Change**: ocurre cuando se modifica el contenido del área de edición del cuadro de texto.
- **KeyPress**: se produce cuando el usuario presiona una tecla.

Cuadro de lista (ListBox)

El control **Cuadro de lista** (**ListBox**) presenta una lista de elementos para que el usuario seleccione uno o varios de ellos. Si el número de

elementos supera a los que se pueden mostrar en el cuadro de lista, aparecen de manera automática las barras de desplazamiento en el control, que permiten recorrer todos los elementos.

Propiedades

Las propiedades más importantes del cuadro de lista son:

- **RowSource**: permite definir los valores de un **ListBox** o un **ComboBox** (este control lo veremos más adelante). El valor válido para esta propiedad es una cadena, que puede ser una dirección de celda o un nombre de rango. Por ejemplo, si queremos crear un cuadro de lista con los datos del rango **A3:A15** de la hoja de cálculo **Hoja1**, escribimos la siguiente sentencia en el evento **Activate** del formulario, de modo que los datos se carguen en forma automática al cuadro de lista al abrir el formulario.

```
Private Sub UserForm_Activate()
lst_bebidas.RowSource = "Hoja1!$A$3:$A$15"
End Sub
```

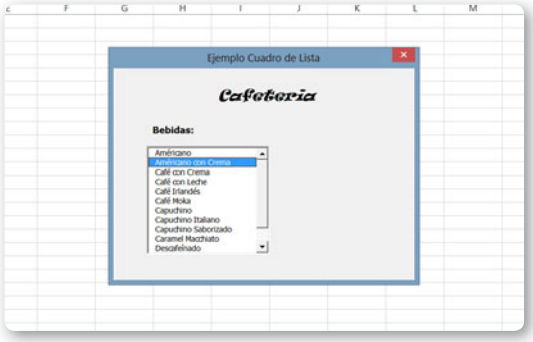


Figura 10. La propiedad **RowSource** también se puede establecer en tiempo de diseño, introduciendo el rango sin comillas en la ventana **Propiedades**. Por ejemplo: **=Hoja1!$A3:A15**.

- **ColumnCount**: permite establecer el número de columnas que se mostrarán en un **ListBox** o **ComboBox**. El valor por defecto es 1.

- **ColumnHeads**: permite mostrar u ocultar el encabezado de las columnas. Si establecemos el valor en **True**, se utilizará la primera fila de los datos como encabezados de las columnas.
- **ColumnWidths**: permite ajustar el ancho de cada columna de la lista. Si dejamos este valor en blanco (por defecto) o lo establecemos en **-1**, el ancho de la columna será determinado por Microsoft Excel. Es importante tener en cuenta dos restricciones para configurar de manera adecuada este parámetro. Los valores deben estar expresados en unidades (centímetros, pulgadas) y separados por punto y coma (**;**). Por ejemplo: 6 cm; 2 cm. Si no especificamos una unidad de medida, el ancho de la columna se medirá en pulgadas.

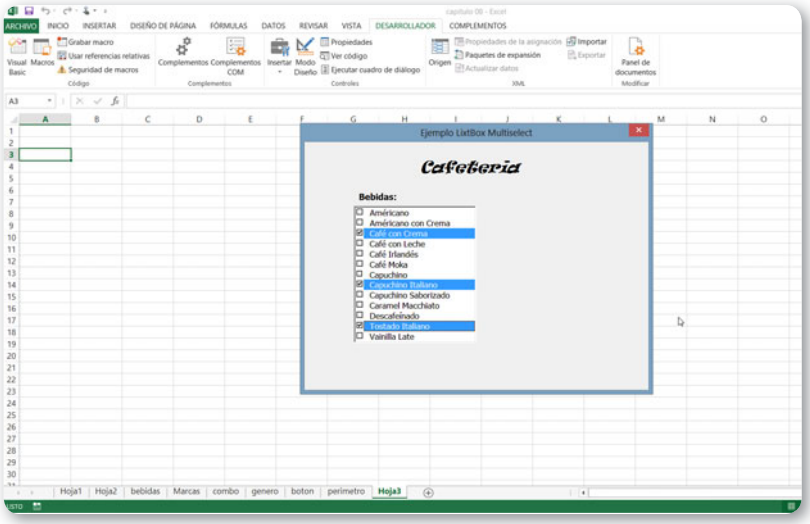


Figura 11. Por convención, como nombre de un **ListBox** se emplea el prefijo **lst** seguido de un nombre significativo. Por ejemplo: **lst_bebidas**.

LISTSTYLE

Esta propiedad es la que permite determinar la forma en que veremos la lista de elementos. La opción **fmListStylePlain** es el valor por defecto y muestra la lista sin ningún efecto visual. En cambio, **fmListStyleOption** agrega a la lista unos botones de opción, si se permite seleccionar un solo elemento; o unas casillas de verificación, si se permite una selección múltiple.

- **MultiSelect**: de forma predeterminada, podemos seleccionar un solo elemento de la lista, pero esta propiedad nos permite habilitar la selección de varios elementos. Tiene tres opciones: **0 - fmMultiSelectSingle** habilita la selección de un solo elemento de la lista; **1 - fmMultiSelectMulti** permite seleccionar o deseleccionar más de un elemento de la lista haciendo un clic con el mouse o presionando la barra espaciadora; **2 - fmMultiSelectExtended** habilita la selección de un rango continuo de la lista, combinando la tecla **SHIFT** con el botón izquierdo del mouse, y, para seleccionar elementos no contiguos, utilizamos la tecla **CTRL** y el botón izquierdo del mouse.
- **ListCount**: devuelve el número total de elementos que contiene un cuadro de lista.
- **ListIndex**: devuelve la posición del elemento seleccionado en el cuadro de lista. Si no se selecciona ningún elemento, el valor es **-1**. Por ejemplo, la siguiente sintaxis muestra un cuadro de diálogo con el número del elemento seleccionado:

```
MsgBox "Se seleccionó el elemento" & lst_bebidas.ListIndex & "de la lista"
```

- **Value**: devuelve el valor del elemento que ha sido seleccionado dentro de un **ListBox** o **ComboBox**.

Métodos

Los métodos más importantes del cuadro de lista son:

- **AddItem**: permite añadir a un cuadro de lista elementos que no están contenidos en un rango de una hoja de cálculo. También podemos

OTRAS PROPIEDADES

La propiedad **Selected** indica qué elementos han sido seleccionados de la lista. Se emplea en lugar de **ListIndex** cuando creamos un cuadro de lista con la propiedad **Multiselect** en el valor **fmMultiSelectMulti** o **fmMultiSelectExtended**. Los valores que puede tomar esta propiedad son **True** cuando el elemento ha sido seleccionado y **False** cuando el elemento no está seleccionado.

usarlo para recuperar elementos a partir de un rango de una hoja de cálculo o una matriz. Por ejemplo, si queremos cargar un cuadro de lista llamado **lst_aviones** con las marcas de ocho aviones, podemos escribir la siguiente sentencia:

```
With lst_aviones
    .RowSource = ""
    .AddItem "Airbus"
    .AddItem "Beechcraft"
    .AddItem "Boeing"
    .AddItem "Bombardier"
    .AddItem "Cesna"
    .AddItem "Embraer"
    .AddItem "Honda"
    .AddItem "Learjet"
End With
```

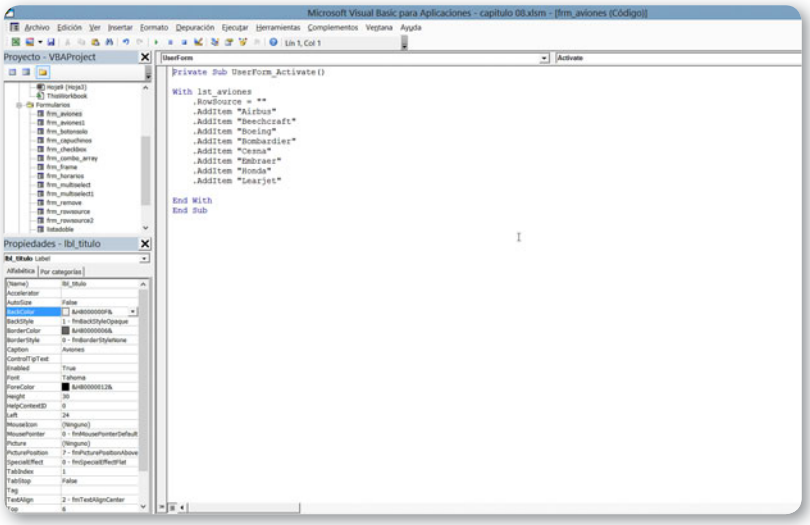


Figura 12. El método **`AddItem`** no funcionará si el **`ListBox`** está enlazado a datos, por lo tanto, **`RowSourse`** se borrará si se hubiera establecido.

Si los datos que queremos que contenga el cuadro de lista se encuentran en un rango de celdas de una hoja de cálculo, podemos

usar una estructura de bucle que recorra el rango de celdas y vaya ingresando cada elemento al cuadro de lista. Por ejemplo, si tenemos una hoja de cálculo llamada **Marcas**, que contiene en el rango **A3:A10** los elementos que queremos que aparezcan en el cuadro de lista llamado **lst_aviones**, podemos utilizar la siguiente sentencia para cargar los datos al cuadro de lista.

```
Worksheets("Marcas").Range("A3").Select
Do While ActiveCell <> Empty
   lst_aviones.AddItem ActiveCell
   ActiveCell.Offset(1, 0).Select
Loop
```

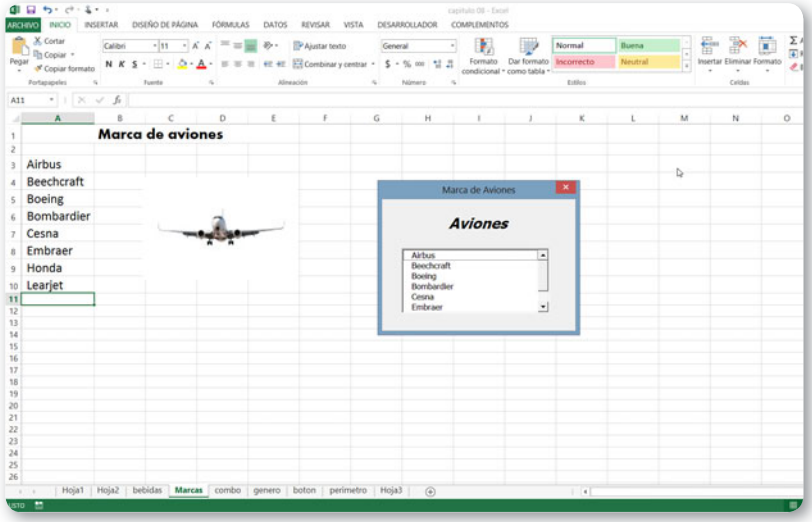


Figura 13. Los nombres que están debajo de **A2** serán enviados al **`ListBox`**. Cuando aparezca una celda vacía, la condición **`Do While`** finalizará la ejecución del código.

- **RemoveItem**: permite eliminar elementos de un cuadro de lista. Por ejemplo, si tenemos un cuadro de lista llamado **lst_aviones** con ocho elementos y queremos eliminar el segundo, escribimos la sentencia:

  lst_aviones.RemoveItem (1).

Si en cambio queremos eliminar el elemento que hemos seleccionado de la lista, usamos la sentencia:

```
lst_aviones.RemoveItem (lst_aviones.ListIndex).
```

- **Clear**: permite borrar todos los elementos de un cuadro de lista. Por ejemplo, si queremos eliminar el contenido del cuadro de lista llamado **lst_aviones**, escribimos la sentencia:

```
Lst_aviones.Clear.
```

Eventos

Los eventos más empleados en un cuadro de lista son:
-**Click**: ocurre cuando el usuario interactúa con el control.
-**Change**: se produce cuando se cambia un elemento de la lista.

Cuadro combinado (ComboBox)

El control **Cuadro combinado** (**ComboBox**) es un cuadro de lista desplegable. Cuando está cerrado, muestra un solo elemento, pero, si lo desplegamos mediante la flecha que se encuentra a su derecha, presenta la lista completa de elementos.

Un **ComboBox** reúne las características de un cuadro de texto (**TextBox**) y un cuadro de lista (**ListBox**), porque le permite al usuario elegir una opción de la lista o agregar elementos a la lista.

Propiedades

El cuadro combinado presenta muchas propiedades que son comunes al control cuadro de lista, como, por ejemplo, **RowSource**, **ColumnCount**, **ColumnHeads**, **ColumnWidths**, **ListCount** y **ListIndex**. Una propiedad específica de este control es **Style**, que establece el comportamiento del control y tiene dos opciones:

- **0 - fmStyleDropDownCombo**: es el estilo por defecto; despliega la lista cuando hacemos clic en la flecha. En este caso, el usuario puede seleccionar o escribir un valor en el área de edición.

- **2 - FmStyleDropDownList**: es similar a la anterior, pero no permite ingresar texto, solo seleccionar elementos.

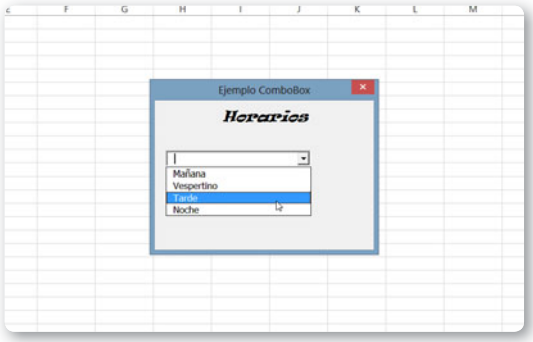


Figura 14. Por convención, como nombre de un **ComboBox** se emplea el prefijo **cbo** seguido de un nombre significativo. Por ejemplo: **cbo_horarios**.

Métodos

Al igual que el control **ListBox**, que vimos anteriormente, el control **ComboBox** posee el método **Additem** para agregar elementos a la lista, el método **RemoveItem** para eliminar un elemento determinado y el método **Clear** para eliminar todos los elementos de la lista.

Por ejemplo, si queremos crear un cuadro combinado llamado **cbo_turnos**, podemos utilizar la siguiente sentencia:

```
cbo_turnos.AddItem "Mañana"
cbo_turnos.AddItem "Vespertino"
cbo_turnos.AddItem "Tarde"
cbo_turnos.AddItem "Noche"
```

REMOVEITEM

El método **RemoveItem** solo se puede utilizar con una macro o código VBA. Debemos tener en cuenta que este método no funcionará si el **ListBox** o el **ComboBox** están enlazados a datos a través de la propiedad **RowSource**. Por tal motivo, esta propiedad debe ser limpiada antes de emplear el método **RemoveItem**.

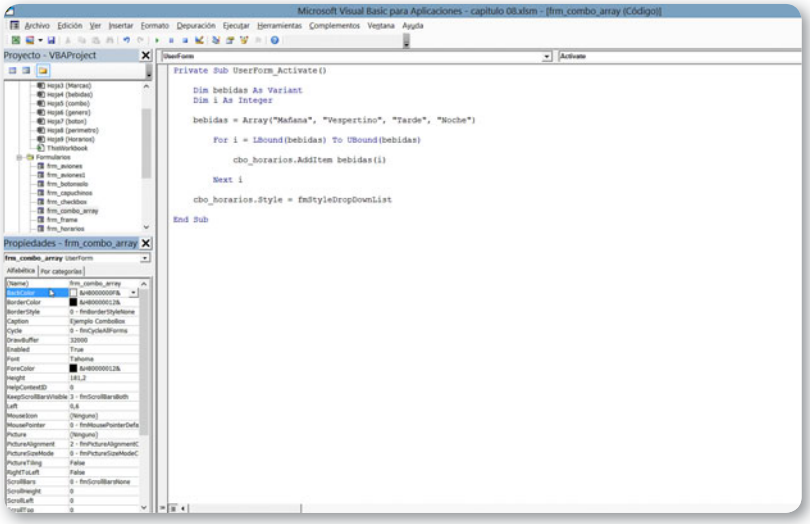


Figura 15. También podemos utilizar un **Array** para rellenar un **ComboBox** con elementos que no están enlazados a una hoja de cálculo.

Eventos

Los eventos más empleados en un cuadro combinado son:

- **Click**: ocurre cuando el usuario interactúa con el control.
- **Change**: se produce cuando se modifica un elemento de la lista.

Botón de comando (CommandButton)

La mayoría de los formularios tiene al menos un control **Botón de comando** (**CommandButton**), que le permite al usuario hacer clic sobre él para realizar acciones, como, por ejemplo, ejecutar una función.

ESPACIO ENTRE CONTROLES

Es posible configurar el espacio que queremos que haya entre los controles de un formulario. Para esto, seleccionamos los controles; luego, vamos al menú **Formato** y seleccionamos **Espacio horizontal** (o **Espacio vertical**). En función del espacio que deseamos modificar, pulsamos **Igualar, Aumentar, Disminuir o Quitar**.

Por lo general, se utiliza el evento **Click** para ejecutar alguna acción cuando se produce dicho evento.

Propiedades

Las propiedades más importantes del botón de comando son:

- **Name**: permite asignarle un nombre al botón de comando.
- **Caption**: permite ingresar el texto que muestra el botón de comando.
- **Font**: permite elegir la fuente, el estilo y el tamaño para el texto contenido en el botón de comando.
- **Height**: determina el alto del botón.
- **Width**: determina el ancho del botón.
- **Left**: permite establecer la distancia desde el extremo izquierdo del botón hasta el extremo izquierdo del formulario que lo contiene.
- **Top**: permite configurar la distancia desde el extremo superior del botón hasta el extremo superior del formulario que lo contiene.

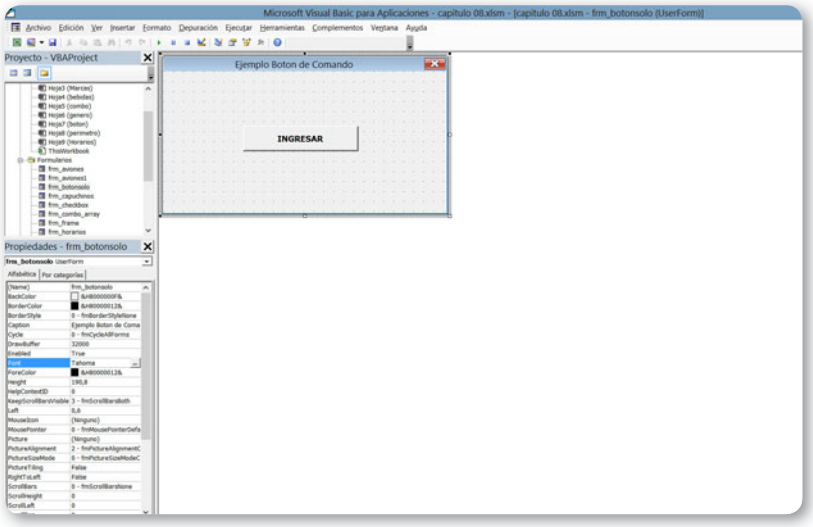


Figura 16. Por convención, como nombre de un **CommandButton** se emplea el prefijo **cmd** seguido de un nombre significativo. Por ejemplo **cmd_calcular**.

- **BackStyle**: permite establecer el estilo de fondo para el botón de comando. Tiene dos opciones: **fmBackStyleTransparent** hace

transparente el fondo del botón, es decir, se ve lo que hay detrás de este objeto; **fmBackStyleOpaque** establece el fondo del botón en opaco.

- **BackColor**: permite especificar el color de segundo plano del botón de comando. Para poder visualizar el color del botón, la propiedad **BackStyle** debe ser **fmBackStyleOpaque**.
- **Enabled**: permite habilitar o deshabilitar el botón. Si tiene el valor **True**, el botón estará habilitado y responderá a las acciones generadas por el usuario. Si el valor es **False**, el botón quedará inhabilitado para el usuario.

Métodos

Este control prácticamente no posee métodos. El más importante es **SetFocus**, que establece el foco en un control VBA, en este caso, el botón de comando. Para usar este método, el control debe estar habilitado.

Eventos

El evento más empleado en el botón de comando es **Click**, que ocurre cuando el usuario presiona el botón de comando con el botón izquierdo del mouse o cuando pulsa la tecla **ENTER**.

Marco (Frame)

El control **Marco** (**Frame**) se utiliza para agrupar otros controles y darle una mayor funcionalidad a la interfaz. Es especialmente útil cuando el cuadro de diálogo contiene más de un grupo de controles de opción. También podemos usarlo para subdividir un formulario y organizar su estructura.

CÓDIGO EN MINÚSCULAS

Es una buena ayuda escribir el código que usamos en minúscula; de esta manera, sabremos de forma rápida si la sintaxis y la ortografía del código es correcta ya que, al presionar la tecla **ENTER**, VBA pone automáticamente en mayúscula todas las palabras reconocidas. También, podemos recurrir a la ayuda de las palabras clave, ubicando el cursor sobre el término y presionando la tecla **F1**.

Para ubicar diferentes controles dentro de un marco, primero debemos dibujar el marco en el formulario y, luego, arrastrar los controles desde el **Cuadro de herramientas** al interior del marco.

Propiedades

Las propiedades principales del marco son:

- **Caption**: es la propiedad por defecto de este control, permite agregar un texto, a modo de título, en la parte superior del marco.
- **BorderStyle**: establece si el marco tendrá o no una línea de borde. Tiene dos opciones. **fmBorderStyleNone** para un marco sin borde. Debemos tener en cuenta que, si le quitamos el borde al control, el texto que le asignemos en **Caption** no se visualizará. **fmBorderStyleSingle** mostrará un borde de una sola línea.
- **Visible**: permite ocultar el control. Si **Frame** contiene otros controles, y establecemos esta propiedad en **False**, estos quedarán ocultos.

Otras propiedades que presenta este control y que ya conocemos son **Font**, **BackColor**, **Enabled**, **Left**, **Top**.

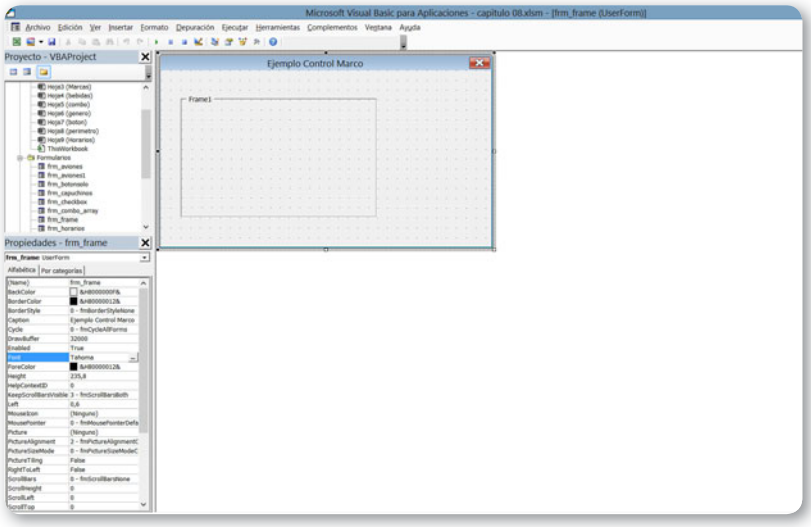


Figura 17. Por convención, como nombre de un **Frame** se emplea el prefijo **fra** seguido de un nombre significativo. Por ejemplo: **fra_opciones**.

Métodos y eventos

Este control también tiene métodos y eventos, pero, como su uso es infrecuente, no los veremos en este libro.

Casilla de verificación (CheckBox)

El control **Casilla de verificación** (**CheckBox**) le ofrece al usuario una opción para que dé una respuesta del tipo **Sí/No** o **Verdadero/Falso**. También se emplea, por lo general, para listar opciones y que el usuario pueda elegir entre ellas; es muy útil cuando el usuario tiene que elegir entre más de una opción. Dependiendo de si la casilla está seleccionada (**True**) o no (**False**), se realiza una acción u otra.

Propiedades

La propiedad **Value** devuelve y establece el valor seleccionado. Tiene tres posibles valores: **Null** indica que la casilla de verificación no está seleccionada, ni desactivada; **True** indica que la casilla está marcada, y **False**, que se encuentra desactivado.

El control tiene otras propiedades que ya conocemos, como **Alignmen**, **AutoSize**, **Font** y **ForeColor**.

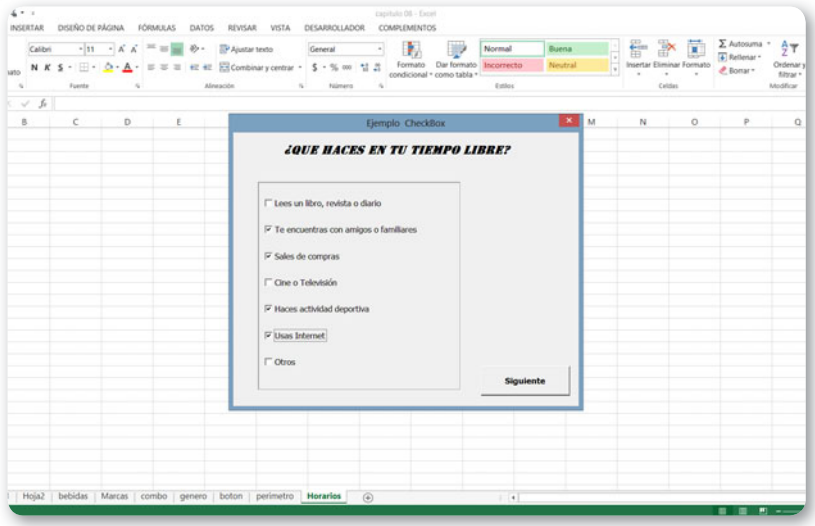


Figura 18. Por convención, como nombre de un **CheckBox** se emplea el prefijo **chk** seguido de un nombre significativo. Por ejemplo: **chk_auto**.

Métodos

El método más importante de la casilla de verificación es **SetFocus**, que establece el enfoque a un objeto específico. Cuando un control tiene el enfoque (o foco) se convierte en activo y puede responder directamente a una entrada del usuario con el teclado o con el mouse.

Eventos

El evento más empleado es **Click**, que ocurre cuando el usuario selecciona de manera definitiva un valor entre otros posibles.

Botón de opción (OptionButton)

El control **Botón de opción** (**OptionButton**) se emplea cuando el usuario tiene que seleccionar solo una opción entre varias disponibles. Por lo general, se usa en grupos de al menos dos elementos, y es conveniente agrupar los botones de opción en un control **Frame**.

Debemos tener en cuenta que todos los controles de este tipo que están dentro de un mismo marco actúan como un solo grupo, independiente de los botones de opción que se encuentran en otros grupos.

> EL CONTROL BOTÓN DE OPCIÓN PERMITE SELECCIONAR UN ELEMENTO ENTRE VARIOS DISPONIBLES

Propiedades

Las principales propiedades del botón de opción son:

- **Name**: permite establecer el nombre con el que haremos referencia al botón de opción en el código.
- **Caption**: muestra la descripción que acompaña al botón de opción.
- **Value**: los valores que puede tomar esta propiedad son de tipo boolean. Si el botón está seleccionado, toma el valor **True**; en caso contrario, toma el valor **False**.

El resto de las propiedades que posee son comunes a la mayoría de los otros controles, como **Font**, **Enabled**, **BackColor**, **Visible**.

Métodos

El método más importante que posee es **SetFocus**, que establece el enfoque a un objeto específico, en este caso, el botón de opción.

Eventos

El evento más empleado en el botón de opción es **Click**, que ocurre cuando el usuario selecciona de forma definitiva un valor entre otros.

Imagen (Image)

El control **Imagen** (**Image**) se utiliza para mostrar una foto, gráfico o ilustración dentro de un formulario, que puede provenir de un archivo o del **Portapapeles**. La imagen se guarda en el libro de trabajo; de esta manera, es posible distribuir el libro de Excel a cualquier persona sin incluir una copia del archivo de imagen. Las imágenes pueden ser del tipo **BMP**, **CUR**, **GIF**, **ICO**, **JPG**, **WNF**.

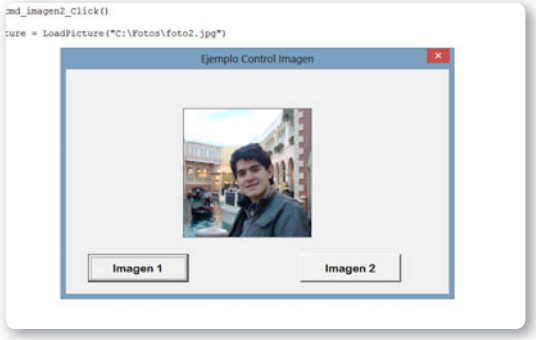


Figura 19. Por convención, como nombre de un **Image** se emplea el prefijo **img** seguido de un nombre significativo. Por ejemplo: **img_foto**.

BOTÓN DE ALTERNAR

El control **ToggleButton** (botón de alternar) posee dos estados: **activado** y **desactivado**. Al hacer clic sobre este botón, se intercambian estos dos estados, y el botón modifica su apariencia. Si está presionado, toma el valor verdadero (**True**), y, en el caso contrario, toma el valor falso (**False**).

Propiedades

Las principales propiedades del control **Imagen** son:

- **AutoSize**: establece si la imagen cambia o no de tamaño automáticamente. Tiene dos valores: **False** o **True.**
- **Picture**: se utiliza para indicar el archivo de imagen que se mostrará en el control. Cuando se especifica desde el código VBA, hay que emplear la función **LoadPicture**.
- **PictureSizeMode**: especifica cómo se mostrará la imagen, en relación a su tamaño original y a la escala. Tiene tres opciones. **fmPictureSizeModeClip** es el valor predeterminado, recorta la imagen que sea más grande que el formulario. **fmPictureSizeModeStrech** agranda la imagen para adaptarla al formulario, puede distorsionarla horizontal o verticalmente. Con esta opción, no se tendrá en cuenta la propiedad **PictureAlignment**. **fmPictureSizeModeZoom** agranda la imagen sin distorsionarla.
- **PictureAlignment**: especifica la alineación de la imagen. Tiene cinco opciones: **fmPictureAlignmenTopLeft** alinea la imagen a la esquina superior izquierda; **fmPictureAlignmenTopRight** alinea la imagen a la esquina superior derecha; **fmPictureAlignmenCenter** centra la imagen en el formulario; **fmPictureAlignmenBottomLeft** alinea la imagen a la esquina inferior izquierda; **fmPictureAlignmenBottomRight** alinea la imagen a la esquina inferior derecha.

> EL CONTROL IMAGE SE UTILIZA PARA MOSTRAR UNA FOTO DENTRO DE UN FORMULARIO

Este control también tiene otras propiedades que ya conocemos, como **BorderStyle**, **BorderColor**, **Height**, **Width**.

Eventos

Los principales eventos del control **Image** son:

- **Click**: cuando el usuario hace clic sobre la imagen.
- **DoubleClick**: cuando el usuario hace doble clic sobre la imagen.
- **MouseMove**: cuando el mouse se mueve sobre el área del control.

- **MouseDown**: cuando el mouse está presionado, antes de **Click**.
- **MouseUp**: se produce al soltar el botón del mouse.

Página múltiple (MultiPage)

El control **Página múltiple** (**MultiPage**) contiene diferentes fichas. Cada ficha es una nueva página que puede contener un conjunto de controles diferentes. La selección de una página oculta a las otras páginas de la hoja. Este control es útil cuando queremos manejar una gran cantidad de datos que se pueden clasificar en varias categorías, porque crea una página para cada una.

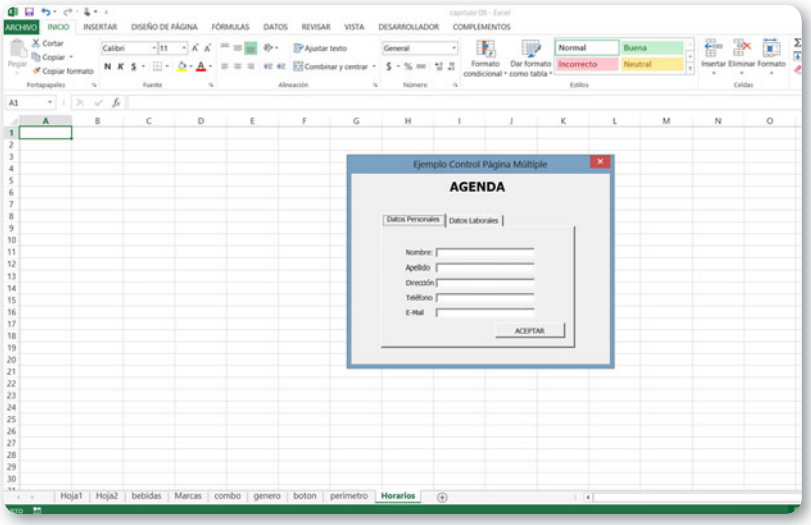


Figura 20. Por convención, como nombre de un **`MultiPage`** se emplea el prefijo **`mpg`** seguido de un nombre significativo. Por ejemplo: **`mpg_datos`**.

REFEDIT

El control **RefEdit** se utiliza para permitirle al usuario seleccionar un rango de una hoja de cálculo. Este rango lo podemos aplicar dentro de un formulario o de una instrucción VBA. El control muestra la dirección de la celda o rango que el usuario selecciona o introduce. El rango de celdas puede ser una celda o un rango de celdas contiguas o no.

De manera predeterminada, el control tiene dos páginas, y cada página tiene su propio orden de tabulación (la activación de cada control de una página al presionar la tecla **TAB**). La numeración del orden de tabulación comienza por **0**.

Propiedades

La principal propiedad del control **MultiPage** es **Caption**, que establece el nombre que aparece en la ficha del control.

Eventos

El evento predeterminado de este control es el evento **Change**, que se produce cada vez que se cambia de página.

Barra de desplazamiento (ScrollBar)

El control **Barra de desplazamiento** (**ScrollBar**) permite rápidos desplazamientos a lo largo de una lista de elementos. Recorre un determinado rango de valores cuando un usuario hace clic en las flechas de desplazamiento, cuando arrastra el **ScrollBox** o hace clic en algún área entre las flechas de desplazamiento y el **Scrollbox**. Es similar a las barras de desplazamiento de cualquier programa.

> LA BARRA DE DESPLAZAMIENTO PERMITE RECORRER UNA LISTA DE ELEMENTOS

Propiedades

Las principales propiedades que posee la barra de desplazamiento son las que detallamos a continuación:

TAMAÑO DE LOS CONTROLES

Para unificar el tamaño de varios controles, debemos seleccionar los controles, haciendo clic sobre cada uno de ellos mientras mantenemos presionada la tecla **SHIFT**, luego vamos al menú **Formato/Igualar tamaño** y, en función de lo que deseamos, elegimos una de las siguientes opciones: **Ancho, Alto o Ambos**.

- **Value**: es el valor actual del cuadro de desplazamiento, consiste en un número entero que se encuentra comprendido entre los valores asignados a las propiedades **Max** y **Min**.
- **Max**: establece el máximo valor de desplazamiento cuando el botón se encuentra en la posición más alta.
- **Min**: establece el mínimo valor de desplazamiento cuando el botón se encuentra en la posición más baja.
- **SmallChange**: determina la cantidad en que la propiedad **Value** cambia cuando se hace clic sobre el control, va de 1 a 32767.
- **LargeChange**: especifica el cambio incremental cuando un usuario hace clic en el desplazamiento largo. El valor por defecto es **1**.
- **Orientation**: determina si la barra de desplazamiento se ubica en sentido horizontal o vertical. Tiene tres configuraciones.
  -1 - fmOrientationAuto: el alto y el ancho del control definen si la barra se muestra en forma horizontal o vertical. **0 - fmOrientationVertical**: el control se mostrará en forma vertical. **1 - fmOrientationHorizontal**: el control se mostrará en forma horizontal.

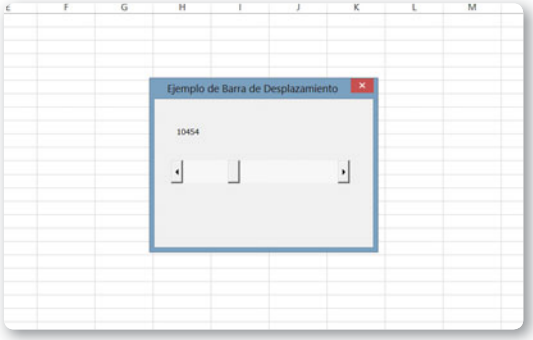


Figura 21. Por convención, como nombre de un **ScrollBar** vertical se emplea el prefijo **vsb** y para el horizontal, **hsb**.

FORMULARIOS NO MODALES

Un formulario es no modal cuando el usuario puede hacer clic en cualquier otra parte de la aplicación mientras el formulario se encuentra abierto. Estos formularios solo son compatibles con Microsoft Office 2000 y las versiones posteriores. Si tratamos de generar un formulario no modal en una versión anterior de Office, se producirá un error en tiempo de ejecución.

Métodos

El método más importante es **SetFocus**, que establece el enfoque a un objeto específico, en este caso el **ScrollBar**.

Eventos

El evento más utilizado para este control es el evento **Change**, que ocurre cada vez que el usuario genera un desplazamiento.

Botón de número (SpinButton)

El control **Botón de número** (**SpinButton**) está conformado por una flecha hacia arriba y otra hacia abajo, y permite al usuario seleccionar un valor haciendo clic sobre una de las dos flechas. Se usa frecuentemente junto con los controles **Cuadro de texto** o **Etiqueta**, que muestran el valor actual (**Value**) del botón de número.

> EL CONTROL BOTÓN DE NÚMERO LE PERMITE AL USUARIO SELECCIONAR UN VALOR

Propiedades

Las propiedades más importantes del botón de número son las siguientes:

- **Value**: determina el valor del **SpinButton**.
- **Max**: establece el valor máximo que puede tener el control.
- **Min**: determina el valor mínimo que puede tener el control.
- **SmallChange**: indica la cantidad en que la propiedad **Value** cambia cuando se hace clic sobre el control. El valor predeterminado es **1**.

MULTIPAGE CONTRA TABSTRIP

Un control **MultiPage** es un contenedor de controles similar a un **Frame**. Cada página tiene un conjunto diferente de controles, y la selección de una página oculta las otras. En cambio, el control **TabStrip** posee un conjunto de controles del mismo tipo en todas las fichas. El contenido de los controles cambia cuando seleccionamos una ficha, pero el diseño de los controles sigue siendo igual.

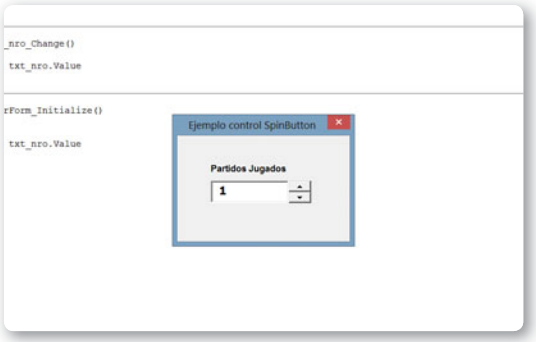


Figura 22. `Min`, `Max` y `SmallChange` toman valores enteros largos, lo que significa que no hay decimales

Métodos

El método más importante es **SetFocus**, que establece el enfoque a un objeto específico, en este caso, el botón de número.

Eventos

El evento más utilizado para este control es **Change**, que se produce cada vez que presionamos una de las flechas del control.

Usar un formulario en una hoja de cálculo

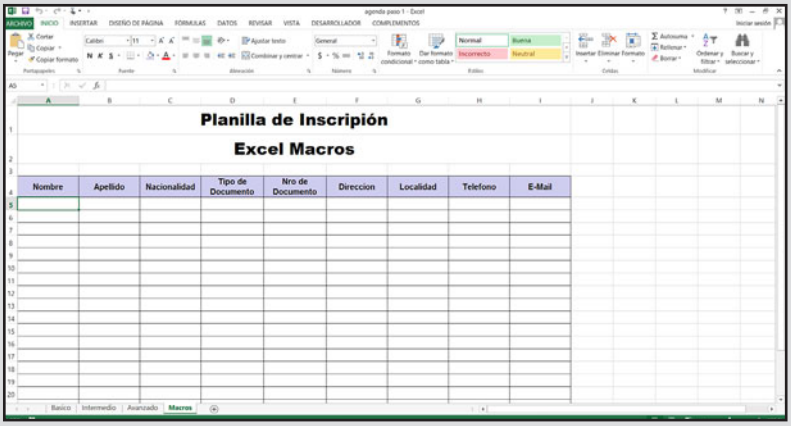
Luego de haber aprendido a crear formularios y de haber conocido cada uno de sus controles, veremos un ejemplo donde crearemos un formulario que nos permita ingresar datos en una hoja de cálculo. En este caso, haremos un registro de alumnos de distintos cursos.

BARRA DE TABULACIONES

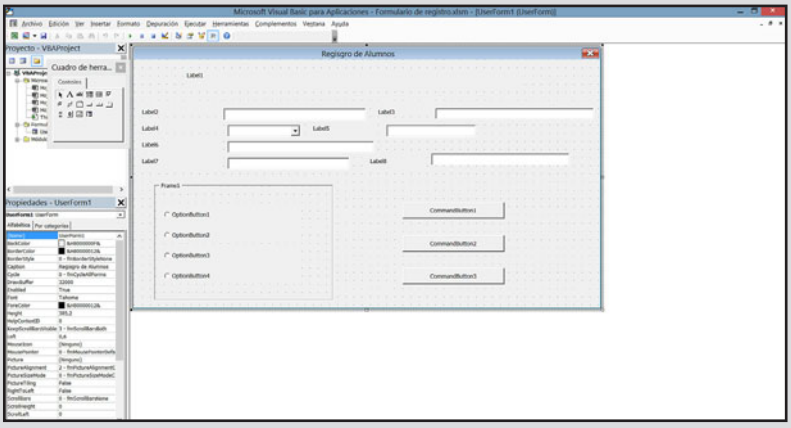
Un control **TabStrip** (barra de tabulaciones), de forma predeterminada, tiene dos fichas, llamadas **Tab1** y **Tab2**, respectivamente. Cada ficha es un objeto independiente. Entonces, este control se emplea para visualizar distintos contenidos en cada ficha, para el mismo conjunto de controles.

PXP: CREAR UN FORMULARIO DE REGISTRO

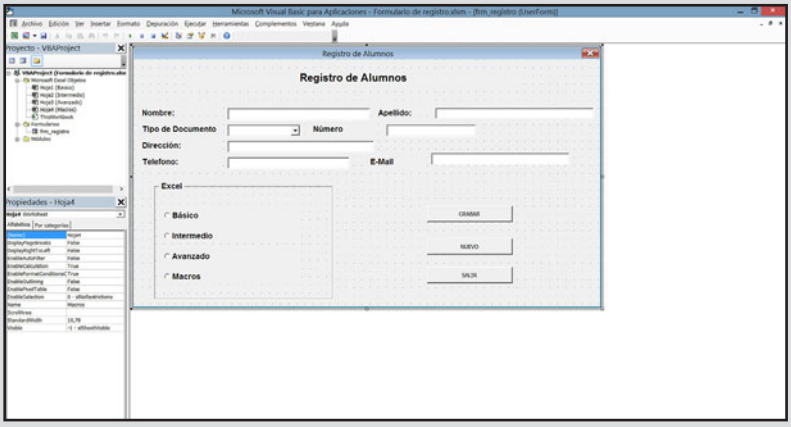
01 Diseñe una planilla como muestra la imagen. Copie la hoja tres veces y cambie el nombre de cada una por `Básico`, `Intermedio`, `Avanzado` y `Macros`.



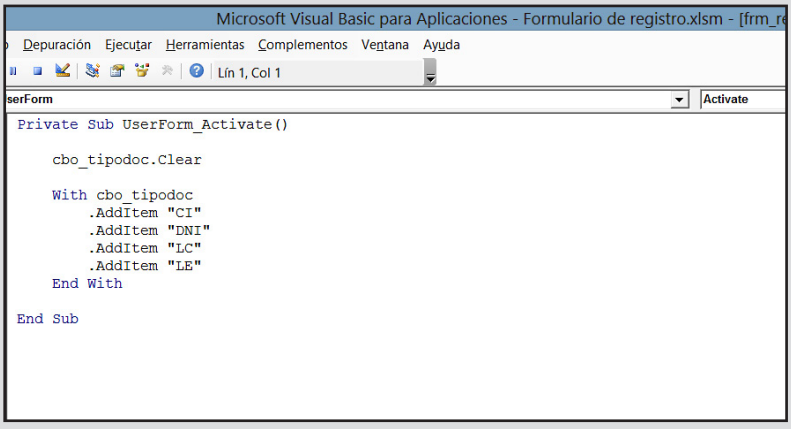
02 Abra el editor de VBA e inserte un formulario. En `Caption`, ingrese `Registro de Alumnos`. Agregue ocho etiquetas, seis cuadros de texto, un cuadro combinado, un marco, cuatro botones de opción y dos botones de comando.



03 Defina las propiedades Name y `Caption` para cada uno de los controles. Por ejemplo, para los botones de comando, Name: `cmd_grabar`; `cmd_nuevo`; `cmd_salir`, `Caption`: Grabar, Nuevo y `Salir`.



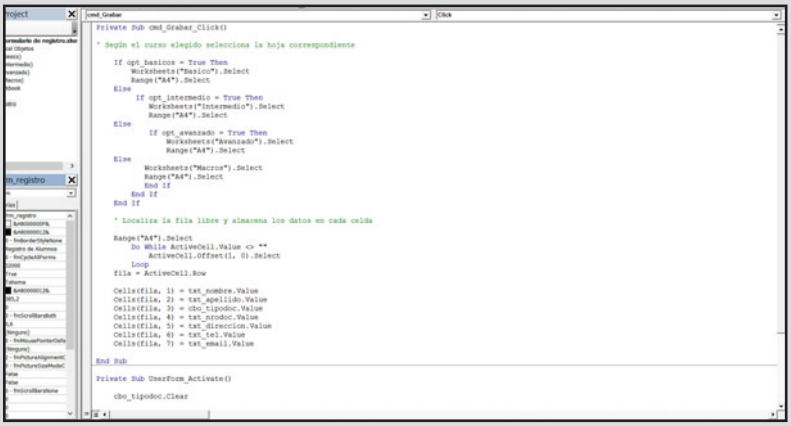
04 Para cargar el cuadro de lista (ComboBox) cuando el formulario se carga por primera vez, haga doble clic sobre el formulario y, en el evento `Activate`, escriba el código que se muestra en la imagen.



05 Para guardar los datos ingresados desde el formulario, haga doble clic sobre el botón `Grabar` y, en el evento `Click`, escriba el código que se muestra en la imagen, que permite según el curso elegido seleccionar la hoja correspondiente.

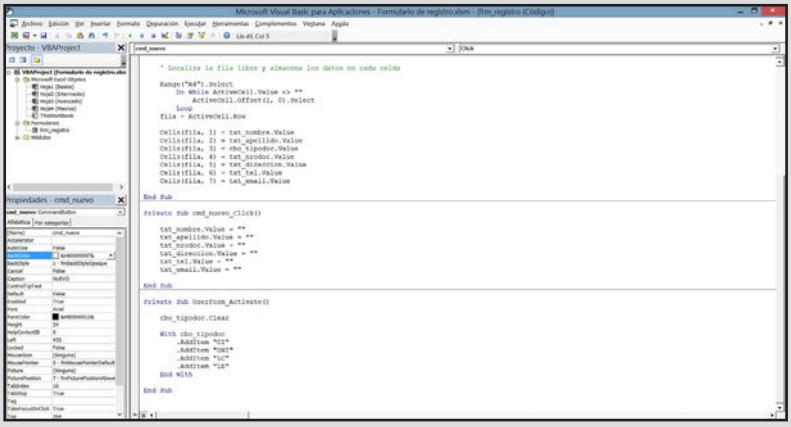
```
Private Sub cmd_Grabar_Click()

' Según el curso elegido selecciona la hoja correspondiente

    If opt_basicos = True Then
        Worksheets("Basico").Select
        Range("A4").Select
    Else
        If opt_intermedio = True Then
            Worksheets("Intermedio").Select
            Range("A4").Select
    Else
            If opt_avanzado = True Then
                Worksheets("Avanzado").Select
                Range("A4").Select
    Else
            Worksheets("Macros").Select
            Range("A4").Select
            End If
        End If
    End If

End Sub

Private Sub UserForm_Activate()

    cbo_tipodoc.Clear

    With cbo_tipodoc
        .AddItem "CI"
        .AddItem "DNI"
        .AddItem "LC"
        .AddItem "LE"
    End With

End Sub
```

06 A continuación, escriba el código que se muestra en la imagen. Este se encarga de localizar la fila libre y almacena, en cada una de las celdas correspondientes, los datos que provienen del formulario.



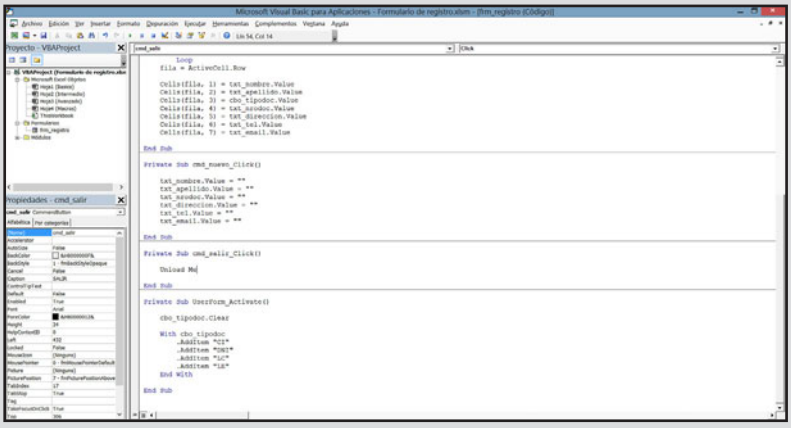
```
Private Sub cmd_Grabar_Click()

' Según el curso elegido selecciona la hoja correspondiente

    If opt_basicos = True Then
        Worksheets("Basico").Select
        Range("A4").Select
    Else
        If opt_intermedio = True Then
            Worksheets("Intermedio").Select
            Range("A4").Select
    Else
            If opt_avanzado = True Then
                Worksheets("Avanzado").Select
                Range("A4").Select
    Else
            Worksheets("Macros").Select
            Range("A4").Select
            End If
        End If
    End If

    ' Localiza la fila libre y almacena los datos en cada celda

    Range("A4").Select
        Do While ActiveCell.Value <> ""
            ActiveCell.Offset(1, 0).Select
        Loop
    fila = ActiveCell.Row

    Cells(fila, 1) = txt_nombre.Value
    Cells(fila, 2) = txt_apellido.Value
    Cells(fila, 3) = cbo_tipodoc.Value
    Cells(fila, 4) = txt_nrodoc.Value
    Cells(fila, 5) = txt_direccion.Value
    Cells(fila, 6) = txt_tel.Value
    Cells(fila, 7) = txt_email.Value

End Sub

Private Sub UserForm_Activate()

    cbo_tipodoc.Clear
```

07 Programe el botón Nuevo para ingresar nuevos registros. Este procedimiento blanquea el formulario para un nuevo ingreso. En el evento `Click` del botón, escriba el siguiente código.



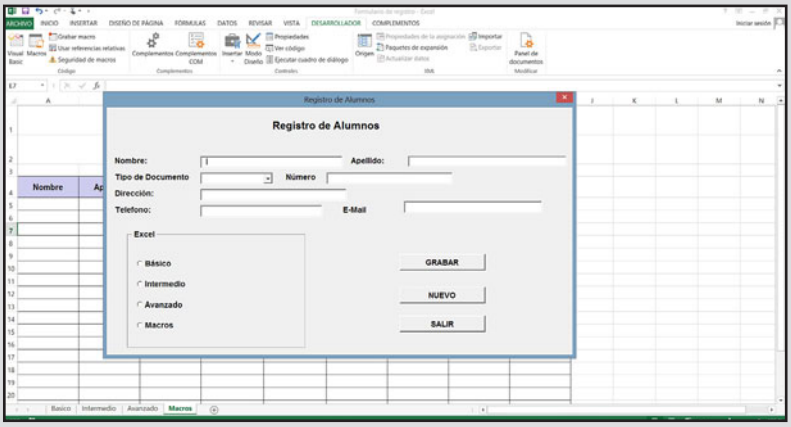
```
' Localiza la fila libre y almacena los datos en cada celda

Range("A4").Select
    Do While ActiveCell.Value <> ""
        ActiveCell.Offset(1, 0).Select
    Loop
fila = ActiveCell.Row

Cells(fila, 1) = txt_nombre.Value
Cells(fila, 2) = txt_apellido.Value
Cells(fila, 3) = cbo_tipodoc.Value
Cells(fila, 4) = txt_nrodoc.Value
Cells(fila, 5) = txt_direccion.Value
Cells(fila, 6) = txt_tel.Value
Cells(fila, 7) = txt_email.Value

End Sub

Private Sub cmd_nuevo_Click()

    txt_nombre.Value = ""
    txt_apellido.Value = ""
    txt_nrodoc.Value = ""
    txt_direccion.Value = ""
    txt_tel.Value = ""
    txt_email.Value = ""

End Sub

Private Sub UserForm_Activate()

    cbo_tipodoc.Clear

    With cbo_tipodoc
        .AddItem "CI"
        .AddItem "DNI"
        .AddItem "LC"
        .AddItem "LE"
    End With

End Sub
```

08 Finalmente, solo resta que programe el botón `Salir`. Para realizar este procedimiento, solo tendrá que ecribir en el evento `Click` del botón, el código `Unload Me`, como puede obeservarse en la imagen.



```
    Loop
fila = ActiveCell.Row

Cells(fila, 1) = txt_nombre.Value
Cells(fila, 2) = txt_apellido.Value
Cells(fila, 3) = cbo_tipodoc.Value
Cells(fila, 4) = txt_nrodoc.Value
Cells(fila, 5) = txt_direccion.Value
Cells(fila, 6) = txt_tel.Value
Cells(fila, 7) = txt_email.Value

End Sub

Private Sub cmd_nuevo_Click()

    txt_nombre.Value = ""
    txt_apellido.Value = ""
    txt_nrodoc.Value = ""
    txt_direccion.Value = ""
    txt_tel.Value = ""
    txt_email.Value = ""

End Sub

Private Sub cmd_salir_Click()

    Unload Me

End Sub

Private Sub UserForm_Activate()

    cbo_tipodoc.Clear

    With cbo_tipodoc
        .AddItem "CI"
        .AddItem "DNI"
        .AddItem "LC"
        .AddItem "LE"
    End With

End Sub
```

09 Ejecute el formulario presionando la tecla F5, y a continuación ingrese algunos datos para probar que el código funciona de manera adecuada.



RESUMEN

En este capítulo, describimos la utilidad del formulario y sus principales propiedades y métodos. También explicamos los distintos controles que podemos emplear en un UserForm para crear una interfaz de usuario amigable. Conocimos sus propiedades, métodos y eventos más importantes. A través de ejemplos, vimos cómo podemos programar dichos controles mediante el lenguaje VBA. Por último, aprendimos a almacenar los datos provenientes de un formulario en varias hojas de cálculo.

Actividades

TEST DE AUTOEVALUACIÓN

1. ¿Qué es un formulario?
2. Explicar el método **Show**.
3. ¿Para qué se emplea la propiedad **Caption** del control **TextBox**?
4. ¿Para qué son útiles las etiquetas?
5. ¿Para qué se utiliza la propiedad **AutoSize**, del control **Label**?
6. ¿Cuál es el control que permite elegir una opción entre otras?
7. ¿Para qué se usa el control **ListBox**?
8. ¿Las características de qué controles combina el control **ComboBox**?
9. ¿Para qué se emplea el control **Frame**?
10. ¿Cuál es el evento más importante del **ComandButton**?

EJERCICIOS PRÁCTICOS

1. Crear un formulario que muestre un mensaje de bienvenida en un cuadro de texto cuando se hace clic en un botón de comando.
2. Crear un formulario que multiplique el contenido de dos cuadros de texto y que muestre el resultado en otro cuadro de texto cuando se presiona el botón de comando.
3. Realice un formulario que permita ingresar dos números por el teclado y determinar cuál de los dos valores es el menor. Mostrarlo en un cuadro de texto.
4. Crear un formulario que permita cambiar el color de fondo y la letra del contenido de un cuadro de texto.
5. Crear un formulario que contenga veinte números en un control **ListBox**. Cuando se selecciona uno de ellos, deberá mostrarse ese número en un cuadro de texto.

Índice temático

USERS

MACROS EN EXCEL 2013

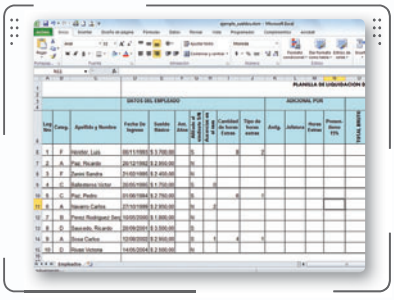
Presentamos un libro ideal para todos aquellos usuarios de Microsoft Excel que quieran iniciarse en la programación de aplicaciones con Visual Basic y, así, ampliar la funcionalidad de sus planillas de cálculo. A lo largo de su contenido, conoceremos los conceptos básicos de programación que nos permitirán generar nuestras propias macros, para luego trabajar sobre nociones más específicas, que nos ayuden a agilizar el trabajo diario. También conoceremos nuevas funciones y procedimientos destinados manipular cada objeto del programa, desde el libro y las celdas, hasta el uso de formularios más complejos.

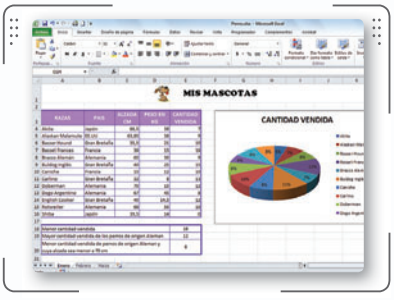
A través de explicaciones sencillas, guías visuales y procedimientos paso a paso, el lector descubrirá una obra que le permitirá desarrollar sus propias macros y aplicaciones VBA en Microsoft Excel.

> **Gracias al lenguaje Visual Basic para Aplicaciones, podemos automatizar nuestras tareas, y así, ahorrar tiempo y esfuerzo.**

EN ESTE LIBRO APRENDERÁ:

- **Automatización:** cuáles son los aspectos básicos del lenguaje VBA para Excel y cómo funciona su modelo de objetos.
- **Editor de VBA:** características del entorno de programación, las ventanas, las barras de herramientas y su funcionamiento en general.
- **Sentencias:** cómo escribir los procedimientos para introducir datos y visualizar resultados. Aspectos para tener en cuenta al exportar o importar módulos.
- **Datos y funciones:** clases de datos utilizados por el lenguaje VBA, sus variables y operadores. Trabajo con funciones predefinidas básicas.
- **Estructuras de programación:** control del flujo de ejecución del programa, estructuras condicionales y repetición de operaciones.
- **Objetos y formularios:** cuáles son los principales objetos de Excel, sus métodos y propiedades. Cómo crear y programar formularios.





SOBRE LA AUTORA

Viviana Zanini es Analista de Sistemas de Computación y profesora de Informática. Ha realizado diferentes cursos de especialización en el área de programación y guías de estudio. También ha colaborado como autora en la colección de fascículos Curso visual y práctico Excel, de esta misma editorial.

NIVEL DE USUARIO

Básico / Intermedio

CATEGORÍA

Desarrollo / Excel / Microsoft

ISBN 978-987-1857-99-9

RU RedUSERS

REDUSERS.com
En nuestro sitio podrá encontrar noticias relacionadas y también participar de la comunidad de tecnología más importante de América Latina.

PROFESOR EN LÍNEA
Ante cualquier consulta técnica relacionada con el libro, puede contactarse con nuestros expertos: profesor@redusers.com.