Manual de Macros VBA
Del Estudiante

Contenido

Qué es una macro de Excel

Microsoft Excel se cuenta entre las soluciones más demandadas a la hora de editar, analizar y presentar datos. Los usuarios de Windows pueden utilizar las populares hojas de cálculo, incluidas en la suite ofimática de Microsoft, para, entre otras cosas, planificar un presupuesto o crear un calendario personal. En el entorno laboral Excel es también una herramienta muy utilizada, pues no solo permite elaborar planes de proyectos, registros de horas de trabajo o planes presupuestarios, sino también e igual de fácilmente, representaciones gráficas de cifras de ventas, de beneficios o de pérdidas. Una vez se ha aprendido a utilizar el programa, se aprecian pronto sus numerosas funciones, aunque también es fácil desarrollar una cierta aversión hacia tareas rutinarias y repetitivas o ante acciones que no se puedan ejecutar fácilmente con ayuda de la interfaz estándar.

No es extraño, por esto, que la posibilidad de crear macros se cuente entre las características cruciales del programa de cálculo. La herramienta integrada en Excel para grabar macros utiliza el lenguaje de scripts **Visual Basic for Applications (VBA),** que también se implementa en Word, Powerpoint, Access y Outlook. Gracias a este lenguaje es posible crear macros en Excel capaces de ejecutar de forma automática comandos rutinarios o incluso añadir nuevas funciones (algoritmos para el análisis de datos) a la hora de cálculo.

Así funcionan las macros en Excel

Excel dispone los diversos elementos que componen la interfaz de uso como objetos-programa organizados jerárquicamente, cada uno de los cuales posee propiedades y métodos específicos. En este sistema jerárquico, todos los objetos disponibles se encuentran conectados entre sí y se ven reflejados de forma aproximada en la interfaz de usuario, que proporciona **los mandos necesarios para poder interactuar con la aplicación**. Estos objetos pueden editarse determinando sus propiedades y asignándoles métodos. Por ejemplo, para el objeto “Workbook” existen los métodos “Close”, con cuya ayuda se cierra el libro seleccionado, así como la propiedad “ActiveSheet”, que muestra la hoja activa en el libro de trabajo.

Mediante las **listas** (objetos señalados con el sufijo plural -s) las macros también pueden ejecutar acciones sobre un grupo de objetos. El objeto para listas “Worksheets” en una macro tiene como resultado que las instrucciones se aplican a todas las hojas de trabajo. Para ejecutar una macro se tienen estas tres opciones:

- Seleccionar la opción correspondiente en el **menú de macros**
- Seleccionar una **tecla** creada por el usuario
- Un **atajo de teclado** personal

Las ventajas de utilizar macros con Excel

Si, con anterioridad, consciente o inconscientemente, no se ha aprovechado la utilidad de las macros de Excel, no significa esto que haya sido en detrimento del trabajo con archivos de Excel, pero sí que se ha elegido el camino más difícil. Y es que saltan a la vista las ventajas decisivas de la tecnología de macros que las convierten en una tarea obligatoria para todo aquel que quiera sacar el máximo partido al software de hojas de cálculo. Esto es lo que aporta crear macros con Excel:

- Reducen la tasa de errores: con cada comando que se introduce manualmente en la hoja de cálculo aumenta la probabilidad de cometer un error y es que, especialmente cuando se trata de secuencias complejas de comandos y de acciones repetitivas, es muy fácil dar un paso en falso que bloquee incluso al programa al completo. Grabando macros, en cambio, solo esconde un potencial de error el propio proceso de su creación, pero si se hizo todo correctamente, la macro funciona siempre a la perfección.

- Reducen el tiempo de trabajo: como las macros se crean una sola vez y se utilizan siempre que se necesite, ahorran un tiempo valioso que es posible dedicar a otras tareas.

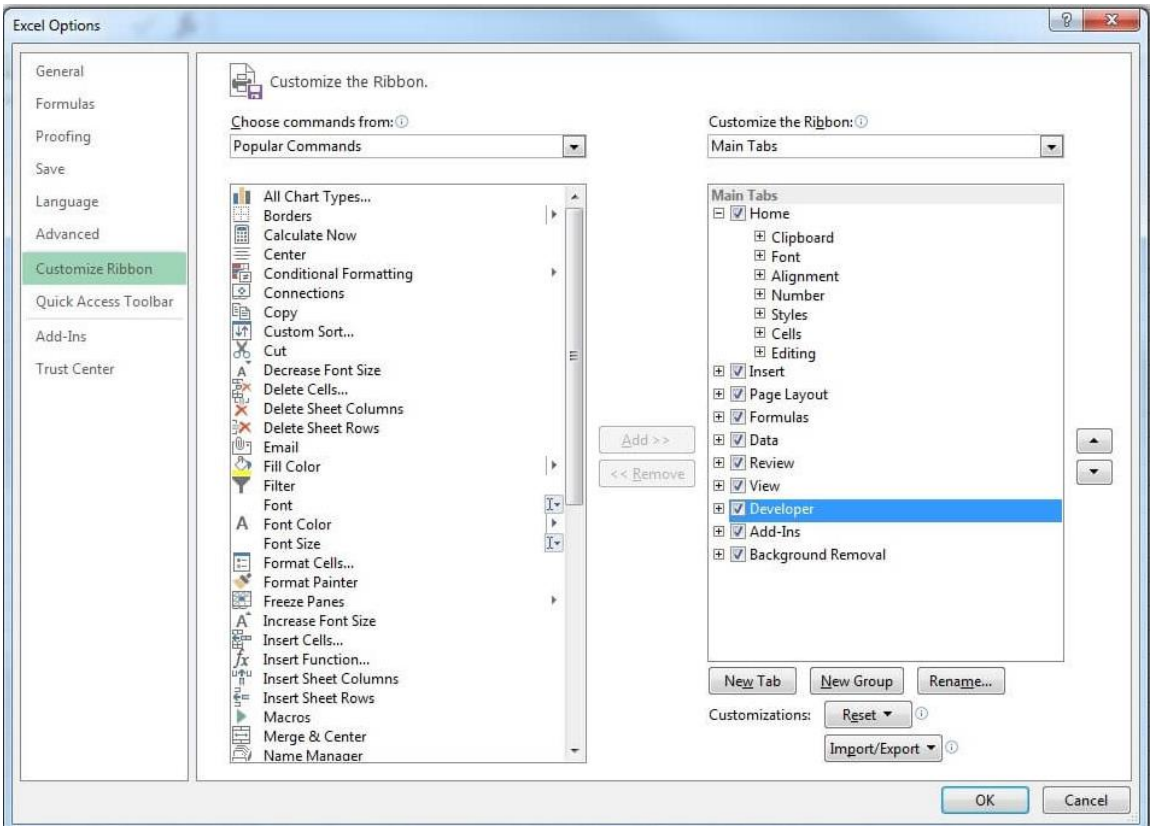
- Aumentan la utilidad de Excel: Con Visual Basic no solo se pueden programar macros, sino también desarrollar nuevas funciones. De este modo el usuario tiene la posibilidad de ampliar el abanico de funciones con aquellas adecuadas a sus necesidades y de simplificar de forma considerable sus fórmulas. Algo muy práctico es que Excel presenta estas funciones definidas por el usuario como si fueran nativas del programa y, más aún, el usuario puede fijar botones de acceso rápido a sus propias macros en la barra de símbolos.

Crear macros en Excel: Paso a Paso

Para crear macros propias hay que activar el editor Visual Basic, que forma parte de las herramientas para desarrolladores y no está, por defecto, disponible en la cinta de opciones (la antigua barra de herramientas). En un primer paso, entonces, se debe añadir la pestaña o ficha denominada Developer (Programador).

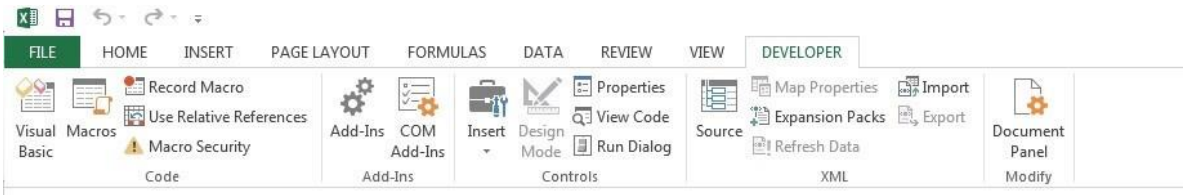
Primer paso: añadir la ficha Developer a la cinta de opciones

Dirígete a la pestaña File (Archivo) y selecciona Options (Opciones). En el punto Customize the Ribbon (Personalizar la cinta de opciones) se encuentra un listado de todas las pestañas principales o Main Tabs, entre las que se encuentra Developer (Programador):



Para poder grabar macros hay que añadir la pestaña Developer (Programador o Desarrollador) a la cinta de opciones.

Marca la casilla de Developer y confirma haciendo clic en OK. Ahora la pestaña ya se encuentra en la cinta de opciones.



Vista de la cinta de opciones con la pestaña Developer

Segundo paso: Grabar una macro

Grabar una macro

1. En el grupo **Código** en la pestaña **Programador**, haga clic en **Grabar macro**.
2. De manera opcional, escriba un nombre para la macro en el cuadro **Nombre de la macro**, especifique una tecla de método abreviado en el cuadro **Tecla de método abreviado**, y una descripción en el cuadro **Descripción**. A continuación, haga clic en **Aceptar** para comenzar a grabar.

3. Realice las acciones que desee automatizar, como escribir texto repetitivo o completar hacia abajo una columna de datos.
4. En la pestaña **Programador**, haga clic en **Detener grabación**.

Examine la macro y pruébela

Al modificar una macro, puede aprender un poco acerca del lenguaje de programación Visual Basic.

Para modificar una macro, en el grupo **Código** en la pestaña **Programador**, haga clic en **Macros**, seleccione el nombre de la macro y haga clic en **Editar**. Esta acción hará que se inicie el Editor de Visual Basic.

Observe el código y vea de qué manera las acciones que ha grabado aparecen como código. Es probable que entienda bien algo del código y que otra parte le resulte un poco misteriosa.

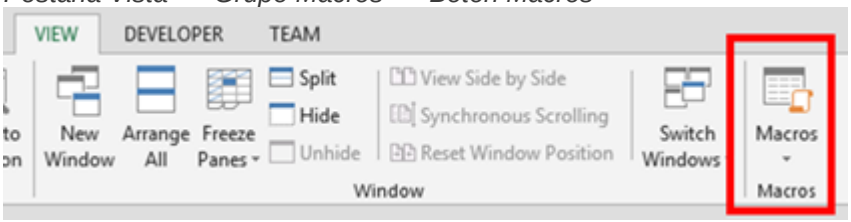
Experimente con el código, cierre el Editor de Visual Basic y ejecute la macro nuevamente. Esta vez observe si sucede algo distinto.

5 maneras de Ejecutar una macro

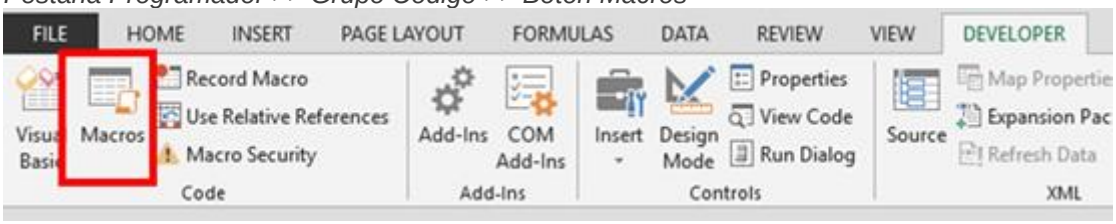
1 – EJECUTAR UNA MACRO DESDE LA BARRA DE HERRAMIENTAS

Para ejecutar una macro desde la barra de marcadores debes ir a:

Pestaña Vista >> Grupo Macros >> Botón Macros



También se puede ejecutar desde la pestaña *Programador* (si no la tienes activada te explicamos cómo hacerlo aquí).
Pestaña Programador >> Grupo Código >> Botón Macros

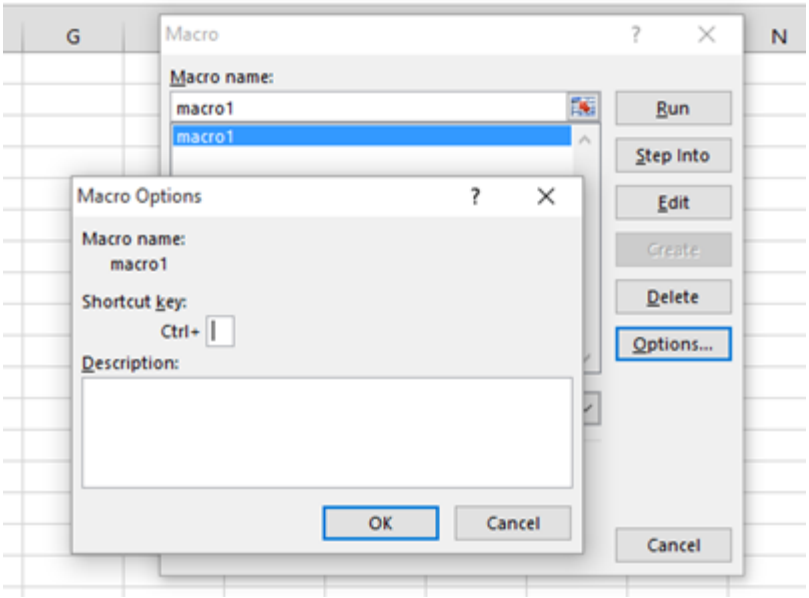


2 – EJECUTAR UN MACRO DESDE EL EDITOR DE VISUAL BASIC (VBE)

Para ello lo primero que tienes que hacer es abrir el Editor de Visual Basic (VBE) desde la pestaña *Programador* o simplemente pulsando Alt + F11
Una vez abierto VBE puedes ejecutar la macro seleccionada pulsando F5 o dándole al botón de Play (►) en la barra de herramientas.

3 – EJECUTAR UNA MACRO CON UN ACCESO RÁPIDO (CTRL + TECLA)

Para asignar un acceso rápido a un macro lo primero que tienes que hacer es abrir la ventana de diálogo de macros (en la pestaña *Vista* o *Programador*). Una vez abierta esta ventana selecciona la macro a la que quieras asignar un acceso rápido y a continuación pulsa el botón *Opciones*. En la nueva ventana que se abre escribe la letra que quieres que ejecute la macro (no todas valen, como es lógico, ya hay muchas que están cogidas como por ejemplo el popular Ctrl + C) y pulsa Aceptar.



4 – EJECUTAR UNA MACRO DESDE OTRA MACRO

Una macro se puede ejecutar desde otra macro utilizando la **instrucción Call**. Por ejemplo, como podemos ver en el siguiente procedimiento.

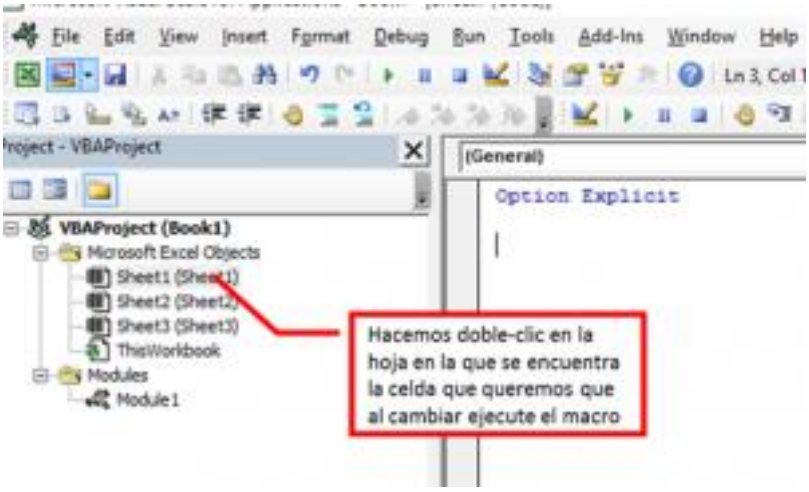
```
Sub macro1()

    Call macro2

End Sub



Sub macro2()

    MsgBox ("Has ejectuado el macro1")

End Sub
```

En esta macro que acabamos de ver, la Macro1 llama a la Macro2 mediante la instrucción Call. Es decir, al ejecutar la macro1 esta irá directamente a ejecutar la macro2 cuando llegue a la instrucción Call.

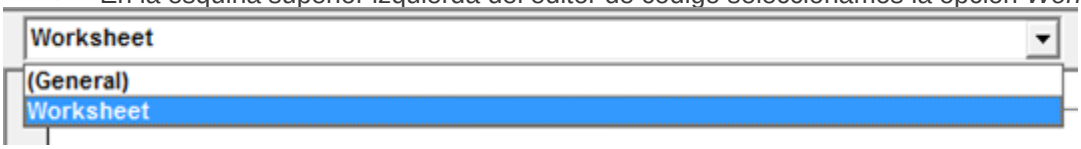
5 – EJECUTAR UNA MACRO CUANDO CAMBIE EL VALOR DE UNA CELDA

Cambiar el valor de una celda en VBA se considera un *evento.* ¿Sabes lo que son los eventos? Si quieres aprender más de este tema puedes leer el artículo: ¿Qué son los eventos en VBA? Hay diferentes tipos de eventos, como pueden ser guardar el documento, abrirlo, añadir una hoja etc. Todos estos eventos ya vienen predefinidos en VBA. El ejemplo a continuación explica el evento *Change* o cambio de valor de una celda. El procedimiento sería similar para el resto de eventos:
Los pasos a seguir son los siguientes:

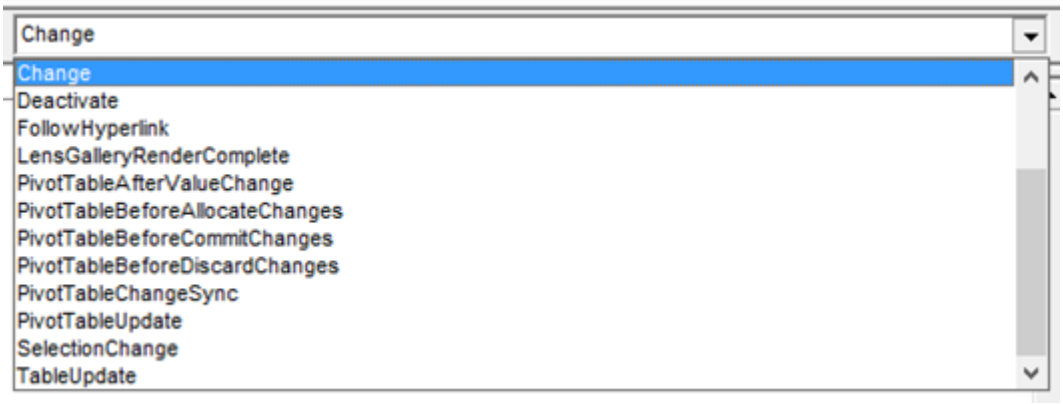
- Abrir VBE (Alt + F11)
- Para el cambio de valor de una celda tenemos que escribir el código en la hoja en la que se encuentra esa celda.



- En la esquina superior izquierda del editor de código seleccionamos la opción *Worksheet*



- En la parte superior derecha seleccionamos el evento que queremos que ejecute la macro, en nuestro caso es un cambio de valor en una celda o *Change:*



- Una vez seleccionado el evento, VBE escribe automaticamente la iniciación del macro:

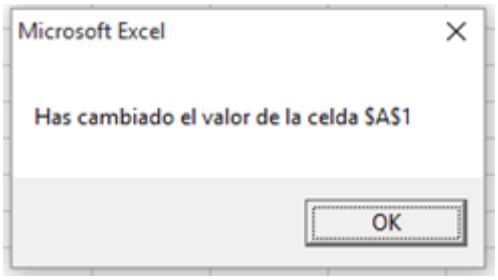
```
Private Sub Worksheet_Change(ByVal Target As Range)

'mi código

End Sub
```

- A continuación lo único que tenemos que hacer es definir el target (celda que al cambiar ejecute el macro) y escribir el código que queremos que ejectue cuando ese valor cambie:

```
Private Sub Worksheet_Change(ByVal Target As Range)
```

```
If Target.Address = "$A$1" Then

MsgBox ("Has cambiado el valor de la celda " & Target.Address)

End If



End Sub
```

A partir de ahora cada vez que cambiemos el valor de la celda A1 nos aparecerá el siguiente mensaje en pantalla:



Nota: El evento change se refiere al cambio de valor de una celda. Cuando la celda contiene una formula y lo que cambia es el resultado de la formula la macro NO se ejecutará ya que el contenido de la celda (en este caso una fórmula) sigue siendo la misma. Para este caso el evento que queremos generar es el evento *Calculate*.

El editor de Visual Basic

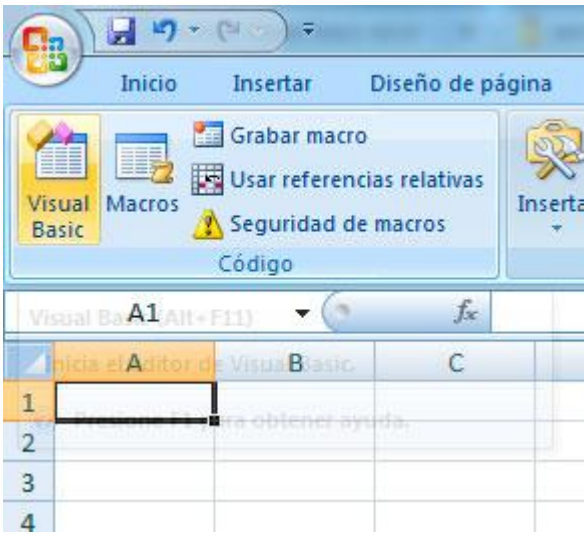
El **editor de Visual Basic en Excel** es un programa que no depende de la aplicación de **Office**, pero que nos permitirá crear **macros para Excel**. Además, con el siguiente tutorial de Visual Basic para Excel podremos abrir el **editor de Visual Basic** de distintas maneras y comenzar a explorar los proyectos, insertar módulos o escribir el código VBA directamente. Se podrán utilizar las distintas plantillas de Excel para después emplear el editor de Visual Basic en Excel y ejecutar distintas acciones.

Abrir el editor de Visual Basic en Excel

Podemos **abrir el editor de Visual Basic en Excel** de diferentes formas y os explicaremos cómo hacerlo para que conozcáis varias opciones para realizar esta acción. Para **abrir el editor de VBA en Excel** pulsaremos el botón de Visual Basic de la ficha de **Programador** o con la combinación de teclas adecuada.

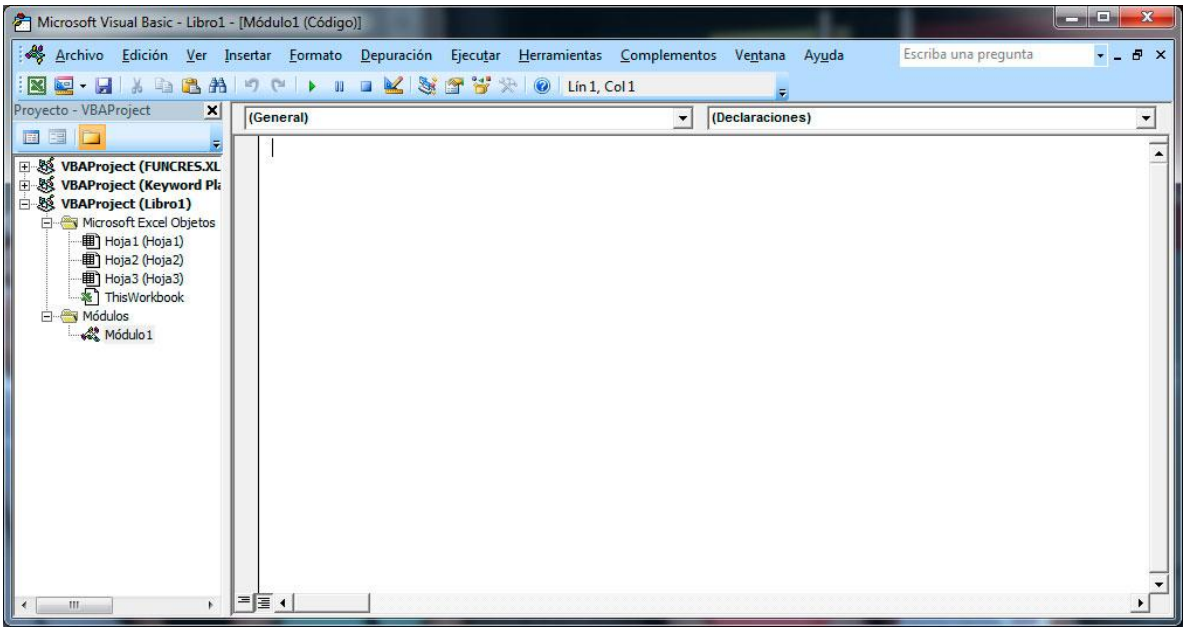
Botón de Visual Basic en Excel

La primera de las opciones para **abrir el editor de Visual Basic en Excel** es pulsando el botón de VB en Excel. Para ello pulsaremos sobre el **botón Visual Basic de la ficha de programador**. Automáticamente, se abrirá el editor y podremos utilizar los menús y herramientas de la ventana seguimos los mismos pasos que al crear macros en Excel.



Atajo con el teclado para abrir editor de Visual Basic en Excel

La segunda de las opciones para **abrir el editor de Visual Basic en Excel** será utilizando la combinación de teclas **ALT+F11** permitiéndonos abrir la ventana de forma rápida.



Editor de Visual Basic en Excel

La diferencia entre el **editor de Visual Basic en Excel 2007** y el **editor de Visual Basic en Excel 2010**, es la forma en que activamos el editor. En **Excel 2010** seleccionaremos en el menú Archivo el botón opciones. Seguidamente, pulsaremos la opción Personalizar cinta de opciones y escogeremos fichas principales en el desplegable de la parte derecha. Entre las fichas principales elegiremos la opción Programador para activar dicho menú y poder hacer uso del **Editor de Visual Basic en Excel 2010**.

Editor de Visual Basic en Excel 2016

En **Excel 2016 la pestaña que contiene Visual Basic y los macros se llama Desarrollador.** Tienes que comprobar si está activada y se muestra en la cinta de opciones, para ello tienes que mirar al lado de la pestaña Vista, si no la ves tendrás que activarla.

1. Acudimos a la pestaña Archivo
2. Seleccionamos "Opciones"
3. Clicamos sobre el botón "Personalizar cinta de opciones"
4. Buscamos entre el listado que aparece la pestaña "Desarrollador" y en concreto el botón "Visual Basic"
5. Seleccionamos aceptar.

Siguiendo estos pasos obtendremos un acceso directo a la pestaña desarrollador desde la cinta de opciones.

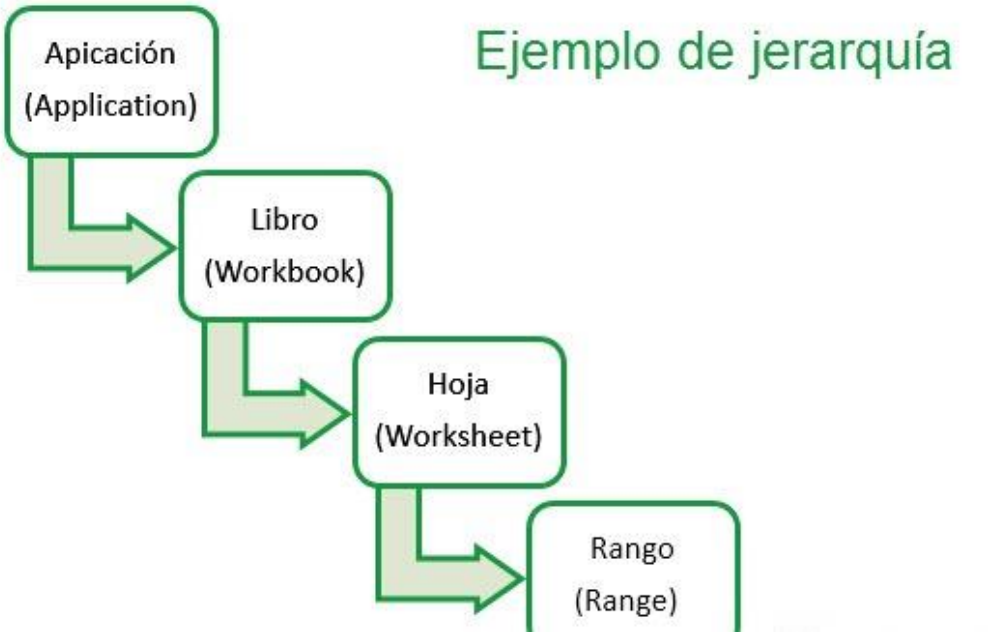
Programación orientada a objetos

Un libro de Excel está lleno de objetos que se organizan mediante una jerarquía. La programación orientada a objetos se basa en la modificación de las propiedades y los métodos de estos objetos para que realicen las acciones que quieras.
En el mundo real, un objeto es algo tangible, como puede ser un coche o un gato y se identifica porque tiene propiedades y métodos. Se considera que las propiedades son las características y los métodos son las acciones que puede realizar. En un coche, una propiedad sería el color y un método sería la posibilidad de girar el volante hacia la izquierda.

Excel cuenta con más de 200 objetos. **Cada elemento de Excel es un objeto**. Un libro (Workbook), una hoja (Worksheet), un gráfico (Chart) o una tabla dinámica (PivotTable) son algunos ejemplos de objetos dentro del modelo de Excel.
Muchos de los objetos se agrupan en colecciones. En el caso del objeto coche, una colección representaría todos los coches del mundo. En cuanto a Excel, la colección Workbooks haría referencia a todos los libros de Excel abiertos.

Como te digo, **los objetos del modelo de Excel están organizados por jerarquías**, que significa que unos objetos pueden contener a otros objetos.
El ejemplo más descriptivo para que entiendas este concepto es el siguiente:

Cuando abres el programa Excel, en realidad estás abriendo el objeto **Application** y al mismo tiempo estás abriendo un libro en blanco representado por el objeto **Workbook**. Este libro contiene una hoja activa, representado por el objeto **Worksheet**, que a su vez contiene un conjunto de celdas, representado por el objeto **Range**.



Gracias a esta organización puedes hacer referencia a cualquiera de los objetos siguiendo la posición jerárquica que ocupa dentro del modelo de objetos.

Por ejemplo, para hacer referencia al libro “Ventas anuales.xlsx” habría que hacerlo de la siguiente forma:

```
Application.Workbooks (“Ventas anuales.xlsx”)
```

Si quisieras seleccionar la celda B2 de la hoja “Totales” del libro “Ventas anuales.xlsx”, tendrías que hacerlo siguiendo toda la jerarquía:

```
Application.Workbooks(“Ventas anuales.xlsx”). _
```

```
Worksheets("Totales").Range("B2").Select
```

La instrucción anterior se puede simplificar en la mayoría de los casos omitiendo el objeto **Application**. Esto es posible porque la instrucción está creada dentro de Excel (**Application**).

```
Workbooks("Ventas anuales.xlsx"). _
Worksheets("Totales").Range("B2").Select
```

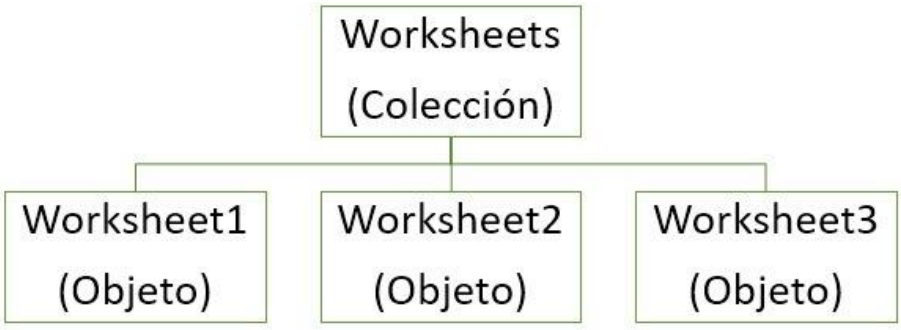
Como ves, si el objeto de mayor rango es el objeto activo, puedes omitirlo de la referencia. Si tienes activo el libro "Ventas anuales.xlsx", podrías seleccionar la celda B2 de la siguiente forma:

```
Worksheets("Totales").Range("B2").Select
```

Bajando un poco más por la jerarquía, si la hoja activa fuese "Totales", podrías seleccionar la celda B2 así:

```
Range("B2").Select
```

Como he dicho antes, todos **los objetos del mismo tipo forman una colección**. Las colecciones permiten trabajar con un grupo de objetos como si de un solo objeto se tratara. Normalmente, el nombre de una colección es el objeto en plural de los objetos que contiene.
Por ejemplo, **Worksheets** es el nombre de la colección que reúne todos los objetos **Worksheet**.



Debes entender que una colección es siempre dinámica, ya que se pueden agregar o eliminar objetos, como en el caso de las hojas de un libro.

Para hacer referencia a un objeto dentro de una colección, puedes utilizar los siguientes métodos:

```
Coleccion!Objeto
```

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

Var es una variable del tipo **String** que contiene el nombre del objeto. **Index** es el número de la posición que ocupa el objeto dentro de la colección.

Propiedades, métodos y eventos de los objetos de Excel

Al principio comenté que la programación orientada a objetos se basaba en la modificación de las propiedades y métodos de los objetos.

Las propiedades son las características propias del objeto, que hace que se distinga de otros objetos (nombre, tamaño, color, ubicación en la pantalla…)
Por ejemplo, propiedades del objeto Range pueden ser Name (nombre), Value (valor) y Column (columna), entre muchos otros.

El uso de las diferentes propiedades te permitirá modificar las características del objeto. Por ejemplo, puedes utilizar la propiedad **Value** para modificar el valor que se muestra en una celda. También es posible utilizar las propiedades para modificar el aspecto de un objeto. Por ejemplo, la propiedad **Borders** permite cambiar el borde de la celda.
Algunas propiedades pueden ser a su vez objetos. Por ejemplo, si quieres cambiarle el tipo de fuente al objeto **Range** debes utilizar la propiedad **Font**. Como las fuentes tienen diferentes nombres, tamaños o estilos, existen otras propiedades que califican a la propiedad **Font**.
Por ejemplo, si quieres cambiarle el tamaño de fuente a la celda B2, debes escribir lo siguiente:

```
Range("B2").Font.Size = 25
```

Por su parte, **se denominan métodos a las acciones que puedes hacer con un objeto**. Se puede decir que son órdenes que se le dan a los objetos para que haga algo sobre sí mismo. De esta forma, el objeto **Range** tiene los métodos **Activate** (activar) y **Clear** (borrar), entre muchos otros.
La implementación de un método en el objeto es muy sencilla. Esta es la sintaxis:

```
Objeto.Método
```

De esta forma, si quieres seleccionar la celda B2 de la hoja activa, tendrías que utilizar la siguiente instrucción:

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

Este es un ejemplo sencillo. Algunos otros métodos admiten argumentos, que son parámetros que permiten especificar mejor las opciones de la acción que debe realizar el objeto. Por ejemplo, si quieres guardar el libro actual con el nombre "Ventas 2016" tienes que utilizar la siguiente instrucción:

```
ThisWorkbook.SaveAs Filename:="Ventas 2016.xlsm"
```

Además de las propiedades y los métodos, algunos de los objetos también pueden reaccionar ante eventos.

Un evento se da cuando sucede una determinada situación. Por ejemplo, abrir un libro, imprimir o cerrar una ventana son eventos.
Con VBA es posible programar acciones cuando se produce un evento.

Imagina que cada vez que abras un libro, quieres que aparezca un mensaje de bienvenida. Para ello se debe trabajar dentro del evento Open del objeto Workbook de la siguiente manera:

```
Private Sub Workbook_Open()
MsgBox "¡Bienvenido!"
End Sub
```

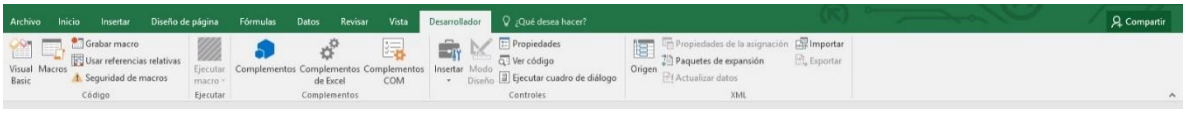
Aunque esto es algo complejo de entender, con un poco de práctica conseguirás dominarlo.

La ficha Desarrollador

¿No encuentras la ficha Desarrollador en la cinta de opciones? Eso es porque **al instalar Excel, se encuentra oculta**. Obviamente debes mostrarla antes de comenzar a trabajar con ella.

Para mostrar la ficha te recomiendo que hagas clic con el botón derecho en cualquier parte de la cinta de opciones y selecciones **Personalizar la cinta de opciones**. A continuación, activa la casilla de verificación correspondiente en la parte de la derecha del cuadro de diálogo.

En la ficha Desarrollador (o Programador, según la versión de Excel que utilices) se encuentran los comandos que utilizarás para gestionar los complementos y los controles.



Si vas a programar macros te recomiendo que la tengas siempre visible en la pantalla.

Formato de archivo para macros

La potencia que tienen las macros y el lenguaje VBA en general, hacen que un Libro de Excel sea la puerta perfecta para que nuestros equipos se infecten con virus.

Microsoft tomó medidas hace ya varios años para prevenir este uso indebido de Excel, creando un **formato de archivo especial para guardar los libros que contienen macros**. Además, incluyó varios niveles de seguridad.

La extensión **.xlsm** fue la elegida para designar a los archivos con macros. En las versiones anteriores a Excel 2007 era imperceptible a menos que tuvieras habilitados los niveles de seguridad que respectan a las macros.

Ahora, con sólo comprobar la extensión del libro podrás determinar si contiene alguna macro y así decidir si quieres abrirlo o no, en función de la confianza que te genere.

Siempre que abras un libro que contenga VBA, se mostrará de forma predeterminada y debajo de la cinta de opciones un mensaje en el que se indica que se han deshabilitado las macros.

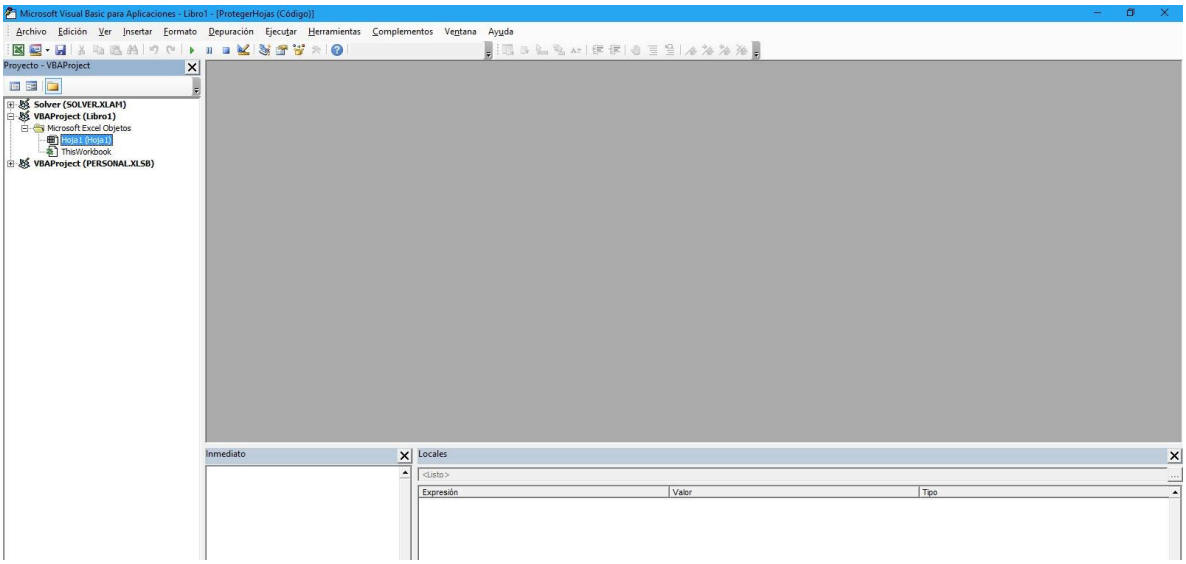
Hace unos años se pusieron de moda los virus dentro de macros. Estos virus se aprovechaban de archivos aparentemente inofensivos para hacer varios tipos de ataque.

Para poder trabajar con archivos que contienen macros, debes configurar las **restricciones de seguridad de Excel desde el Centro de Confianza**. Para ver qué configuración tiene Excel en el momento, pulsa en el botón **Seguridad de macros** que se encuentra en el grupo **Código** de la ficha **Desarrollador**.

El editor de VBA

El editor de Visual Basic for Applications es el lugar donde se desarrollan, prueban y modifican las macros. Se ejecuta en una ventana diferente a la de Excel y se accede a ella de estas tres formas:

- Haciendo clic en el botón **Visual Basic** de la ficha **Desarrollador**.
- Cuando ya existe la macro, pulsando el botón **Modificar** del cuadro de diálogo **Macros**, que aparece al pulsar el botón **Macros** de la ficha **Desarrollador**.
- Pulsando **Alt + F11**.



La ventana del editor muestra todo lo necesario para comenzar a crear una macro.

- **La barra de menús**: Desde aquí se accede a la mayoría de funciones de VBE para desarrollar, comprobar y guardar las macros.
- **La barra de herramientas estándar**: Contiene los botones con los comandos más utilizados.
- **La barra de herramientas de Edición**: Aquí se encuentran los comandos más útiles cuando se está escribiendo el código.
- **El explorador de proyectos**: Se muestra un árbol con todos los archivos que se encuentran abiertos, que contienen los componentes de los proyectos.
- **Ventana de código**: Es la que se emplea para escribir todo el código VBA. Cada elemento de un proyecto tiene asignada su propia ventana de código.
- **La ventana propiedades**: permite cambiar las propiedades del objeto seleccionado, mientras estás en Modo Diseño.
- **La ventana Inmediato**: Permite probar una instrucción estando en **Modo Diseño**. No se muestra por defecto pero puedes visualizarla pulsando **Ctrl + G**.
- **La ventana Locales**: Permite comprobar el valor de una variable en cualquier momento de la ejecución de la macro. Para mostrar esta ventana, debes pulsar el botón **Ventana Locales** del menú **Ver**.

- **La ventana Inspección**: Permite agregar objetos para ver el valor actual de una variable cuando estás en **Modo Interrupción**. Para mostrarla, pulsa el botón correspondiente del menú **Ver**.
- **El Examinador de objetos**: Es de gran ayuda para encontrar los objetos, sus propiedades y métodos asociados. Para mostrarlo pulsa **F2**.

  Como en la mayoría de aplicaciones, puedes modificar a tu gusto todos los elementos de la ventana del editor. Puedes hacer estas modificaciones pulsando en el botón **Opciones** del menú **Herramientas**.

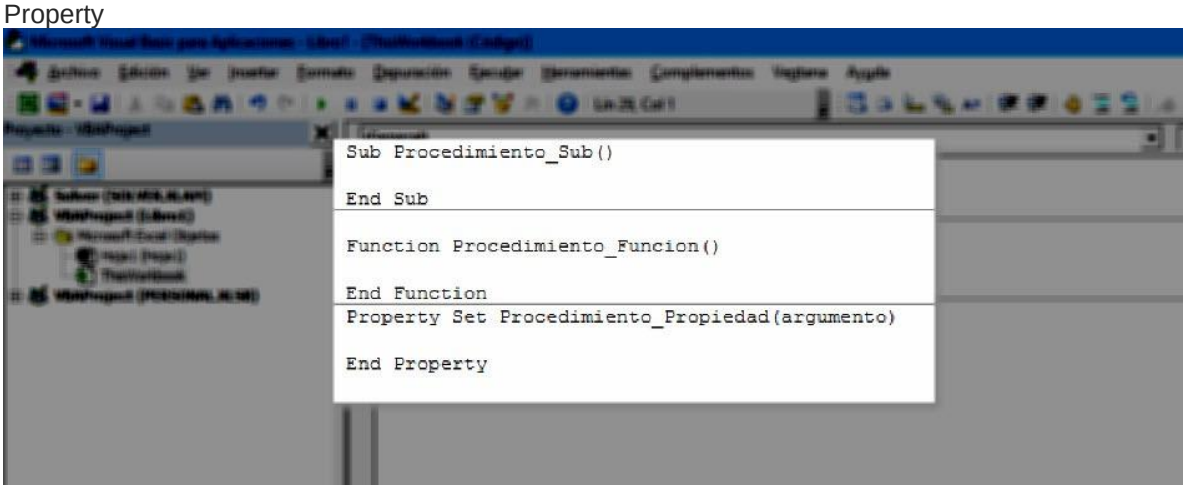
  Es posible modificar los siguientes aspectos:

- La tipografía de cada tipo de instrucción (palabras clave, instrucciones, comentarios…).
- La introducción del código.
- El acople de las ventanas.
- La gestión de errores.

Los procedimientos en VBA

Como te mostré antes, la ventana del **Explorador de Proyectos** muestra todos los objetos que pueden contener código VBA (los objetos **Hojas**, **ThisWorkbook**, **Módulos**, **Userforms**). Dentro de estos objetos se ubican los procedimientos. En Excel existen tres tipos de procedimiento:

- Sub
- Function
- Property



De forma predeterminada, los procedimientos son públicos (**Public**), en todos los módulos. Esto significa que se los puede llamar desde cualquier parte del proyecto.
Si estableces el procedimiento como **Private**, sólo podrá ser llamado desde otros procedimientos que se encuentren en el mismo módulo.

El procedimiento Sub

Se puede decir que un procedimiento Sub es un conjunto de instrucciones que realizan una serie de acciones específicas.

Hay dos tipos de procedimientos Sub:

- Procedimientos generales: Son los que se declaran dentro de un módulo.
- Procedimientos de evento: Se ejecuta automáticamente cuando se produce una determinada situación o contexto.

Todos los procedimientos Sub comienzan con la palabra Sub seguida del nombre que le quieras dar, y finalizan con la instrucción **End Sub**.
La instrucción Sub también puede ir precedida de otras palabras que determinan el ámbito, y puede contener una lista de argumentos para que el procedimiento pueda usarlos de manera interna.

El procedimiento Function

Ya sabes que Excel cuenta con muchas funciones de hoja predefinidas como **SUMA**, **BUSCARV** o **SUMAR.SI**. Pero a veces es necesario realizar cálculos más complejos para los que no existe una función.
Gracias a los procedimientos Function, puedes crear nuevas funciones, ampliando así las que ya existen. Al igual que los procedimientos Sub, los Function también admiten argumentos que permiten realizar los cálculos más eficazmente.
Como diferencia de los procedimientos Sub, los procedimientos Function pueden devolver un valor o resultado, por eso, al declarar la función, es necesario especificar qué tipo de variable será el valor que va a devolver el procedimiento.

El procedimiento Property

Este tipo de procedimiento se utiliza para **crear y personalizar las propiedades de los objetos de Excel**. Se declaran automáticamente como públicos aunque es posible hacerlo como privados.

Variables, constantes y otros datos interesantes

Variables

En muchas ocasiones necesitarás **guardar datos de forma temporal para utilizarlos más adelante** en otros cálculos. Por ejemplo, puedes almacenar los valores de venta de un producto cada mes para luego utilizarlo para hallar el total anual.
En general, una variable es una pequeña región de la memoria que se utiliza para guardar valores o información que luego se usará en la ejecución del programa.

Toda la información que se maneja necesita utilizar la memoria RAM del ordenador, por eso es una buena práctica reservar un lugar para guardar una variable antes de utilizarla. Esto se conoce como **declaración de la variable** y, aunque no es totalmente necesario, **te ayudará a entender mejor el código** una vez escrito.
Puedes declarar una variable de dos formas diferentes:

- **Forma implícita**: La variable se declara en el momento en el que se va a utilizar. Se hace mediante la asignación de un valor al nombre.

```
X = 13
Mes3 = “Marzo”
Total_anual = 32000
```

- **Forma explícita**: Es necesario definir previamente el nombre de la variable antes de utilizarla. Para declarar una variable de esta forma debes escribir la palabra **Dim** seguida del nombre de la variable. **Dim Empleado**.
  Existe una forma de obligarte a declarar las variables y es introducir la instrucción **Option Explicit** al comienzo del módulo, fuera de cualquier procedimiento. Esto hará que cada vez que VBA encuentre una variable sin declarar, te informe de que debes declararla antes de utilizarla.

Tipos de variables VBA

Byte

Se utiliza para guardar números positivos enteros desde 0 a 255. Ocupa 1 byte en la memoria.

Boolean

Sólo puede almacenar dos valores: Verdadero (1) o Falso (0). Ocupa 2 bytes en la memoria

Integer

Este tipo de variable también ocupa 2 bytes pero puede almacenar un rango de números enteros muy alto: desde -32.768 hasta 32.767.

Long

También almacena números enteros desde -2.147.483.648 hasta 2.147.483.647.

Currency

Se utiliza para cálculos donde intervienen monedas. Puede almacenar un rango desde -922.334.203.685.477,5808 hasta 922.337.203.685.477,5807. Ocupa 8 bytes en memoria.

Single

Se suele utiliza para almacenar números fraccionarios periódicos. Abarca desde -3,4028235E+38 a -1,401298E-45 para números negativos y 1,401298E-45 a 3,4028235E+38 para números positivos. Ocupa 4 bytes en memoria.

Double

Similar al anterior pero con mucha más capacidad. Ocupa 8 bytes en memoria y comprende desde -1,79769313486231570E+308 a -4,94065645841246544E-324 para los valores negativos y desde 4,94065645841246544E-324 a 1,79769313486231570E+308 para los valores positivos.

Date

Almacena fechas y horas como números de serie. Estos números funcionan igual que en Excel: la parte positiva del número corresponde al día y la parte decimal corresponde a la hora, de forma que 04/04/1986 a las 16:32:15, está representado por el número 31506,6890625.

String

Almacena cadenas de caracteres desde uno solo a unos dos mil millones aproximadamente. El espacio ocupado por este tipo de variable depende de la longitud de la cadena guardada.

Object

Almacena cualquier referencia a objetos. Ocupa 4 bytes en memoria.

Variant

Es un tipo de variable especial que puede almacenar cualquier tipo de dato (numéricos, textos o fechas). Ocupan un espacio variable en la memoria (un tamaño fijo de 22 bytes + la longitud de los datos). Si una variable no se declara como un tipo de los anteriores, se supone que es una Variant.

Constantes

Las constantes, como lo indica su nombre, son valores que nunca cambian, que mantienen su valor durante toda la ejecución de la macro. Normalmente se utilizan para almacenar valores difíciles de recordar y que representan constantes (por ejemplo, la velocidad de la luz, que son 299 792 458 m/s, o el número pi, que es 3.14159265358979323846264338327950288419 71…)

Declarar una constante se hace de la misma forma que lo harías con una variable.

Operadores

Una operación simple puede ser **4+5=9**. En esta expresión, el 4 y el 5 son los operandos y el signo **+** el operador. Este es el ejemplo más simple que se me ocurre. Se utilizan para modificar un valor o para crear uno nuevo.
VBA cuenta con varios tipos de operadores:

- **Aritméticos**: Son utilizados para hacer operaciones matemáticas, como sumar, restar, multiplicar o dividir.
- **De comparación**: Se utilizan para comparar expresiones devolviendo VERDADERO o FALSO.
- **Lógicos**: Con estos operadores es posible evaluar varias expresiones a la vez, devolviendo VERDADERO o FALSO.

Arrays

Un array, llamado también arreglo, es una estructura que almacena varios datos del mismo tipo ordenados de forma lineal, bajo el mismo nombre.

Un array se caracteriza por:

- Almacena los datos en posiciones de memoria contigua.
- Tiene un solo nombre de variable que representa a varios elementos. Los elementos se diferencian mediante un número de índice.
- Es posible acceder a cualquier elemento del array a través de su índice.
  Según su dimensión o su tamaño un array puede ser:

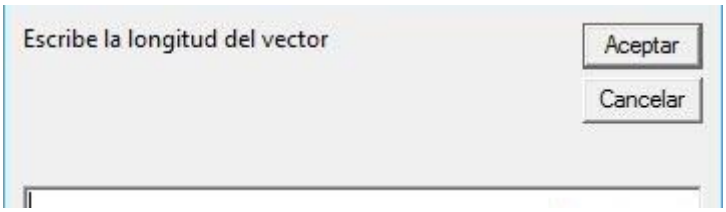
- **De una dimensión**. También llamados vectores. Los datos se guardan de forma lineal.
- **De dos dimensiones**. También llamados matrices. Disponen los datos en forma de tabla.
- **Multidimensional**. Son los que tienen 3 o más índices.
- **Estáticos**. Mantienen siempre el mismo número de elementos. Se utiliza cuando se sabe de antemano el númro exacto de elementos que va a contener.
- **Dinámicos**. Su tamaño puede cambiar en cualquier momento.

Funciones integradas de VBA

Las funciones son uno de los elementos básicos de la programación. Además de las funciones de hoja que proporciona Excel, es posible crear funciones gracias a los procedimientos Function. Además, VBA tiene funciones integradas que puedes utilizar dentro de los procedimientos.

La función InputBox

Es una función que permite interactuar con el usuario solicitándole información mientras se ejecuta una macro. Esto se hace mediante un cuadro de diálogo.



Esta función devuelve los datos introducidos en forma de cadena de caracteres cuando el usuario pulsa el botón **Aceptar** o la tecla **Intro**. Si pulsa el botón **Cancelar** o cierra el cuadro desde la **X** devuelve una cadena de longitud cero (“”).

La función MsgBox

Es la función más utilizada en VBA. Se emplea para obtener una respuesta simple como **Sí** o **No** del usuario y para mostrar mensajes breves como errores y algún dato relevante para la ejecución de la macro.
La función MsgBox hace dos cosas: **Muestra un cuadro de diálogo** para darle información al usuario y **devuelve un valor del tipo Integer** dependiendo del botón pulsado. Si quieres conocer el botón que se ha pulsado debes guardar el valor generado en una variable.

Las funciones de conversión de tipo

Ya sabes que las variables pueden almacenar diferentes tipos de datos. El tipo de datos determina la naturaleza de los valores que toma la variable.

Algunas veces puede que necesites utilizar estos tipos de datos de forma diferente al tipo de dato predeterminado. Para esto se utilizan las funciones de conversión de tipo.

Ten en cuenta que no todos los tipos de datos pueden convertirse a cualquier otro formato. Cuando utilizas una función de conversión de tipo, ésta devuelve el valor del dato convertido pero no cambia el valor almacenado.

Estas son las funciones de conversión:

- **Cbool**: Convierte un número en un valor boolean.
- **Cbyte:** Convierte un número en byte.
- **Ccur:** Convierte un número en moneda.
- **Cdate:** Convierte un número en fecha.

- **Cdbl:** Convierte un número a double.
- **Cdec:** Convierte un número a Decimal.
- **Cint:** Convierte un número a Integer.
- **CLng:** Convierte un número a un entero largo.
- **Csng:** Convierte un número a Single.
- **CStr:** Convierte una variable de cualquier tipo a una de tipo String.
- **Cvar:** Convierte cualquier variable a una de tipo Variant.
- **Val:** Convierte una cadena de texto en un número.

Las funciones de comprobación

Este tipo de funciones ayudan al usuario a comprobar el tipo de dato que contiene una expresión o variable. Estas son las funciones de comprobación:

- **IsDate:** Comprueba si una expresión se puede convertir en una fecha.
- **IsNumeric:** Comprueba si una expresión contiene un valor que se pueda interpretar como un número.
- **IsNull:** Comprueba si una expresión contiene un valor nulo.
- **IsEmpty:** Comprueba si una expresión contiene algún valor o todavía no se han iniciado.
- **IsObject:** Comprueba si una variable representa una variable de tipo Object.

Las funciones matemáticas

VBA también cuenta con funciones matemáticas propias para realizar operaciones matemáticas y que se pueden emplear en los procedimientos. Casi todas ellas coinciden con alguna de las funciones de hoja. Te muestro algunas de ellas:

- **Abs:** Devuelve el valor absoluto de un número.
- **Int:** Devuelve la parte entera de un número decimal.
- **Rnd:** Devuelve un número aleatorio entre 0 y 1.
- **Sqr:** Devuelve la raíz cuadrada de una expresión numérica.

Las funciones de cadena

VBA también te ofrece muchas funciones que te permitirán trabajar con variables de cadena. Estas son algunas de las variables de cadena más utilizadas:

- **Asc:** Devuelve un número enero de 0 255 que representa el valor ASCII de un carácter.
- **Chr:** Es la función inversa de Asc. Devuelve el carácter correspondiente al código ASCII introducido.
- **Len:** Devuelve el número de caracteres de una cadena.
- **Left:** Devuelve un número de caracteres determinado desde la parte izquierda de la cadena.
- **Right:** Devuelve un número de caracteres determinado desde la parte derecha de la cadena.
- **Mid:** Devuelve los n caracteres de una cadena especificada, situados a partir de una determinada posición.
- **Ltrim:** Elimina los espacios iniciales de una cadena.
- **Rtrim:** Elimina los espacios finales de una cadena.
- **Trim:** Quita los espacios iniciales y finales de una cadena.
- **Ucase:** Convierte los caracteres de la cadena en letras mayúsculas.
- **Lcase:** Convierte los caracteres de la caden en letras minúsculas.
- **InStr:** Devuelve la posición de una subcadena dentro de una cadena.
- **Replace:** Sustituye una cadena por otra.

Las funciones de fecha y hora

Con VBA también puedes realizar operaciones relacionadas con fechas y horas. Estas son las funciones más utilizadas:

- **Date:** Devuelve la fecha actual del equipo.
- **Now:** Devuelve la fecha y la hora actuales del equipo.
- **Time:** Devuelve la hora actual del equipo.
- **DateDiff:** Devuelve la diferencia entre dos fechas.

Estructuras de programación

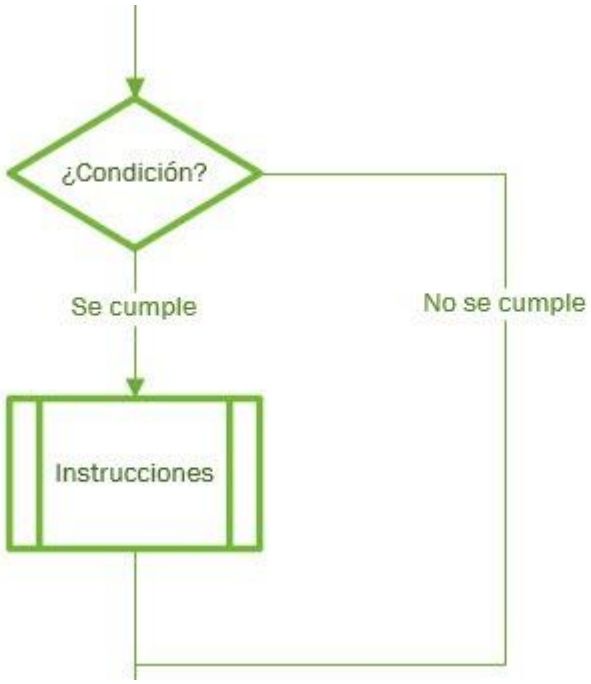
Las estructuras de programación te permiten **controlar el flujo de la ejecución de la macro**, desviándolo según acciones del usuario o repitiendo ciertas instrucciones. Estas estructuras existen en todos los lenguajes de programación y funcionan casi de la misma forma.

Las estructuras condicionales

Este tipo de estructuras se utilizan para tomar decisiones. En ellas se evalúa una expresión y, dependiendo del resultado obtenido, se realizan diferentes acciones. Es posible evaluar expresiones tanto numéricas como de texto.
Las estructuras condicionales o de decisión son las siguientes:

- If – Then
- If – Then – Else
- IIf
- Select Case

La estructura If – Then



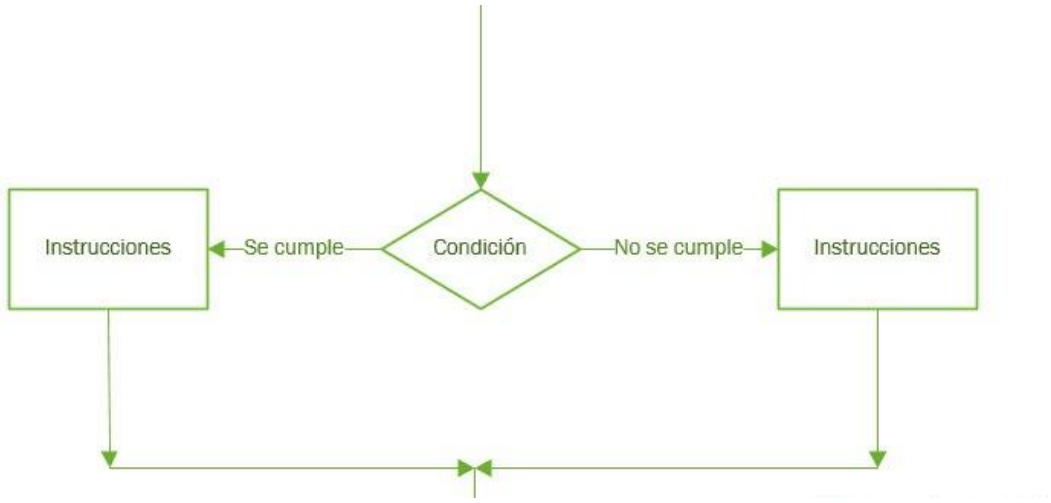
En muchas ocasiones necesitarás que se ejecuten una o varias instrucciones sólo si se cumple una condición. Si la condición no se cumple, el flujo del código continúa normalmente. Este es el contexto perfecto para utilizar la instrucción **If – Then**. Su sintaxis es la siguiente:

```
If condición Then
Instrucciones
End If
```

- **Condición:** Suele ser una comparación, pero puede ser cualquier expresión numérica que dé como resultado un valor de tipo numérico. VBA interpreta este valor como **Verdadero** o **Falso**. Cualquier número diferente a 0 lo considera Verdadero. Los valores 0 o null, los considera Falso.
- **Instrucciones:** El conjunto de órdenes a realizar.
  Cuando VBA encuentra una instrucción **If – Then**, evalúa si la expresión lógica es Verdadera (**True**) o Falsa (**False**). En caso afirmativo, ejecuta las instrucciones que se encuentran detrás de **Then**. En caso contrario, no hace nada y continúa ejecutando las instrucciones encuentra en la siguiente línea detrás de **End If**.

La estructura If – Then – Else

La estructura anterior es un poco limitada. Sólo ejecuta código cuando se cumple la condición.



Normalmente necesitarás ejecutar unas instrucciones si se cumple la condición y otras si no se cumple. Para hacer esto, VBA cuenta con la estructura If – Then – Else. Observa la sintaxis:

```
If condición Then
Unas instrucciones
```

```
Else
Otras instrucciones
End If
```

Esta estructura permite que el flujo de ejecución se bifurque por dos caminos distintos según si la condición se cumple o no. Si la condición es **True**, se ejecutan una serie de instrucciones. Si es **False**, se ejecutan otras distintas desde la instrucción **Else** hasta **End If**.

La estructura If – Then – ElseIf

Esta estructura es una variación de **If – Then – Else**, que se utiliza cuando es necesario evaluar más de una condición. Observa la sintaxis:

```
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
```

En este tipo de estructura, se examina en primer lugar la **condición_1**. Si se evalúa como **True**, se ejecuta el bloque de instrucciones correspondiente. Si por el contrario, la condición es falsa, se evalúa la **condición_2**. Si es verdadera, se ejecuta el bloque siguiente, y así sucesivamente. Si ninguna de las condiciones de cumple, se ejecutará el código que encuentra entre **Else** y **End If**.

Estructuras If – Then anidadas

Como has visto, es posible introducir una instrucción **If – Then** dentro de otra. A esto se le llama “anidación” y se da frecuentemente cuando necesitas construir decisiones con alternativas, cuando unas condiciones se basan en otras.

Select Case

Cuando tienes muchas condiciones para evaluar, puede resultar complicado utilizar instrucciones **If** anidadas. Por eso VBA te ofrece la estructura **Select Case** como alternativa.
Con esta estructura se evalúa una condición que puede dar como resultado un número indeterminado de valores, y realizar acciones según este resultado. Observa la sintaxis:

```
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
```

Select Case evalúa una única condición al principio de la estructura. A continuación compara el resultado con los valores de cada **Case** de la estructura y, si coincide, ejecuta las instrucciones asociadas a ese **Case**.

La estructura With – End With

La mayoría de los objetos cuentan con muchas propiedades. A veces es necesario modificar varias propiedades sobre un mismo objeto, por lo que se debería repetir el nombre del objeto tantas veces como propiedades se vayan a modificar.

Para evitar repetir muchas veces el mismo objeto se utiliza la estructura **With – End With**. El objeto se nombra una sola vez pero se pueden modificar las propiedades que sean necesarias.. Su sintaxis es la siguiente:

```
With objeto
[instrucciones]
End With
```

Observa el siguiente ejemplo. En él te muestro cómo poner una fila entera en negrita, con una fuente Comic Sans de color verde y con tamaño 14:

```
Sub ejemplo_with()
With Range("1:1").Font
.Bold = True
.Name = "Comic Sans MS"
.Size = 14
.ColorIndex = 4
End With
End Sub
```

Las estructuras de bucle

Gracias a este tipo de estructura es posible repetir uno o varios bloques de código, un número de veces determinado por una condición. Mientras se siga cumpliendo esta condición, el bloque se seguirá ejecutando.

Un ejemplo típico se da cuando un usuario intenta acceder a un archivo con su contraseña. Si la contraseña coincide con la que la aplicación tiene guardada en su base de datos, el bucle terminará, permitiéndole el acceso. En caso contrario, se le pedirá la contraseña una y otra vez hasta que introduzca el dato correcto.

Entre las estructuras de bucle se pueden encontrar:

- For – Next
- For Each – Next
- Do – Loop

For – Next

Se utiliza cuando es necesario **repetir un bloque de código un número determinado de veces** que ya se sabe de antemano. Esta estructura utiliza 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
```

- **Contador:** es la variable que se utiliza para contar las “pasadas” que hace el bucle y debe ser de tipo numérico. Esta variable aumenta o disminuye de forma constante.
- **Inicio:** es el valor numérico desde el que el bucle comienza a contar.
- **Final:** es el valor numérico hasta el que el contador debe llegar.
- **Incremento:** Opcional. Se trata de una expresión numérica positiva o negativa. Si es positiva, inicio deberá ser menor o igual que final. Si es negativa, inicio tendrá que ser mayor o igual que final. Si omites este parámetro, VBA incrementará la variable de 1 en 1.
- **Bloque de instrucciones**: son las instrucciones que se ejecutarán repetidamente.
- **Next**: es la palabra clave que indica a VBA dónde finaliza el bucle. Cuando el flujo de ejecución llega a esta línea regresa a la linea donde se encuentra For.

For Each – Next

Este tipo de bucle es exclusivo de Excel y te permite recorrer todos los elementos de una colección o matriz. Observa la sintaxis:

```
For Each elemento In grupo
[Instrucciones]
[Exit For]
[Instrucciones]
Next [elemento]
```

- **Elemento:** es la variable que se utiliza para recorrer la colección o la matriz. Si es una colección, esta variable puede ser de objeto o tipo Variant. Si se trata de una matriz, sólo podrá ser Variant.
- **Grupo:** es el nombre del conjunto de objetos o la matriz.
- **Instrucciones:** representa el grupo de instrucciones para realizar.
- **Next:** indica el final del bucle.

Cómo salir de las estructuras For – Next y For Each – Next

Algunas veces no es necesario ejecutar todas las repeticiones del bucle que tenías previstas en un principio. Para esta operación existe una instrucción muy útil: **Exit For**.

En el siguiente ejemplo te muestro el uso práctico de Exit For. Se trata del ejemplo que he puesto más arriba sobre la introducción de la contraseña para acceder al archivo:

```
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
```

Las estructuras Do – Loop

Las estructuras **For – Next** son perfectas cuando se sabe de antemano el número de veces que se debe ejecutar un código. Pero ¿qué ocurre cuando no hay un número concreto?.

Los bucles **Do – Loop** cubren esta necesidad ya que permiten ejecutar un bloque de código de forma indefinida hasta que se cumple una condición. **Este tipo de bucle hace posible evaluar la condición al principio o al final de la estructura**. También puedes especificar si el bucle se repite mientras (**While**) la condición se siga cumpliendo o hasta (**Until**) que la condición se cumpla. Así que, según esto, existen dos tipos de bucle **Do – Loop**:

- **Do – Loop While**: Se ejecuta mientras la condición se cumpla.
- **Do – Loop Until**: Se ejecuta hasta que se cumpla la condición.

  En los dos tipos de bucle, primero se ejecutan las instrucciones y luego se evalúa la condición.

También es posible hacer que primero se evalúe la condición y, si se cumple, se ejecute el código mientras se cumpla o hasta que lo haga.

- **Do While – Loop**: Se ejecuta mientras la condición se cumpla.
- **Do Until – Loop**: Se ejecuta hasta que se cumple la condición.

  Ejemplos con el bucle **Do – Loop While** y **Until** (primero se hace la comprobación y luego se ejecutan las instrucciones).

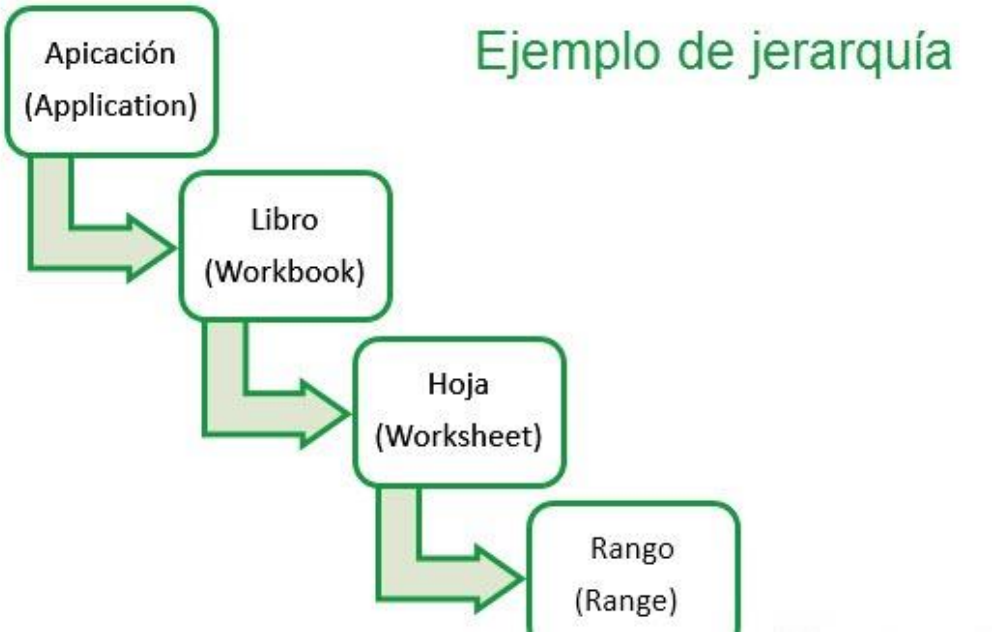
  Al igual que con los bucles **For**, también es posible salir antes de tiempo de un bucle **Do – Loop**. Para ello está la instrucción **Exit Do**.

  Cuando VBA encuentra la instrucción **Exit Do**, el flujo de ejecución del código pasa directamente a la siguiente instrucción tras el bucle.

Los objetos de Excel

Como te expliqué al principio de la guía, VBA es un lenguaje orientado a objetos. El modelo de objetos de Excel contiene más de 200 elementos ordenados de forma jerárquica, aunque hay cuatro de ellos que seguramente sean los que más utilices:

- **Application:** Hace referencia a la aplicación (Excel).
- **Workbook:** Hace referencia al libro de trabajo.
- **Worksheet:** Hace referencia a una hoja de cálculo.
- **Range:** Hace referencia a un rango de celdas.



Te los muestro con más detalle a continuación.

El objeto Application

Es el objeto con mayor rango en el modelo de objetos. **Hace referencia a la aplicación** (en este caso, Excel).
Brinda acceso a las opciones y configuraciones a nivel de aplicación, como por ejemplo a la cinta de opciones, a la impresión o al tamaño por la fuente por defecto, entre otras.
Hacer modificaciones en este nivel supone cambios que se van a reflejar en la interfaz de Excel y en sus diferentes herramientas.

Propiedades de Application

Application es uno de los objetos más importantes de VBA, por eso cuenta con muchas propiedades. Algunas de ellas definen el ambiente donde se ejecuta Excel, tras controlan la presentación de la interfaz y otras devuelven objetos.
Algunas propiedades de las más importantes son:

- **Caption:** Devuelve el nombre que aparece en la barra de titulo.Path: Devuelve la ruta de instalación de Microsoft Excel.
- **DefaultFilePath:** Determina el directorio por defecto donde guardar los libros de Excel.

- **StatusBar:** Sirve para mostrar mensajes en la barra de estado.
- **DisplayFormulaBar:** Muestra u oculta la barra de fórmulas.
- **DisplayAlerts:** Determina si se muestran o no los cuadros de diálogo de alerta.
- **DisplayFullScreen:** Determina si Excel se ejecuta en pantalla completa.

Métodos de Application

Como mencioné más arriba, los métodos son el conjunto de acciones que puede realizar un objeto. Te muestro alguno de ellos:

- **OnTime:** Permite programar los procedimientos para que se ejecuten en un momento determinado.
- **OnKey:** Permite programar los procedimientos para que se ejecuten cuando se pulsa una tecla o un método abreviado.
- **Wait:** Realiza una pausa por un tiempo especificado.
- **Quit:** Permite salir de Excel.

Workbooks y Workbook

El objeto Workbooks se encuentra por debajo de Application. Workbook forma parte de la colección Workbooks. Lo devuelven las siguientes propiedades del objeto Application:

- Application.Workbooks: Devuelve la colección Workbooks.
- Application.ActiveWorkbook: Devuelve el libro activo.
- Application.Thisworkbook: Devuelve el libro que contiene la macro ejecutada.

Propiedades de Workbooks y Workbook

Debes tener en cuenta, a la hora de trabajar con objetos y colecciones, que muchas colecciones comparten propiedades con los elementos que representan. Dependiendo del objeto, los parámetros necesarios o disponibles podrían no ser los mismos. Este es el caso de Workbooks y Workbook:

- **Item:** Hace referencia a un elemento en concreto de la colección.
- **Name:** Devuelve el nombre de un libro.
- **FullName:** Devuelve el nombre y la ruta del libro.

- **Saved:** Se utiliza para saber si un libro ha sido guardado o no. Devuelve False si todavía no se ha guardado o tras el guardado se han seguido haciendo cambios. Devuelve True cuando la última acción que se ha realizado ha sido guardarlo.

Métodos de los objetos Workbooks y Workbook

Estos son los métodos más importantes de estos objetos:

- **Add:** Permite crear un nuevo libro.
- **SaveAs:** Equivale al comando Guardar como.
- **Save:** Guarda los cambios hechos en un libro desde la última vez que se guardó.
- **Close:** Cierra el libro.
- **Open:** Abre un libro. Cada vez que se abre un libro se agrega un nuevo elemento a la colección Workbooks.

Worksheets y Worksheet

Como ya sabes, un libro de Excel puede contener una o varias hojas de cálculo que se pueden insertar, eliminar o mover a cualquier lugar. También es posible realizar otras acciones como renombrarlas, seleccionarlas y editarlas en conjunto.

En VBA una hoja de trabajo recibe el nombre de Worksheet y la colección de todas las hojas del libro es Worksheets.

Propiedades de Worksheets y Worksheet

Estas son las propiedades más utilizadas:

- **Item:** Sirve para hacer referencia a un objeto de la colección. Puedes hacerlo a través de su número de índice o de su nombre.
- **Name:** Permite establecer el nombre de la hoja.
- **CodeName:** Devuelve el nombre interno de la hoja.
- **Visible:** Permite mostrar u ocultar una hoja de cálculo.
- **Count:** Es una propiedad de la colección Worksheets. Devuelve el número de hojas del libro.

- **UsedRange:** Para conocer el rango utilizado en la hoja.

Métodos de Worksheets y Worksheet

- **Add:** Agrega una hoja de cualquier tipo al libro.
- **Move:** Permite organizar las hojas dentro de un libro.
- **Copy:** Copia una hoja de cálculo en otra ubicación.
- **Delete:** Elimina una hoja.

El objeto Range

El objeto Range identifica a una celda o conjunto de ellas, ya sean contiguas o no.

Propiedades de Range

Este objeto es de los que más propiedades tiene. Te muestro las más importantes:

- **ActiveCell:** Devuelve la celda activa de la ventana activa.
- **Range:** Devuelve un objeto Range que representa un rango de celdas.
- **Cells:** Otra forma de hacer referencia a las celdas.
- **Rows:** Devuelve un objeto Range que representa una fila de la hoja.
- **Columns:** Devuelve un objeto Range que representa una columna de la hoja.
- **Offset:** Devuelve un objeto Range desplazado de una referencia un número de filas y de columnas determinado.
- **Value:** Establece el valor que tiene una celda o rango de ellas.
- **FormulaLocal:** Permite introducir fórmulas y funciones en tu propio idioma, como si lo hicieses directamente en la hoja.
- **End:** Desplaza la celda activa a la última celda de la fila o la columna.
- **Font:** Establece el tipo de fuente.
- **Interior:** Permite modificar el fondo de la celda.
- **Border:** Permite modificar los bordes de la celda.

Métodos de Range

Estos son algunos de los métodos de Range más utilizados:

- **Select:** Permite seleccionar una celda o conjunto de ellas.
- **DataSeries:** Permite introducir una serie de datos en un rango de celdas.
- **Copy:** Copia un rango de celdas en otra parte de la cuadrícula.
- **ClearContents:** Borra el contenido de las celdas manteniendo su formato.

Los formularios

Gracias a los formularios es posible manejar datos en Excel de forma más intuitiva y eficiente, ya que, además de acelerar el proceso, consigue evitar los temidos errores de introducción de datos.

Los formularios están llenos de controles (botones, cuadros de texto, etiquetas, etc.) programables para responder a determinadas acciones (clics, selección de datos, etc.).

Para agregar un formulario a Excel desde el editor de VBA, primero debes crearlo a través del menú Insertar – UserForm. Además de crearse el nuevo formulario, en el explorador de proyectos también se crea la carpeta Formularios donde irán alojados todos los que crees para ese proyecto.

En la ventana también aparecerá un cuadro flotante donde se muestran muchos controles para que comiences a agregarlos al formulario.

Un formulario es un objeto **UserForm**, que tiene sus propiedades, métodos y eventos propios con los que modificar su apariencia y comportamiento.

Propiedades de los formularios

El primer paso para diseñar un formulario consiste en establecer las propiedades iniciales. Si no aparece la ventana Propiedades en la parte izquierda, pulsa **F4** para mostrarla.
La ventana **Propiedades** está dividida verticalmente. En la parte izquierda se muestra el nombre de la propiedad y en la parte derecha, el valor correspondiente. Modificar una propiedad es muy fácil. Sólo debes introducir el nuevo valor en la parte derecha.
Estas son las propiedades más utilizadas:

- **Name:** Permite establecer el nombre del formulario.

- **Caption:** Permite personalizar la barra de título del formulario.
- **Height:** Establece la altura del formulario.
- **Width:** Establece el ancho del formulario.
- **BackColor:** Permite elegir el color de fondo.
- **BorderStyle:** Permite establecer un tipo de borde.
- **BorderColor:** Define el color del borde.
- **StartUpPosition:** Establece la ubicación exacta del formulario en la pantalla.
- **Picture:** Permite seleccionar una imagen de fondo para el formulario.
- **PictureAlignment:** Permite alinear la imagen dentro del formulario.
- **PictureSizeMode:** Permite ajustar el tamaño de la imagen.

Métodos de los formularios

Además de modificar las características también puedes cambiar los comportamientos:

Estos son los métodos más utilizados:

- **Show:** Muestra el formulario en pantalla. Si el formulario no está cargado en la memoria, primero lo carga y luego lo muestra.
- **Load:** Carga un formulario en la pantalla pero no lo hace visible.
- **Hide:** Oculta el formulario sin descargarlo de la memoria.
- **Unload:** Descarga el formulario de la memoria.

Eventos de los formularios

Los formularios también responden a eventos. Estos son los más importantes:

- **Initialize:** Se produce cuando se carga por primera vez el formulario con Show o Load.
- **Activate:** Se produce cuando el formulario recibe el foco y se convierte en activo.

Controles de los formularios

La gran mayoría de formularios que crees, servirán para introducir y seleccionar datos más fácilmente. Para ello necesitas insertar controles.

Excel tiene muchos tipos de controles diferentes que puedes encontrar en el **Cuadro de herramientas** (la barra de herramientas flotante). Para insertar cualquiera de ellos sólo tienes que arrastrarlo hasta el formulario o hacer clic sobre él y dibujarlo en la ubicación que desees:

- **Etiqueta** (Label): Se utiliza para mostrar información que los usuario no puedan modificar y para identificar los demás controles.
- **Cuadro de texto** (TextBox): Se suele utilizar para mostrar información o para introducir datos . El contenido puede ser editado.
- **Cuadro de lista** (ListBox): Se muestra una lista de elementos para que el usuario seleccione uno o varios de ellos.
- **Cuadro combinado** (ComboBox): Es un cuadro de lista desplegable. El usuario puede seleccionar uno de los elementos que contiene.
- **Botón de comando** (CommandButton): La mayoría de formularios cuenta como mínimo con uno de estos botones. Se suele utilizar para realizar una acción.
- **Marco** (Frame): Se utiliza como contenedor de otros controles. Se suele utilizar para agrupar varios grupos de botones de opción.
- **Casilla de verificación** (CheckBox): Se utiliza para que el usuario dé una respuesta del tipo booleano (Sí/No, Verdadero/Falso).
- **Botón de opción** (OptionButton): Se emplea cuando el usuario debe seleccionar una sola opción de un grupo de opciones.
- **Imagen** (Image): Inserta una imagen en el formulario.
- **Página múltiple** (MultiPage): Este control contiene diferentes ficha, cada una de las cuales es una nueva página que puede albergar otros controles.
- **Barra de desplazamiento** (ScrollBar): Permite desplazamientos rápidos a lo largo de una lista de elementos.
- **Botón de número** (SpinButton): Permite al usuario seleccionar un número haciendo clic en una de las dos flechas que lo componen.

Anexos

Anexos

Tablas de Colores

Para obtener mas consejos acerca de la utilización de colores en VBA EXCEL visita el siguiente link

https://docs.microsoft.com/en-us/office/vba/api/excel.colorindex

Application.InputBox (Metodo)

Para obtener mas informacion del metodo Application.InputBox visita el siguiente link

https://docs.microsoft.com/en-us/office/vba/api/excel.application.inputbox

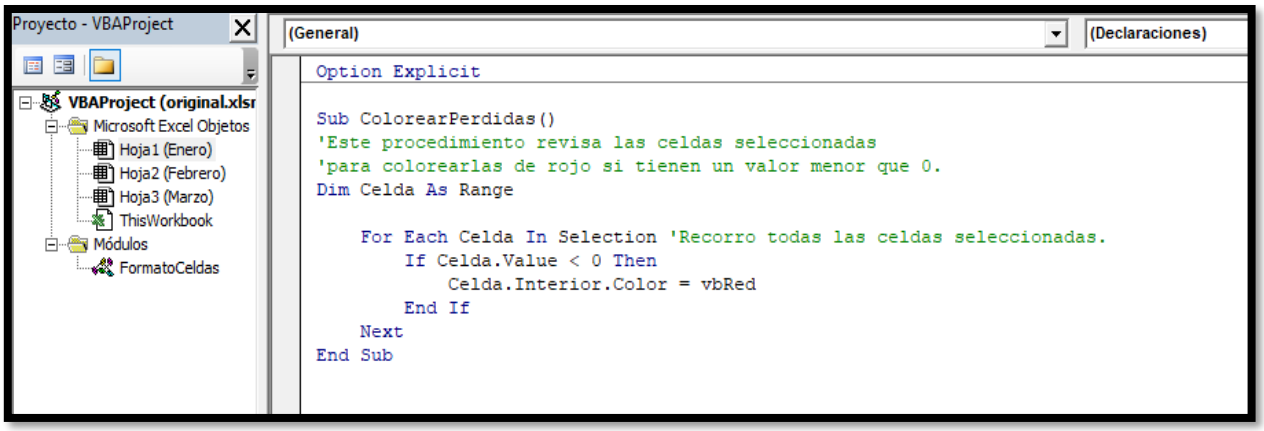
Modificadores de acceso Public and Private

Para obtener más información acerca de los modificadores de acceso public and private visita el siguiente link

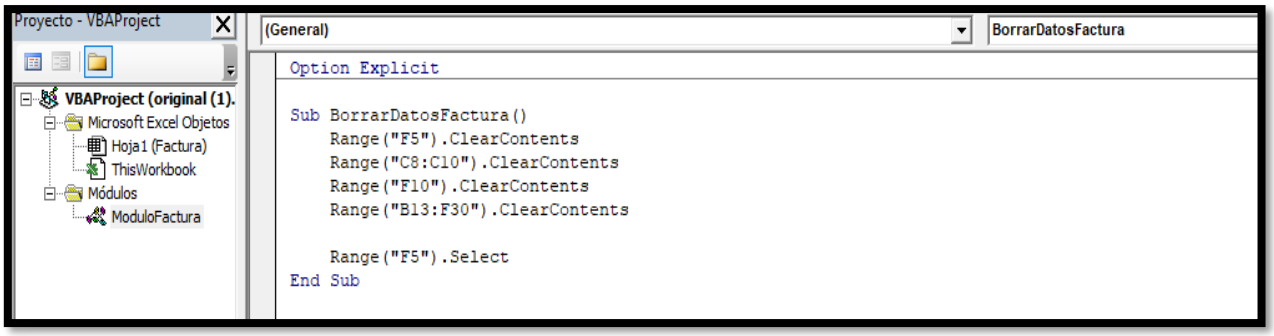
https://es.quora.com/Cu%C3%A1l-es-la-diferencia-entre-Public-y-Private-en-VBA

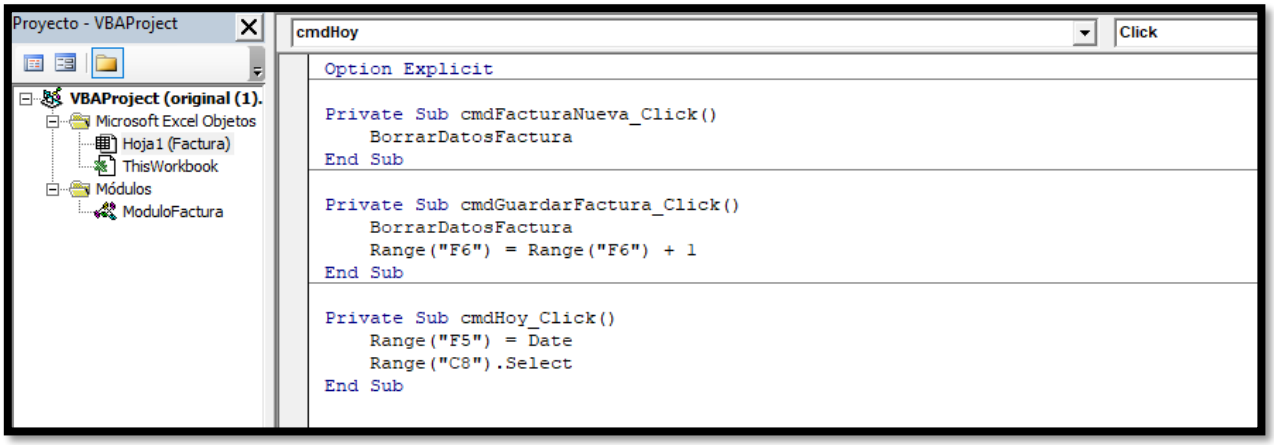
Anexo de programas

Código 1

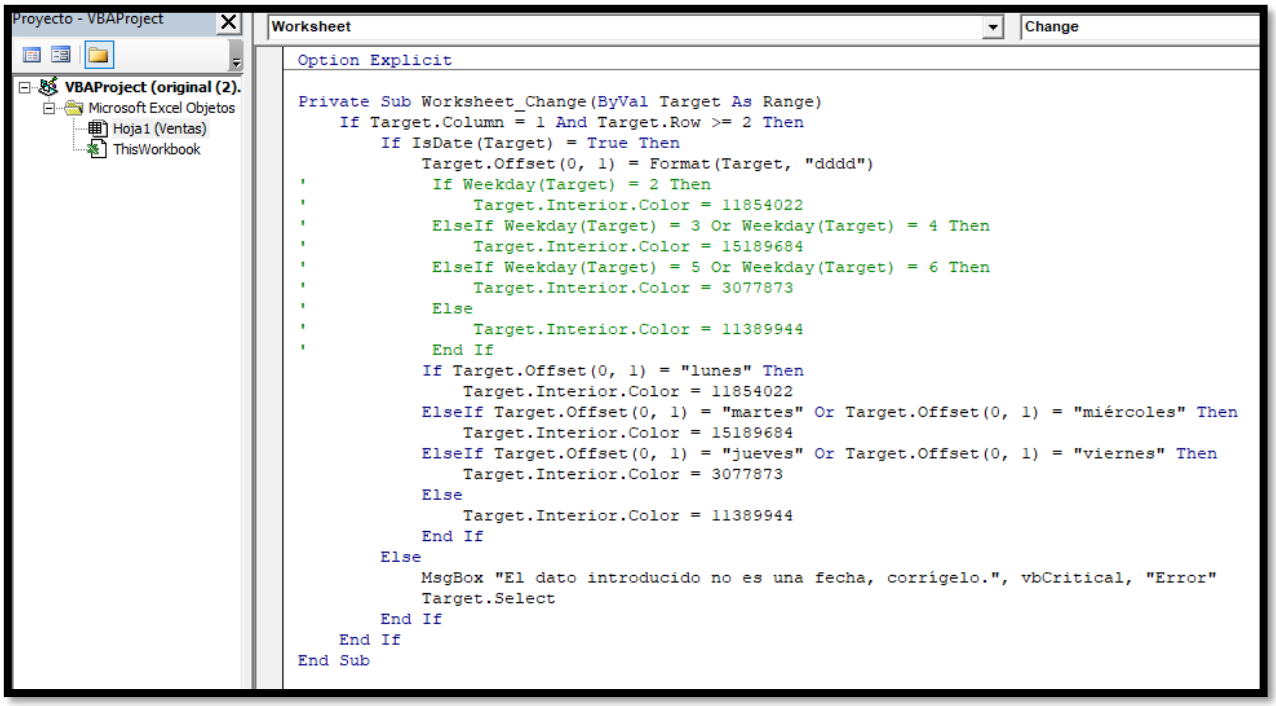


Código 2

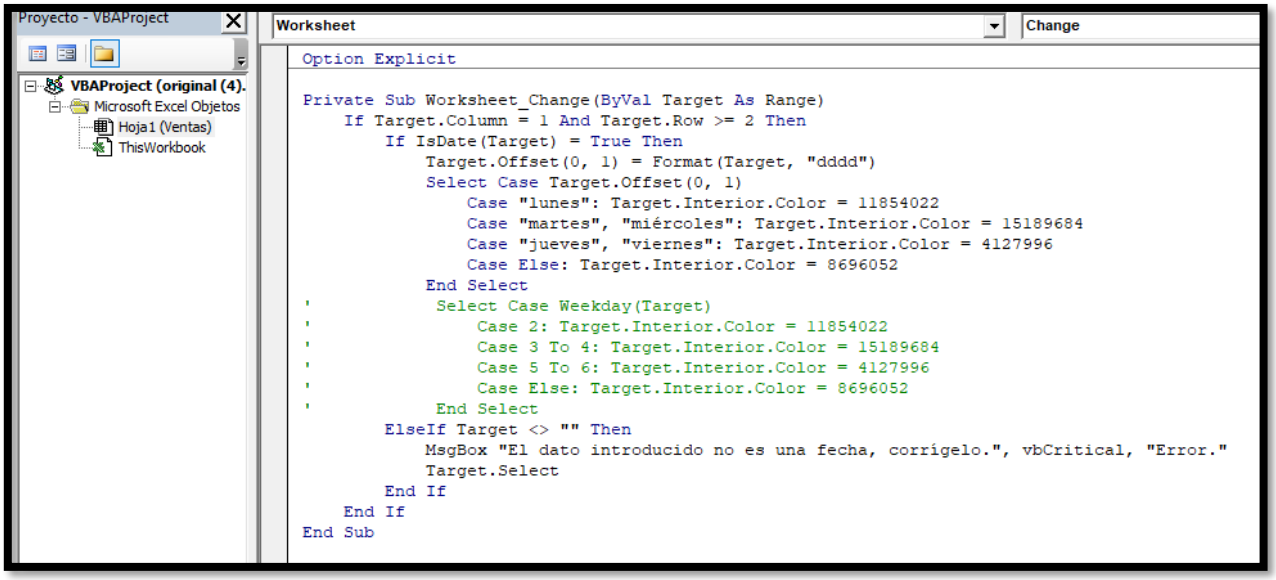




Código 3

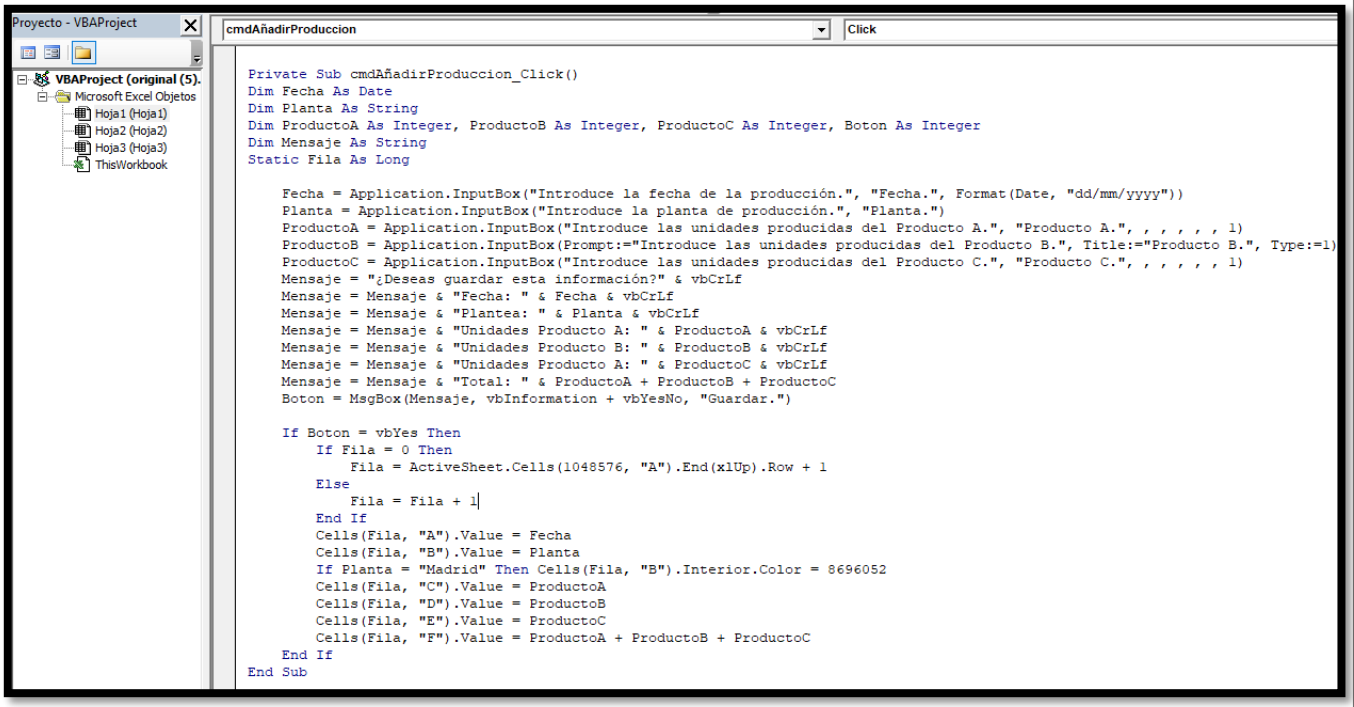



```
Option Explicit

Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Column = 1 And Target.Row >= 2 Then
        If IsDate(Target) = True Then
            Target.Offset(0, 1) = Format(Target, "dddd")
'            If Weekday(Target) = 2 Then
'                Target.Interior.Color = 11854022
'            ElseIf Weekday(Target) = 3 Or Weekday(Target) = 4 Then
'                Target.Interior.Color = 15189684
'            ElseIf Weekday(Target) = 5 Or Weekday(Target) = 6 Then
'                Target.Interior.Color = 3077873
'            Else
'                Target.Interior.Color = 11389944
'            End If
            If Target.Offset(0, 1) = "lunes" Then
                Target.Interior.Color = 11854022
            ElseIf Target.Offset(0, 1) = "martes" Or Target.Offset(0, 1) = "miércoles" Then
                Target.Interior.Color = 15189684
            ElseIf Target.Offset(0, 1) = "jueves" Or Target.Offset(0, 1) = "viernes" Then
                Target.Interior.Color = 3077873
            Else
                Target.Interior.Color = 11389944
            End If
        Else
            MsgBox "El dato introducido no es una fecha, corrígelo.", vbCritical, "Error"
            Target.Select
        End If
    End If
End Sub
```



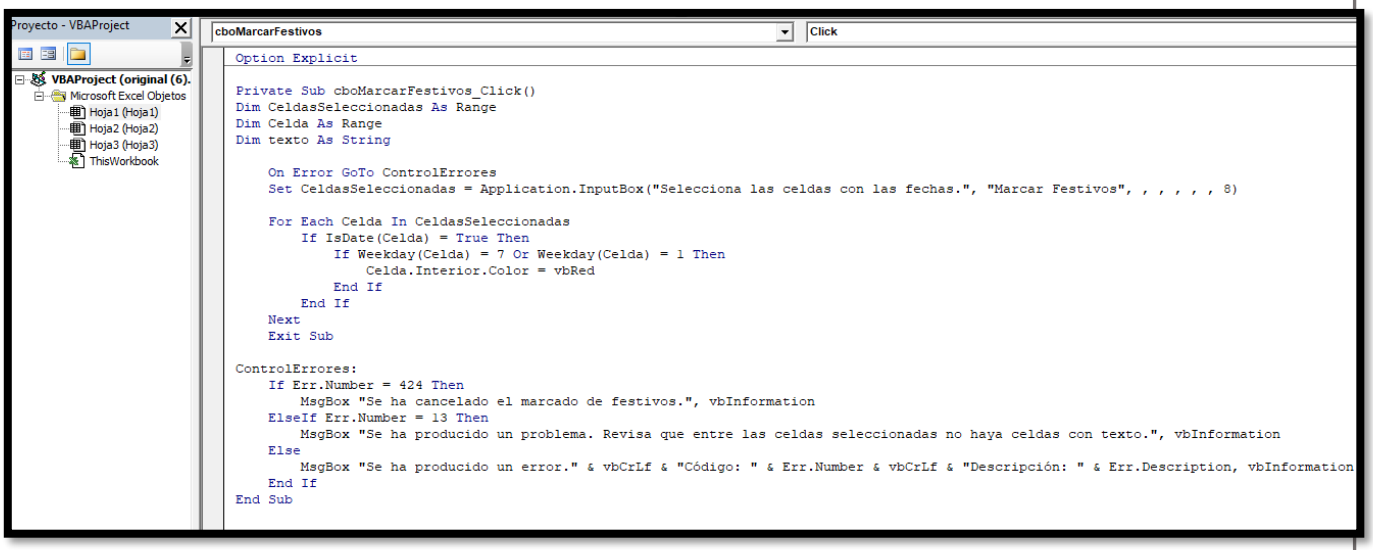

```
Option Explicit

Private Sub Worksheet_Change(ByVal Target As Range)
    If Target.Column = 1 And Target.Row >= 2 Then
        If IsDate(Target) = True Then
            Target.Offset(0, 1) = Format(Target, "dddd")
            Select Case Target.Offset(0, 1)
                Case "lunes": Target.Interior.Color = 11854022
                Case "martes", "miércoles": Target.Interior.Color = 15189684
                Case "jueves", "viernes": Target.Interior.Color = 4127996
                Case Else: Target.Interior.Color = 8696052
            End Select
'            Select Case Weekday(Target)
'                Case 2: Target.Interior.Color = 11854022
'                Case 3 To 4: Target.Interior.Color = 15189684
'                Case 5 To 6: Target.Interior.Color = 4127996
'                Case Else: Target.Interior.Color = 8696052
'            End Select
        ElseIf Target <> "" Then
            MsgBox "El dato introducido no es una fecha, corrígelo.", vbCritical, "Error."
            Target.Select
        End If
    End If
End Sub
```

Código 4



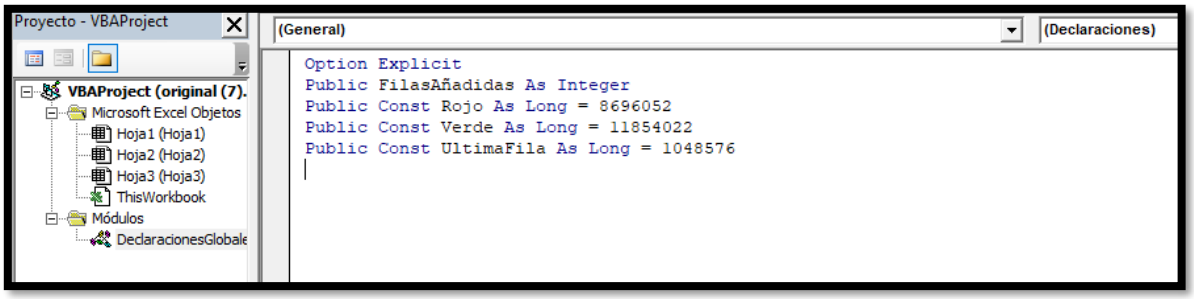

```
Private Sub cmdAñadirProduccion_Click()
Dim Fecha As Date
Dim Planta As String
Dim ProductoA As Integer, ProductoB As Integer, ProductoC As Integer, Boton As Integer
Dim Mensaje As String
Static Fila As Long

    Fecha = Application.InputBox("Introduce la fecha de la producción.", "Fecha.", Format(Date, "dd/mm/yyyy"))
    Planta = Application.InputBox("Introduce la planta de producción.", "Planta.")
    ProductoA = Application.InputBox("Introduce las unidades producidas del Producto A.", "Producto A.", , , , , , 1)
    ProductoB = Application.InputBox(Prompt:="Introduce las unidades producidas del Producto B.", Title:="Producto B.", Type:=1)
    ProductoC = Application.InputBox("Introduce las unidades producidas del Producto C.", "Producto C.", , , , , , 1)
    Mensaje = "¿Deseas guardar esta información?" & vbCrLf
    Mensaje = Mensaje & "Fecha: " & Fecha & vbCrLf
    Mensaje = Mensaje & "Plantea: " & Planta & vbCrLf
    Mensaje = Mensaje & "Unidades Producto A: " & ProductoA & vbCrLf
    Mensaje = Mensaje & "Unidades Producto B: " & ProductoB & vbCrLf
    Mensaje = Mensaje & "Unidades Producto A: " & ProductoC & vbCrLf
    Mensaje = Mensaje & "Total: " & ProductoA + ProductoB + ProductoC
    Boton = MsgBox(Mensaje, vbInformation + vbYesNo, "Guardar.")

    If Boton = vbYes Then
        If Fila = 0 Then
            Fila = ActiveSheet.Cells(1048576, "A").End(xlUp).Row + 1
        Else
            Fila = Fila + 1
        End If
        Cells(Fila, "A").Value = Fecha
        Cells(Fila, "B").Value = Planta
        If Planta = "Madrid" Then Cells(Fila, "B").Interior.Color = 8696052
        Cells(Fila, "C").Value = ProductoA
        Cells(Fila, "D").Value = ProductoB
        Cells(Fila, "E").Value = ProductoC
        Cells(Fila, "F").Value = ProductoA + ProductoB + ProductoC
    End If
End Sub
```

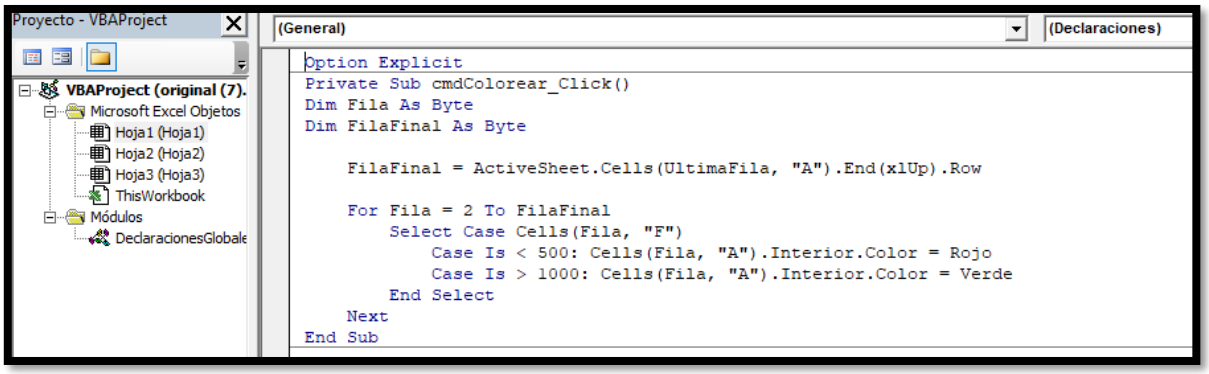
Código 5



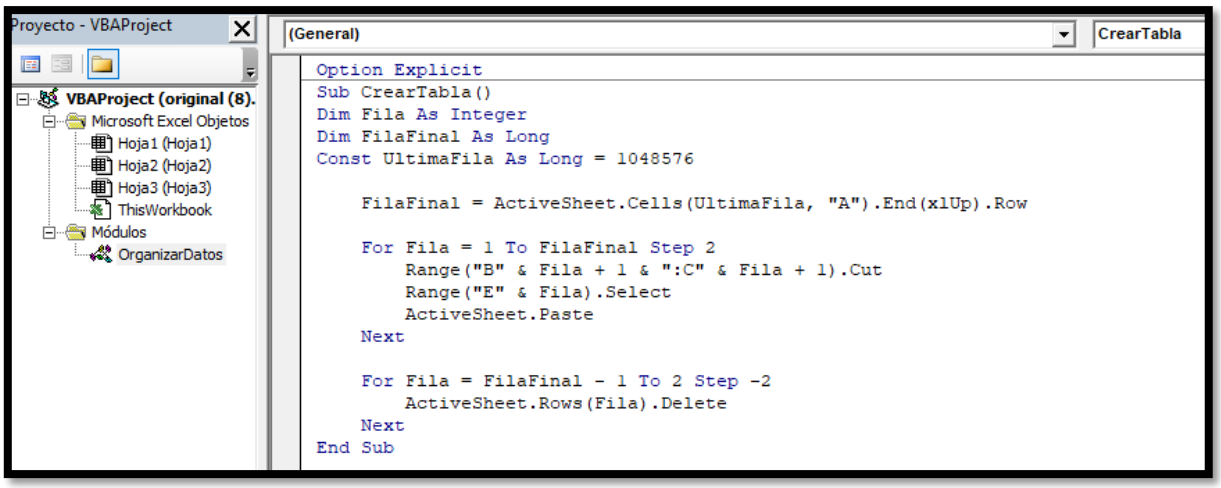

```
Option Explicit

Private Sub cboMarcarFestivos_Click()
Dim CeldasSeleccionadas As Range
Dim Celda As Range
Dim texto As String

    On Error GoTo ControlErrores
    Set CeldasSeleccionadas = Application.InputBox("Selecciona las celdas con las fechas.", "Marcar Festivos", , , , , , 8)

    For Each Celda In CeldasSeleccionadas
        If IsDate(Celda) = True Then
            If Weekday(Celda) = 7 Or Weekday(Celda) = 1 Then
                Celda.Interior.Color = vbRed
            End If
        End If
    Next
    Exit Sub

ControlErrores:
    If Err.Number = 424 Then
        MsgBox "Se ha cancelado el marcado de festivos.", vbInformation
    ElseIf Err.Number = 13 Then
        MsgBox "Se ha producido un problema. Revisa que entre las celdas seleccionadas no haya celdas con texto.", vbInformation
    Else
        MsgBox "Se ha producido un error." & vbCrLf & "Código: " & Err.Number & vbCrLf & "Descripción: " & Err.Description, vbInformation
    End If
End Sub
```

Código 6

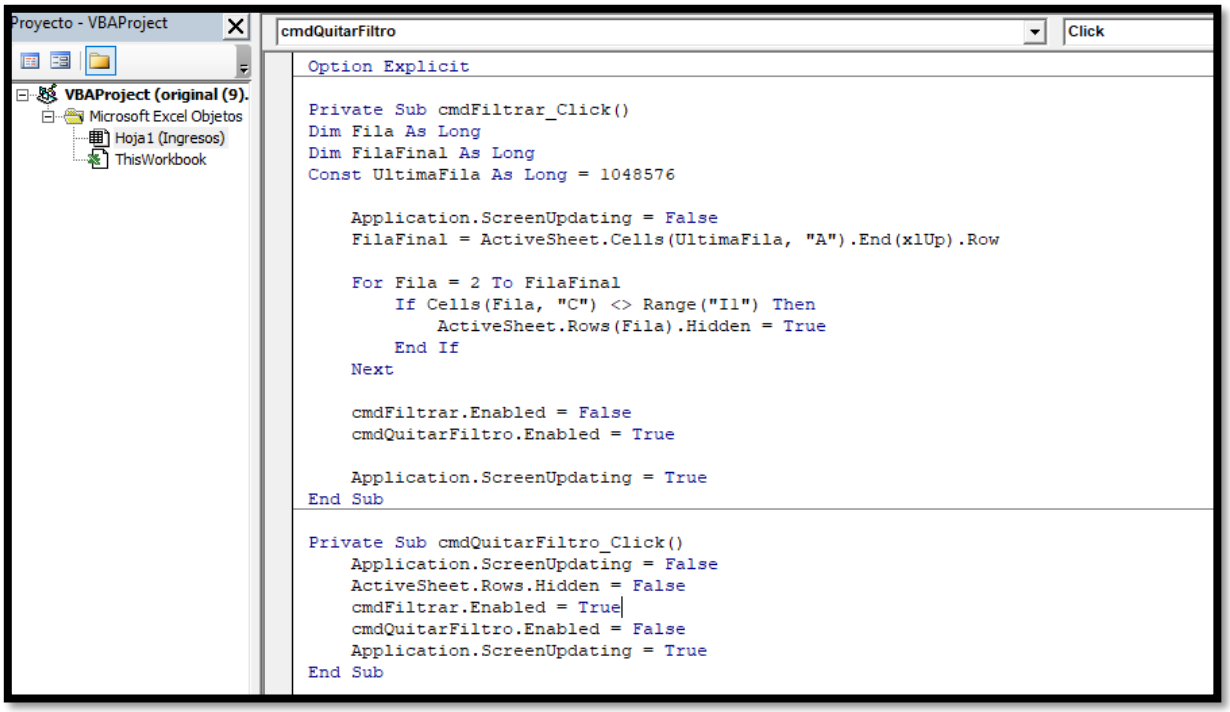




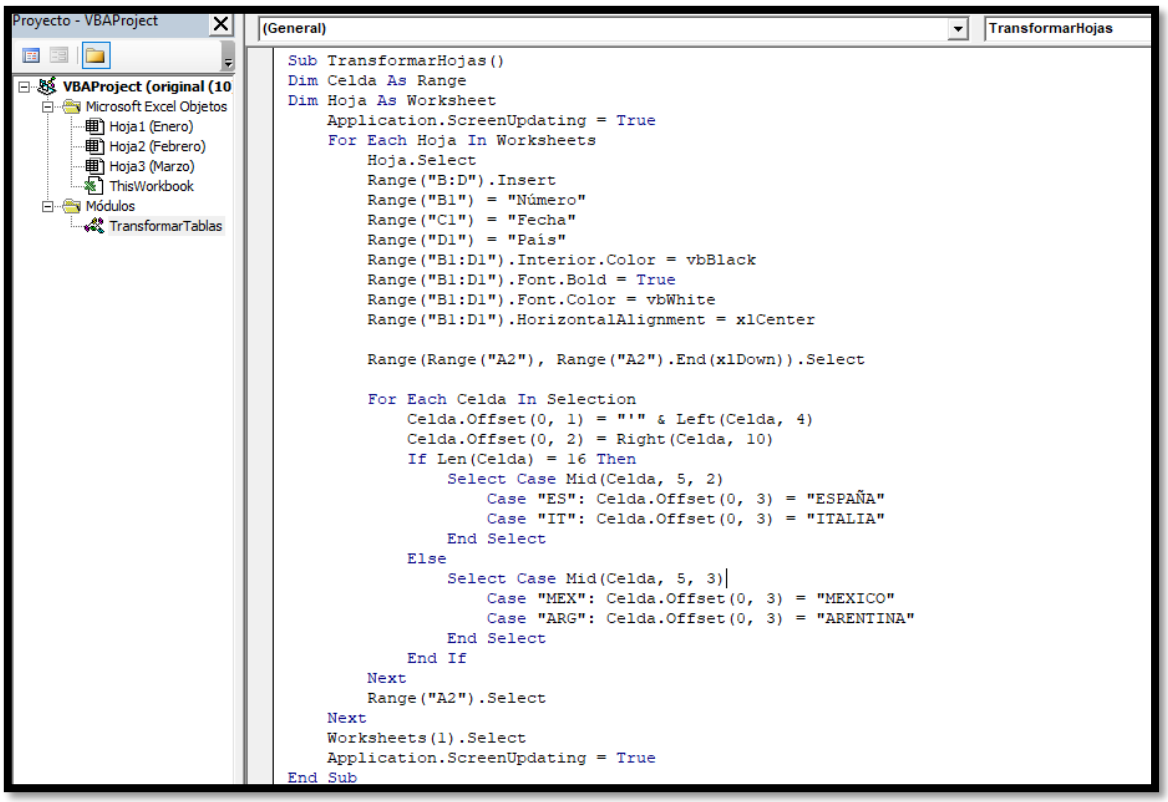
Código 7



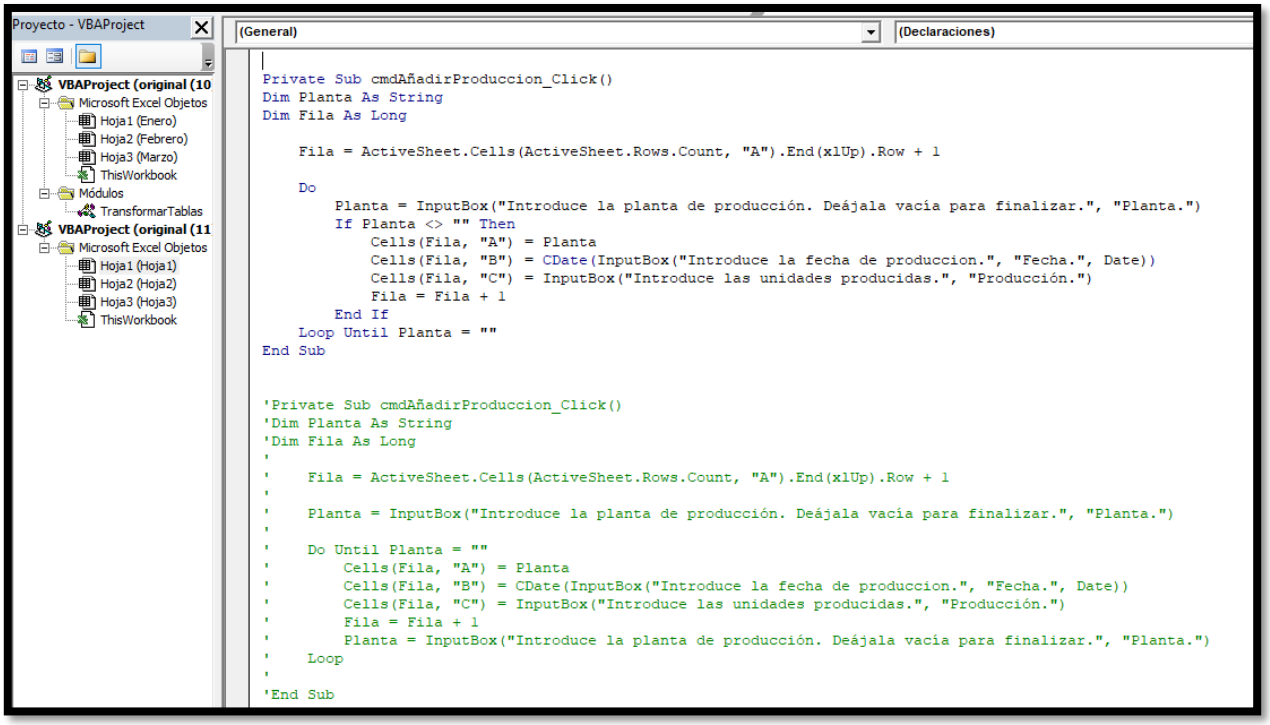
Código 8



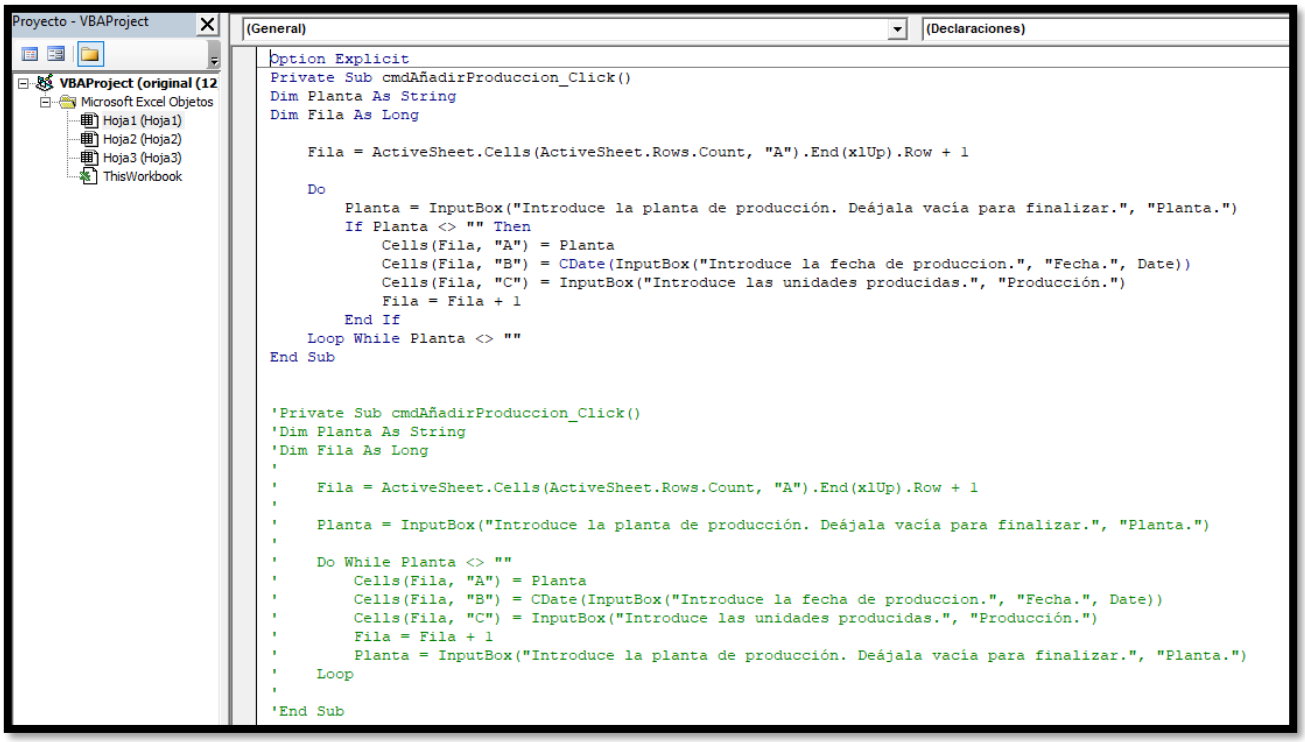
Código 9



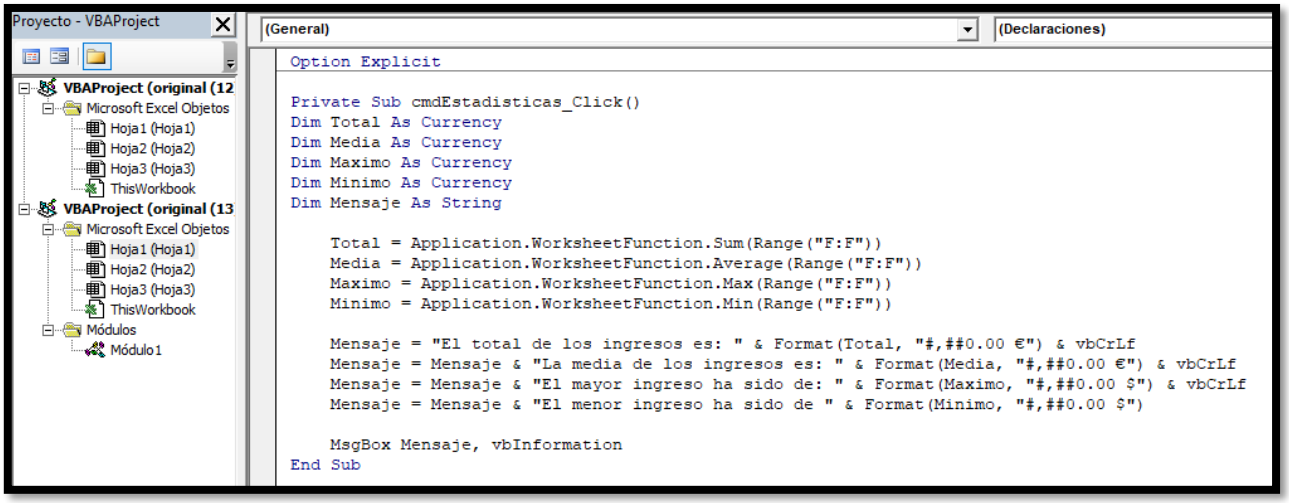
Código 10



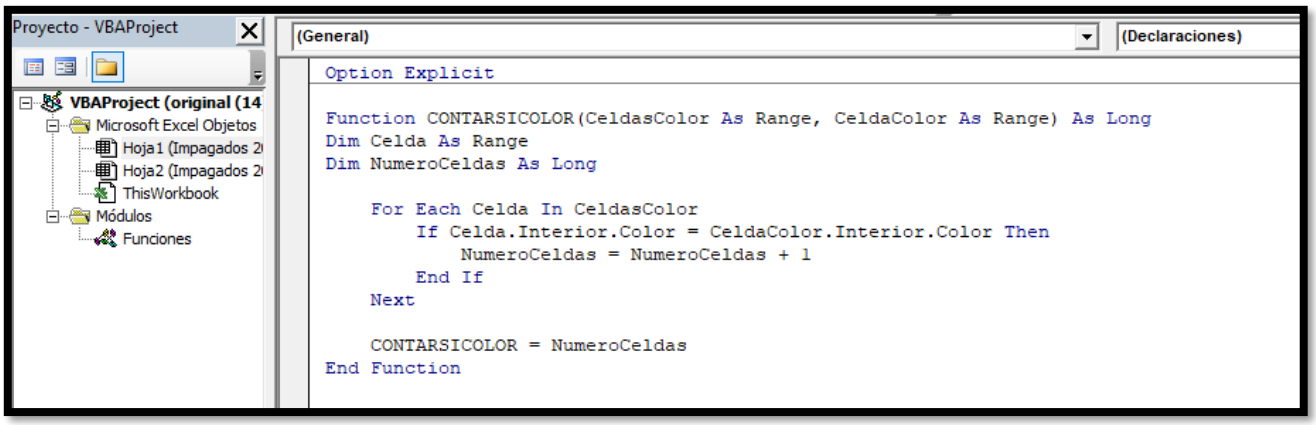
Código 11



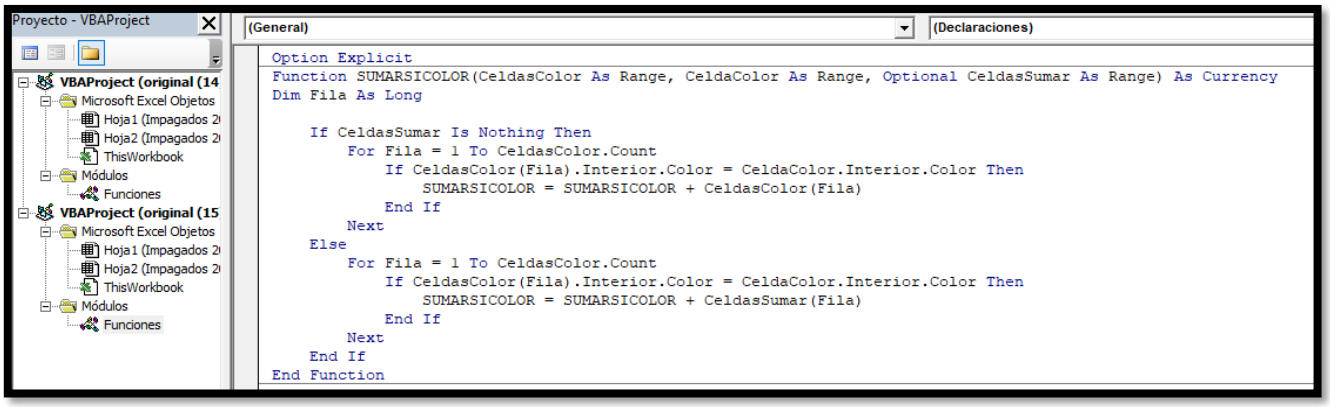
Código 12



Código 13


Proyecto - VBAProject
VBAProject (original (14
Microsoft Excel Objetos
Hoja1 (Impagados 2
Hoja2 (Impagados 2
ThisWorkbook
Módulos
Funciones
(General)
(Declaraciones)
Option Explicit
Function CONTARSICOLOR(CeldasColor As Range, CeldaColor As Range) As Long
Dim Celda As Range
Dim NumeroCeldas As Long
For Each Celda In CeldasColor
If Celda.Interior.Color = CeldaColor.Interior.Color Then
NumeroCeldas = NumeroCeldas + 1
End If
Next
CONTARSICOLOR = NumeroCeldas
End Function

Código 14


Proyecto - VBAProject
VBAProject (original (14
Microsoft Excel Objetos
Hoja1 (Impagados 2
Hoja2 (Impagados 2
ThisWorkbook
Módulos
Funciones
VBAProject (original (15
Microsoft Excel Objetos
Hoja1 (Impagados 2
Hoja2 (Impagados 2
ThisWorkbook
Módulos
Funciones
(General)
(Declaraciones)
Option Explicit
Function SUMARSICOLOR(CeldasColor As Range, CeldaColor As Range, Optional CeldasSumar As Range) As Currency
Dim Fila As Long
If CeldasSumar Is Nothing Then
For Fila = 1 To CeldasColor.Count
If CeldasColor(Fila).Interior.Color = CeldaColor.Interior.Color Then
SUMARSICOLOR = SUMARSICOLOR + CeldasColor(Fila)
End If
Next
Else
For Fila = 1 To CeldasColor.Count
If CeldasColor(Fila).Interior.Color = CeldaColor.Interior.Color Then
SUMARSICOLOR = SUMARSICOLOR + CeldasSumar(Fila)
End If
Next
End If
End Function