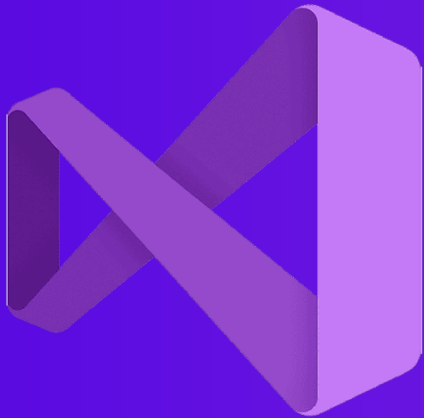


Visual Basic para Aplicaciones - VBA



Visual Basic

Septiembre 2023

Tabla de contenidos

1. Introducción a VBA
2. Acceso a los objetos de Excel
3. Fundamentos de programación
4. ¿Cómo programar?
5. Condicionales en VBA
6. Bucles VBA
7. Generación de complementos .XLAM
8. Interacción con el usuario
9. Eventos asociados a hojas y libro de Excel
10. Programación de tareas
11. Tratamiento de errores

Apéndice I: Librería VBA de funciones

1 | Introducción a VBA

¿Qué es VBA?

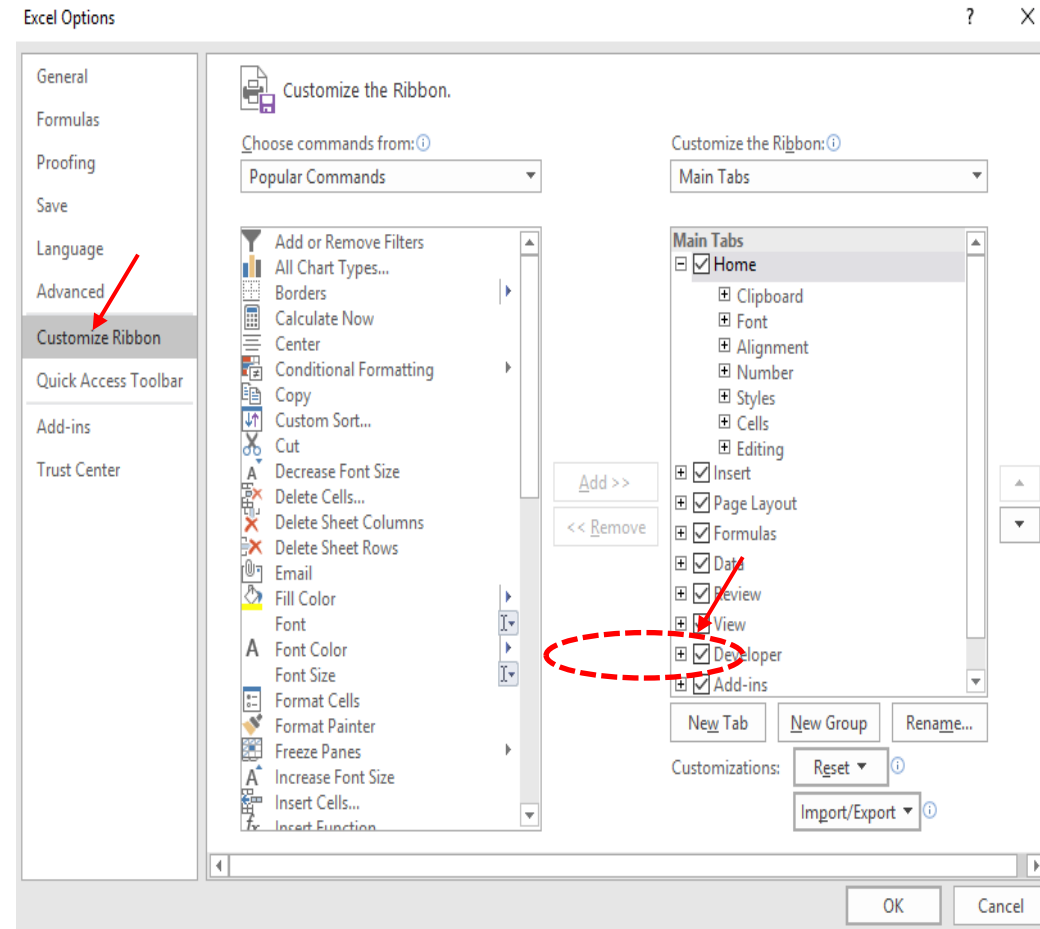
VBA = Visual Basic for Applications

- ✓ Lenguaje de programación macro basado en Microsoft Visual Basic.
- ✓ Útil en el desarrollo de aplicaciones insertadas en programas de Microsoft Office.
- ✓ Comúnmente usado en Microsoft Excel (Excel Visual Basic), aunque todos los programas de Microsoft Office ofrecen la posibilidad de generar código VBA.
- ✓ Ofrece la posibilidad de extender la funcionalidad de los programas.

¿Cómo acceder a VBA desde Excel?

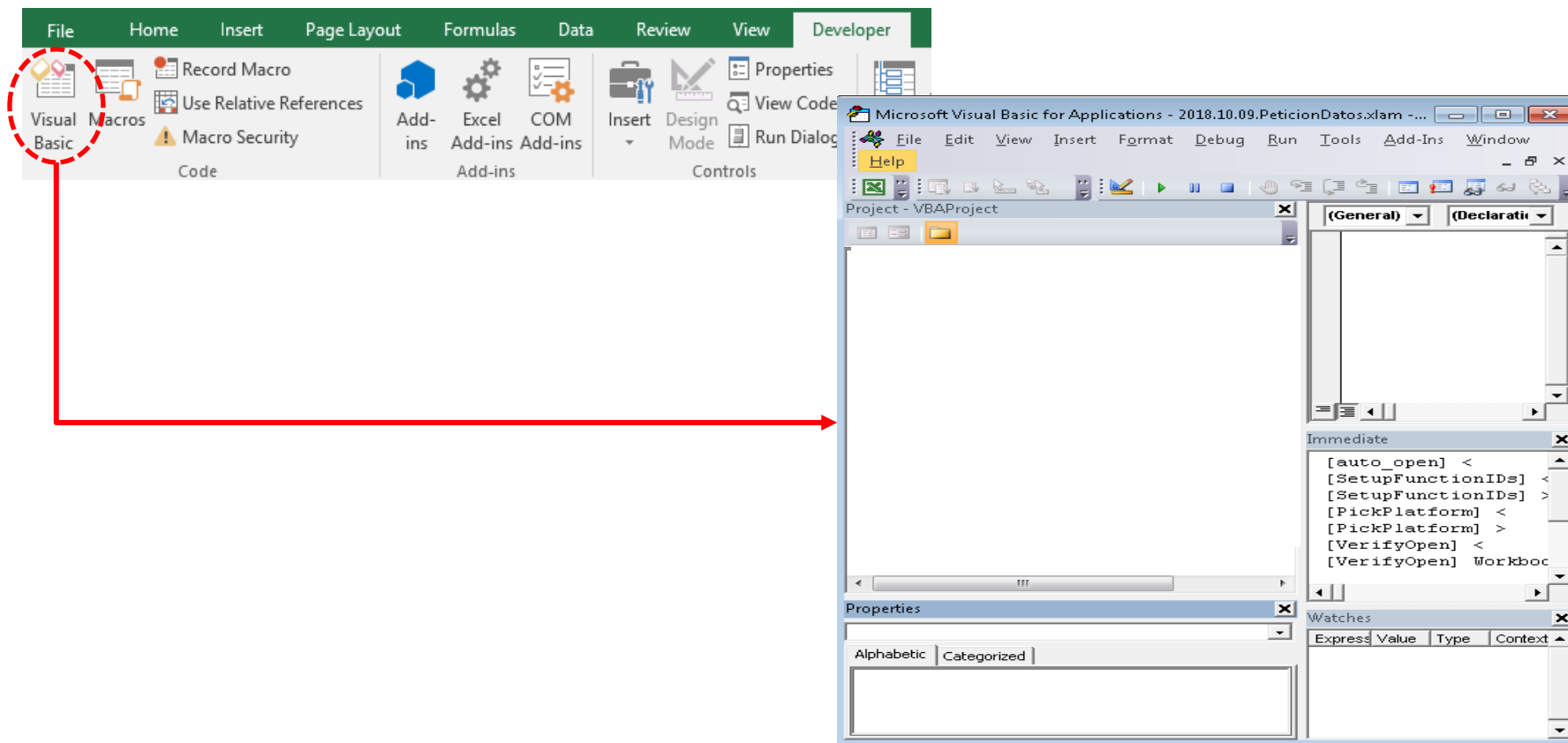
1. Mostrar la pestaña de 'Programador'.
No se exhibe por defecto, se puede añadir a la cinta de opciones.

Archivo => Opciones => Personalizar cinta de opciones => Seleccionar la casilla de Programador



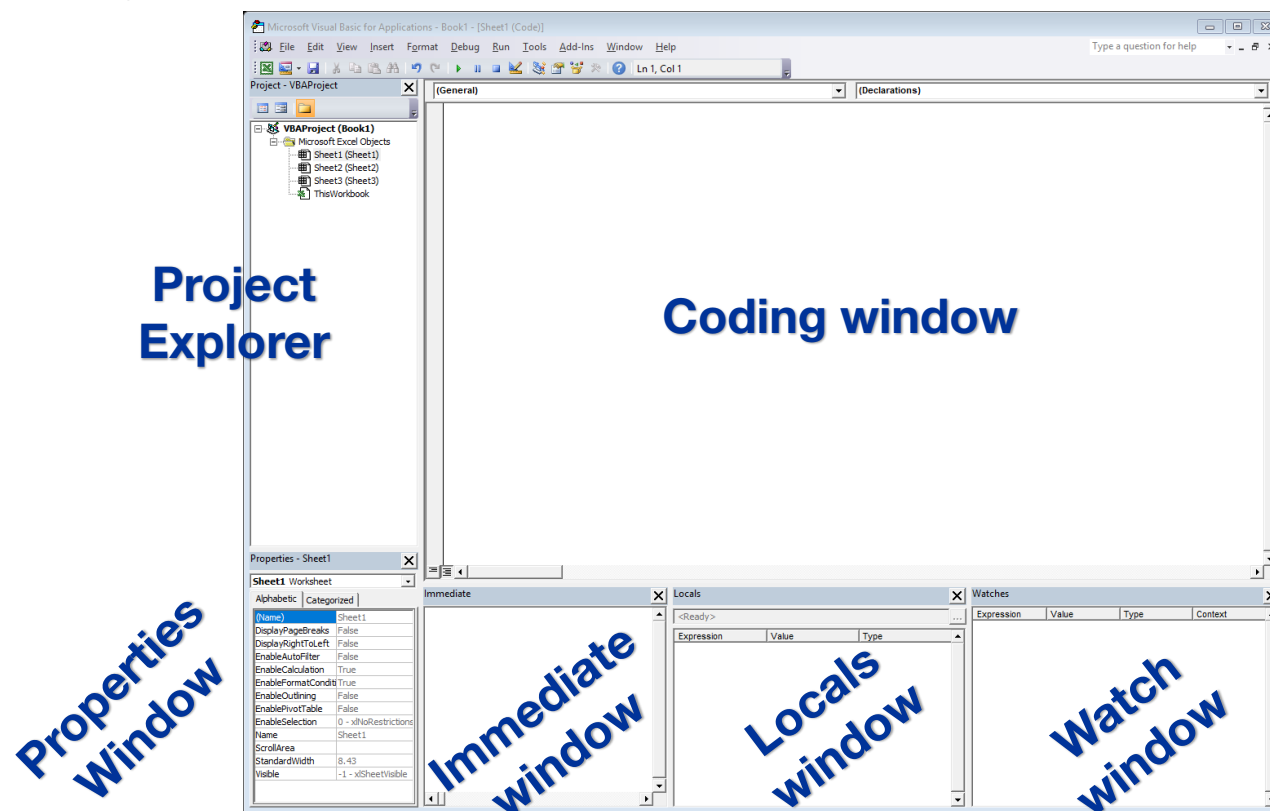
¿Cómo acceder a VBA desde Excel?

2. Programador => Visual Basic o Alt + F11



Ventanas de VBA

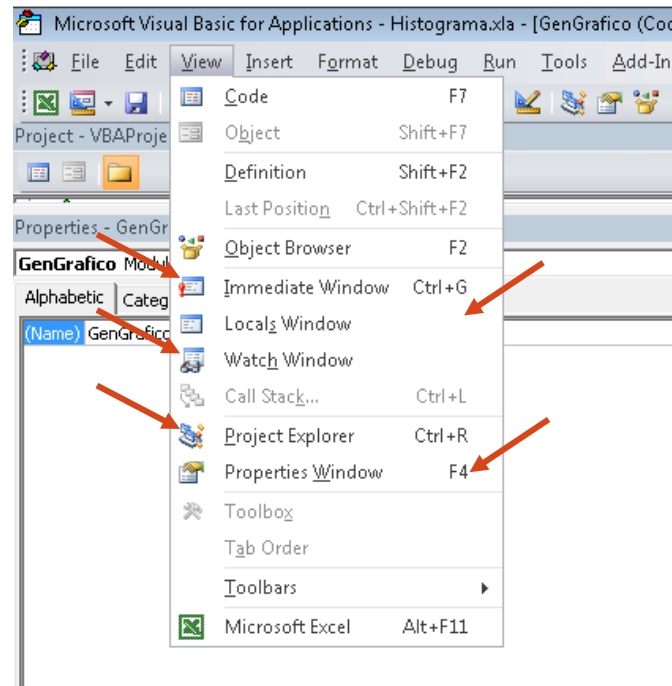
Normalmente, cuando abrimos el entorno de programación de VBA, nos encontramos una interfaz como esta de abajo (a veces, necesitaremos incluir o excluir algunas ventanas).



Configuración de ventanas

Para añadir ventanas:

Ver => (Nombre de la ventana)



Descripción de las ventanas



Explorador de proyectos

- Nos **muestra** los libros abiertos, incluyendo el libro actual, todas sus hojas y objetos creados (como formularios).
- Esta ventana nos permite **crear** y **abrir** módulos de código donde éstos serán escritos.



Código

- En esta ventana **escribiremos funciones y subrutinas**.



Ventana Inmediato

- Esta ventana nos permite **probar** y ejecutar **pequeños trozos de código**.
- Nos deja ejecutar cualquier declaración de VBA manualmente mientras el programa se está ejecutando y depurar el código.



Ventana Inspección

- Esta ventana nos permite **ver** el valor de **ciertas variables** o **expresiones** según se ejecuta el código.

Descripción de las ventanas



Ventana propiedades

- Nos enseña las propiedades del objeto que hayamos seleccionado (libro, pestaña, módulo,...)
- Además, nos da la posibilidad de modificar alguna propiedad sin necesidad de código



Locales

- Esta ventana muestra los valores auxiliares del código mientras está ejecutando
- Útil para ejecutar el código línea a línea

2 | Acceso a los objetos de Excel



Usar el comando “Print” en la ventana de Inmediato para probar

- ❖ **Celdas.** Con frecuencia se querrá acceder a variables contenidas en las celdas. Varias formas:

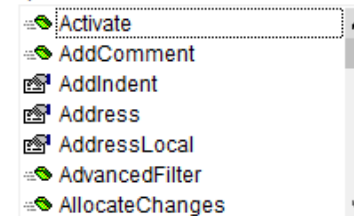
- ☐ **Range (“B1”)** => Hace referencia a la celda B1. Si la celda tiene un nombre definido por el usuario, podemos llamarla usando este comando.

(Ejemplo: **Range(“Celda1”)**) ← **Útil**

```
Sub prueba()
```

```
Range("B1").
```

```
End Sub
```



- ☐ **Cells(1,2)** => Hace referencia a la fila 1 y columna 2, o sea, la celda B1. (No aconsejable)



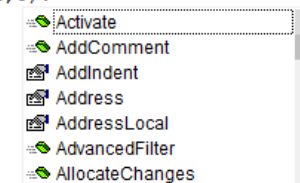
- ☐ **.Offset (NumeroFilas, NumeroColumnas):**

Range(“B1”).Offset(2,3) => Desde la celda B1, se mueve 2 filas hacia abajo y 3 columnas hacia la derecha. (Se pueden usar números negativos para movernos en dirección contraria).

```
Sub prueba()
```

```
Range("B1").Offset(2,3).
```

```
End Sub
```



❖ **Celdas (rangos)**. Con frecuencia se querrá acceder a variables contenidas en los rangos. Varias formas:

❑ **Range (“B1:E7”)** => Hace referencia al rango comprendido entre B1 y E7. Si el rango tiene un nombre definido por el usuario, podemos llamarlo usando este comando.

(Ejemplo: **Range(“Rango1”)**) ← **Útil**

```
Sub prueba()
```

```
Range("B1:E7").
```

```
End Sub
```



❑ **.Offset (NumeroFilas, NumeroColumnas)**

.Resize(LonFilas, LonColumnas):

Range(“B1”).Offset(2,3).Resize(3,4) => Desde la celda B1, se mueve 2 filas hacia abajo y 3 columnas hacia la derecha y accedemos a la matriz con dimensión 3 filas y 4 columnas (en este caso, el rango comprendido entre E3 y H5).

```
Sub prueba()
```

```
Range("B1").Offset(2, 3).Resize(3, 4).
```

```
End Sub
```



❖ **Celdas.** Podemos obtener propiedades del objeto celda (o rango), por ejemplo:

❑ **.Value:** Devuelve el valor que hay en la celda (rango).

	A
1	2

valor = Range("A1").Value

Expression	Value	Type
Module1 valor	2	Module1/Module1 Variant/Double

❑ **.Column:** Da la posición en columnas de la celda (equivalentemente con **.Row**).

```
fila = Range("E10").Row  
columna = Range("E10").Column
```

Expression	Value	Type
Module1 fila columna	10 5	Module1/Module1 Variant/Long Variant/Long



❑ **.Columns.Count:** Dentro del objeto **.Columns** (**.Rows**), podemos acceder al número de columnas de un rango. Muy útil.

```
filass = Range("E10:E15").Rows.Count  
columnass = Range("A10:E10").Columns.Count
```

Expression	Value	Type
Module1 filass columnass	6 5	Module1/Module1 Variant/Long Variant/Long

❖ **Sheets.** Hace referencia a las pestañas. Se pueden contar, añadir, cambiar el nombre, etc. Existen varias formas para acceder a ellas:

❑ **Índice:** Llama a la pestaña a través de su índice. Se puede usar “Worksheets” o “Sheets” .

```
Sub prueba()  
    Sheets(1).|  
  
End Sub
```

❑ **Nombre:** Llama a la hoja a través de su nombre. Se puede usar “Worksheets” o “Sheets”. También puede llamarse directamente con su nombre.

```
Sub prueba()  
    Worksheets("Informe").  
  
End Sub
```

❖ **Sheets.** Algunos métodos útiles:

- ❑ **.Count:** Cuenta cuántas pestañas tiene el libro seleccionado

```
Sub prueba()  
    n = Sheets.Count  
  
End Sub
```

- ❑ **.Add:** Añade una pestaña (o varias con el input *Count*) antes (*Before*) o después (*After*) de una de las pestañas del libro.

```
Sub prueba()  
    Sheets.Add Before:=Sheets(1)  
  
End Sub
```

- ❑ **.Select:** Selecciona la pestaña deseada

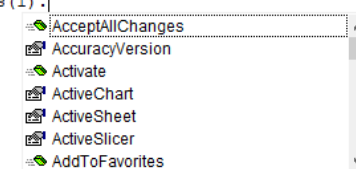
```
Sub prueba()  
    Sheets(2).Select  
  
End Sub
```


❖ **Workbooks.** Hace referencia a los libros. Se pueden abrir, cerrar, activar, etc. Existen varias formas para acceder a ellos:



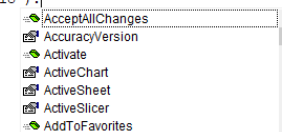
❑ **Índice:** Llama al libro a través de su índice. Necesario usar la clave “Workbooks”. No recomendable.

```
Sub prueba()  
    Workbooks(1).  
End Sub
```



❑ **Nombre:** Llama al libro por su nombre. Necesario usar la clave “Workbooks”. Recomendable.

```
Sub prueba()  
    Workbooks("LibroEjemplo").  
End Sub
```

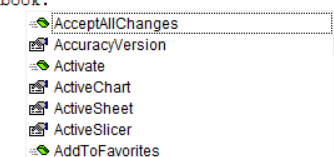


❑ **Path:** Nos devuelve la ruta completa del libro. La estructura ejemplo es “*Ruta/NombreLibro.xlsm*”.

```
Sub prueba()  
    Ruta = Workbooks("LibroEjemplo").Path  
End Sub
```

❑ **ThisWorkbook:** Nos permite acceder al libro donde se encuentra la macro que estamos ejecutando.

```
Sub prueba()  
    ThisWorkbook.  
End Sub
```



• Introducción a VBA

❖ **Workbooks.** Algunos métodos útiles:

❑ **.Open:** Abre un libro dada una ruta específica.

```
Sub prueba()  
  
    Workbooks.Open "D:/.../NombreLibro.xlsm"  
  
End Sub
```

❑ **.Close:** Cierra el libro seleccionado. Te ofrece la posibilidad de guardar los cambios o no antes de cerrar.

```
Sub prueba()  
  
    Workbooks("LibroPrueba").Close SaveChanges:=True  
  
End Sub
```

❑ **.Activate:** Activa el libro seleccionado.

```
Sub prueba()  
  
    ThisWorkbook.Activate  
  
End Sub
```

• Introducción a VBA

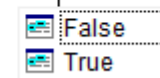
❖ **Application:** es la aplicación de Excel. Métodos útiles:

❑ **DisplayAlerts:** “False” para suprimir las solicitudes y los mensajes de alerta durante la ejecución de una macro. “True” por defecto.

```
Sub prueba()
```

```
Application.DisplayAlerts =
```

```
End Sub
```

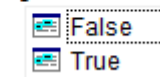


❑ **ScreenUpdating:** “False” para desactivar la actualización de pantallas. “True” por defecto.

```
Sub prueba()
```

```
Application.ScreenUpdating =
```

```
End Sub
```

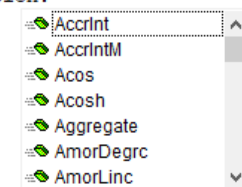


❑ **WorksheetFunction:** Se usa para llamar a las funciones de Excel desde Visual Basic.

```
Sub prueba()
```

```
Application.WorksheetFunction.
```

```
End Sub
```



❑ **StatusBar:** Modifica la barra de estado de la aplicación Excel (útil cuando tenemos un código por iteraciones y queremos saber, mientras se ejecuta, por qué iteración va).

```
Sub prueba()
```

```
Application.StatusBar = "Hola"
```

```
End Sub
```

3 | Fundamentos de programación

• Fundamentos de programación

Variables

Especificaremos el tipo de la variable con la palabra clave '**As**', normalmente al principio del proceso. Los principales tipos de datos son los siguientes:

Tipo	Descripción	Código
Integer	Números enteros (no decimales)	Dim x as Integer
Double	Números decimales con dos cifras de precisión	Dim x as Double
String	Cadenas de caracteres	Dim x as String
Boolean	Valor <i>Verdadero</i> o <i>Falso</i>	Dim x as Boolean
Variant	Cualquier tipo de dato (Fechas, matrices, decimales, cadenas...)	Dim x as Variant



• Fundamentos de programación

Variable Locales y Globales

- ❖ **Variables locales** se declaran **en** la rutina y solo pueden ser llamadas **desde** la rutina y **durante** su ejecución. Por ejemplo:

```
Sub Example1()  
    Dim LocalVariable as Integer  
    ....  
End Sub
```

- ❖ **Variables globales** se declaran **al principio** del módulo, antes de cualquier rutina. Por tanto, podrían ser usadas y modificadas por cualquier rutina del módulo. Por ejemplo:

```
Dim GlobalVariable as String  
Sub Example2()  
    ....  
End Sub
```



• Fundamentos de programación

Funciones definidas por el usuario y Subrutinas

En Excel Visual Basic, el conjunto de comandos para realizar una particular tarea o acción se establece dentro de un **procedimiento**, el cual puede ser una **Función** o una **Subrutina**.

FUNCIONES

Llevan a cabo cálculos y **devuelven** un valor único o una matriz.

SUBROUTINAS

Realizan una acción con **Excel**. Normalmente, no tienen parámetros de entrada o de salida.



En general, si deseas realizar una tarea que devuelva un resultado (e.g. sumar un grupo de números), usarás una **Función**, pero si solamente necesitas que se lleven a cabo un conjunto de acciones (e.g. dar formato a un conjunto de celdas), deberías decantarte por una **Subrutina**.

• Fundamentos de programación

Diferencias entre Funciones y Subrutinas

Función

- Suele llamarse desde una celda de Excel como una fórmula.
- Si se actualiza el Libro (F9), la función automáticamente se ejecutará y los resultados se actualizarán.
- Las funciones no pueden exportar resultados en otras celdas de Excel.

Subrutina

- Se llama independientemente de la celda de Excel en la que estemos (usualmente la asignaremos a un **Botón**).
- Se ejecuta siempre que el usuario quiera.
- Una vez la ejecución finaliza **no se pueden deshacer acciones tras ejecutar la macro**. Por ello, se recomienda guardar el libro antes de ejecutarla.

• Fundamentos de programación

Notas importantes

Es **importante** destacar que:

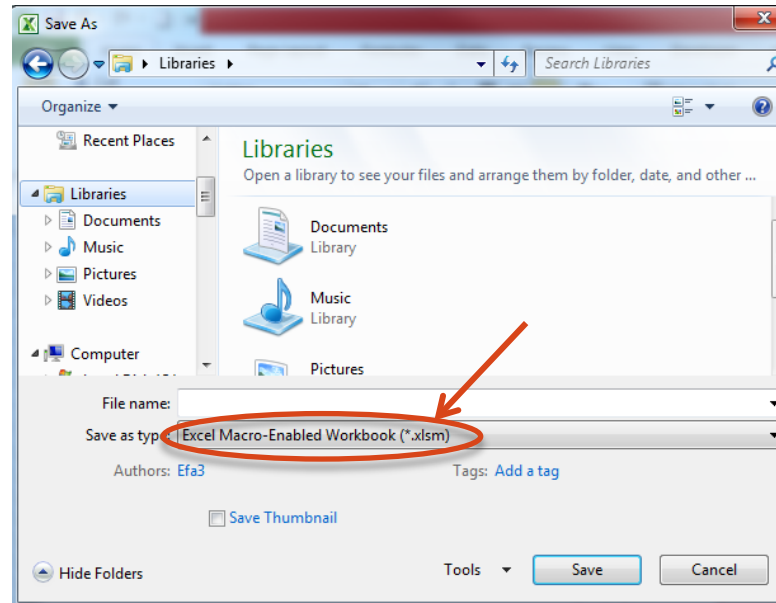
- ☐ VBA no distingue entre mayúsculas y minúsculas
- ☐ **NO** se permiten **espacios en blanco** en los nombres de las variables y los procedimientos. Un espacio en blanco separa dos nombres y el entorno solo reconoce el primero y genera un error en el segundo.
- ☐ Nombres que contengan **puntos** o **caracteres extraños** ('@', '#',...) **NO** están permitidos. Además, el **máximo** de caracteres permitido es 255.
- ☐ Aunque las tildes pueden ser utilizadas, su uso no se recomienda debido a que el lenguaje ha sido desarrollado por diseñadores de habla inglesa y, en determinadas circunstancias, puede producir errores extraños.
- ☐ **El código desarrollado por el usuario estará asociado con el libro actual, pero NO con la aplicación de Excel. En consecuencia, el código es compatible con ese libro y ejecutable desde cualquier otro libro siempre que el libro que contenga el código esté abierto.**

- Fundamentos de programación

Notas importantes



Es muy importante guardar el libro como '***Excel Macro-Enabled Workbook (*.xlm)***'. Si no hacemos esto, podríamos perder el código desarrollado en este libro.



• Fundamentos de programación

Programación estructurada

El código se escribe una vez y se lee muchas veces en el futuro. Por este motivo, se debería organizar de un modo claro y entendible. Algunas **recomendaciones**:

Comentarios

- Usar comentarios para explicar las líneas de código que no son fácilmente comprensibles.
 - VBA considera cualquier texto seguido de un apóstrofe (') como un comentario y no será ejecutado.
-

Sangría del código

- Usar sangría para fácilmente organizar el código en bloques individuales o secciones.
-

Líneas de parada

- Insertar líneas de parada en el medio de largas líneas de código para separar las filas. Para ello, añadir un espacio seguido de una barra baja (_) justo hasta el salto de línea.
-

Nombres auto-explicativos

- Usar nombres intuitivos y auto-explicativos para variables y procedimientos.
-

Longitud de procedimientos

- Crear procedimientos independientes para tareas específicas, acortando el código y haciéndolo más fácil de entender. Las Funciones y Subrutinas pueden ser llamadas desde otros procedimientos.
-

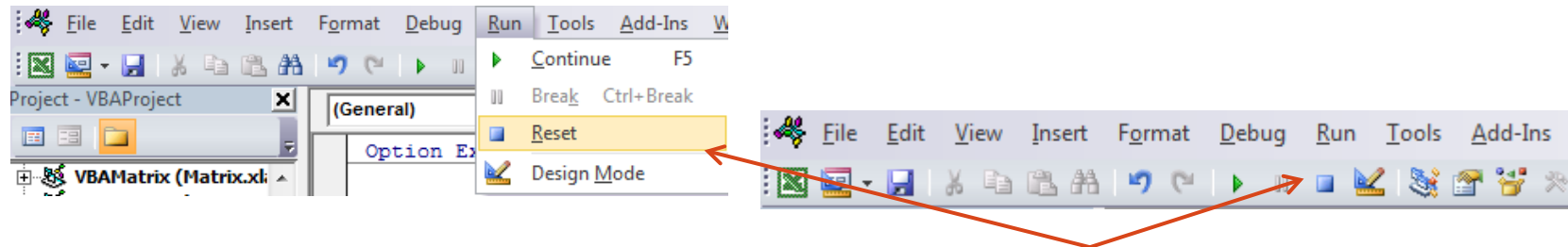
- Fundamentos de programación

Errores VBA

Si durante la ejecución del código el procesador detecta un error, por ejemplo, de sintaxis, el código dejará de funcionar y se mostrará un mensaje. En este caso, dará la opción de **depurar** el error. Se resaltará en amarillo la línea de código que genera el error.

```
⇒ Sub Deposit_Sub_1()  
  
End Sub
```

Cuando esto ocurra, aunque el error sea corregido, la línea seguirá resaltada (esto indica que el código está en *modo de parada*) y no podremos seguir ejecutando el código. Necesitamos pulsar **RESTABLECER** para detener este modo e intentar ejecutar el código otra vez.





4 |

¿Cómo programar?

• ¿Cómo programar?

Distintas posibilidades...

Excel Visual Basic ofrece dos opciones principales para programar:

1. Generación Automática de Macros

- Esta opción nos permite generar Subrutinas de Visual Basic que repliquen cualquier acción que se realice en Excel. Especialmente útil para tareas repetitivas que incluyan el uso de cualquier comando de Excel o secuencia de operaciones.
- No se requiere ningún conocimiento de programación.

2. Programación pura en Visual Basic

- Esta opción consiste en crear Funciones y Subrutinas de Visual Basic escribiendo el código manualmente (implementación de fórmulas, algoritmos matemáticos, etc...)
- Se requiere el conocimiento de la sintaxis de programación por el usuario e insertarla en el editor de Visual Basic.

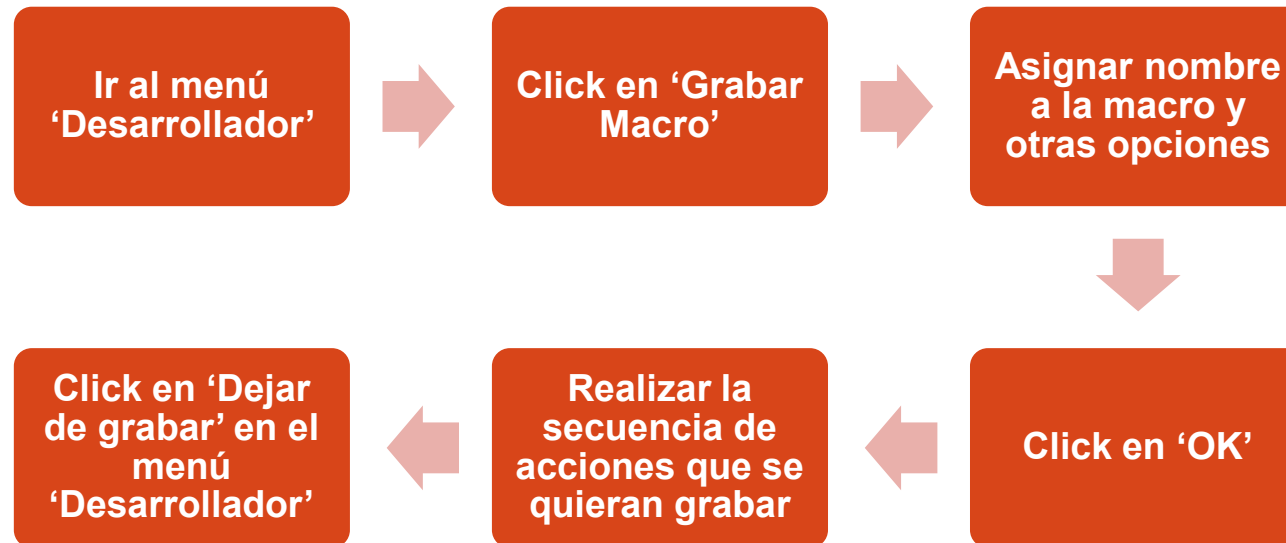


El modo más común de programar en Excel Visual Basic es mezclando macros automáticas y programación pura. Las Macros generadas automáticamente pueden ser modificadas y adaptadas a situaciones o necesidades específicas. Esto incrementa significativamente el poder del generador de macros.

- ¿Cómo programar?

Generación Automática de Macros

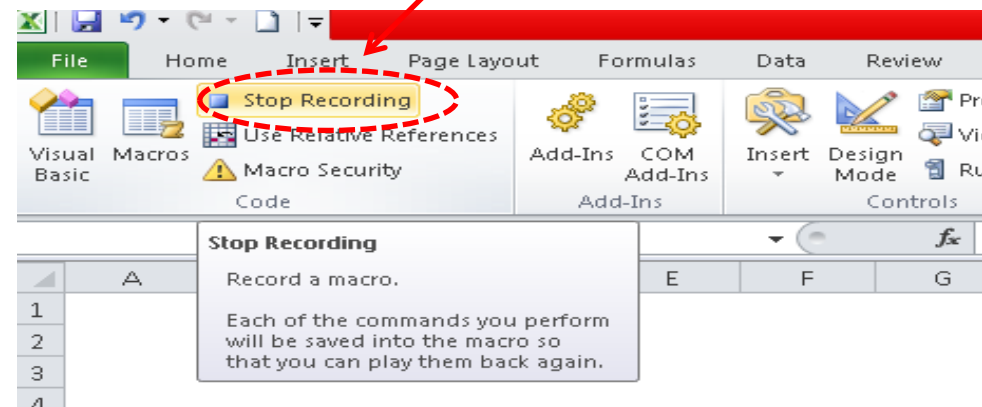
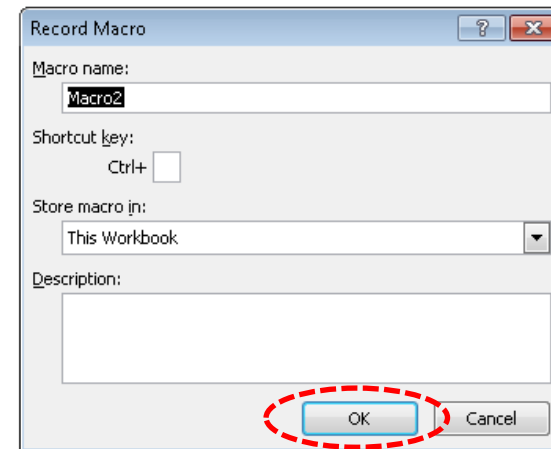
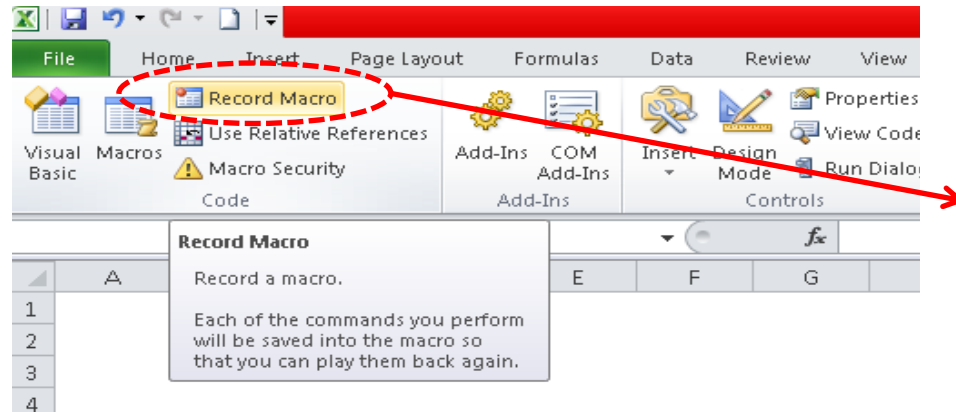
Este método es especialmente recomendable cuando necesitemos trabajar con objetos de Excel tales como libros, hojas, rangos de celdas, celdas, gráficos, etc... o si vamos a usar algún complemento o herramienta de Excel, como filtros, importación de datos, solver, etc. Para generar una macro automática, la **grabaremos**:



Es muy importante dejar de grabar cuando hayamos finalizado.

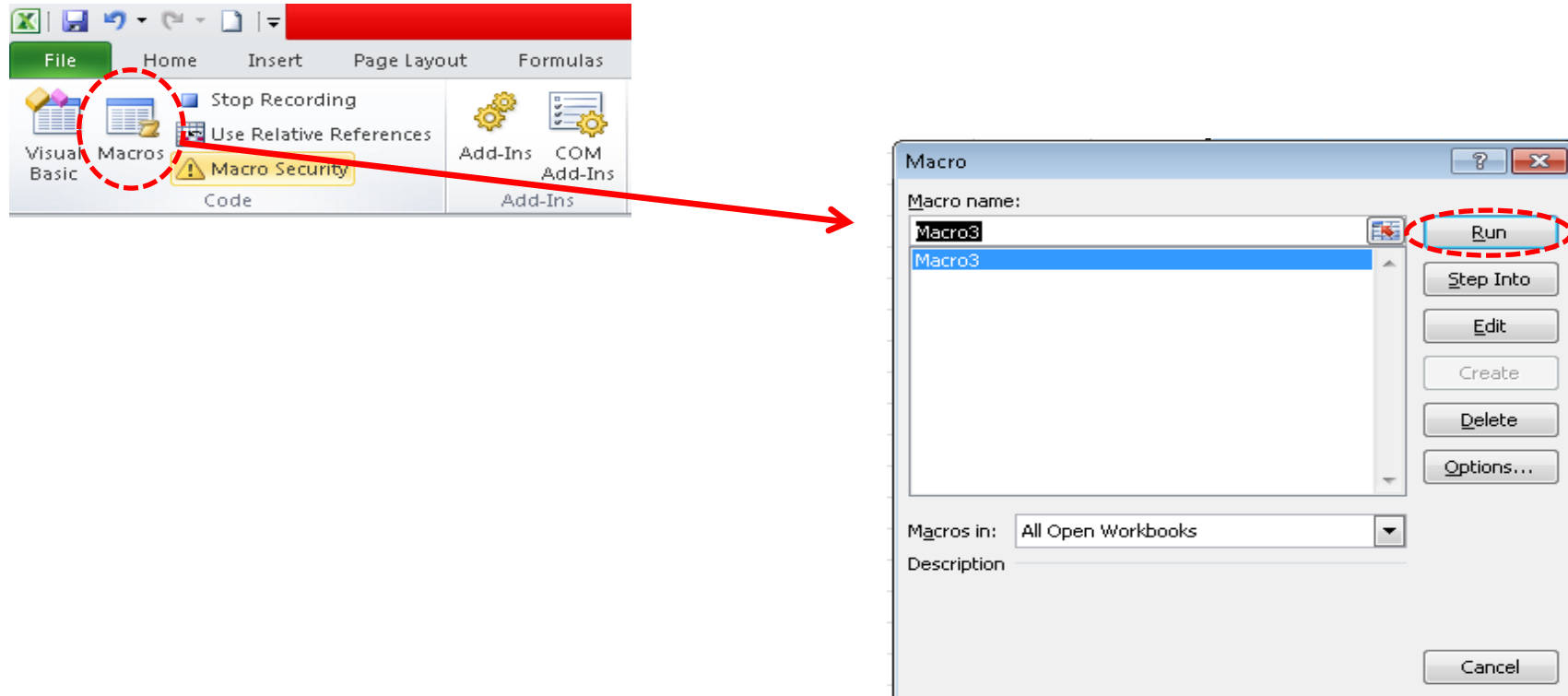
- ¿Cómo programar?

Generación Automática de Macros



- ¿Cómo programar?
Ejecutar una macro automática desde Excel

Una macro puede ser ejecutada desde Excel en **Alt+F8** o usando el botón **Macro** en el menú '*Desarrollador*'.

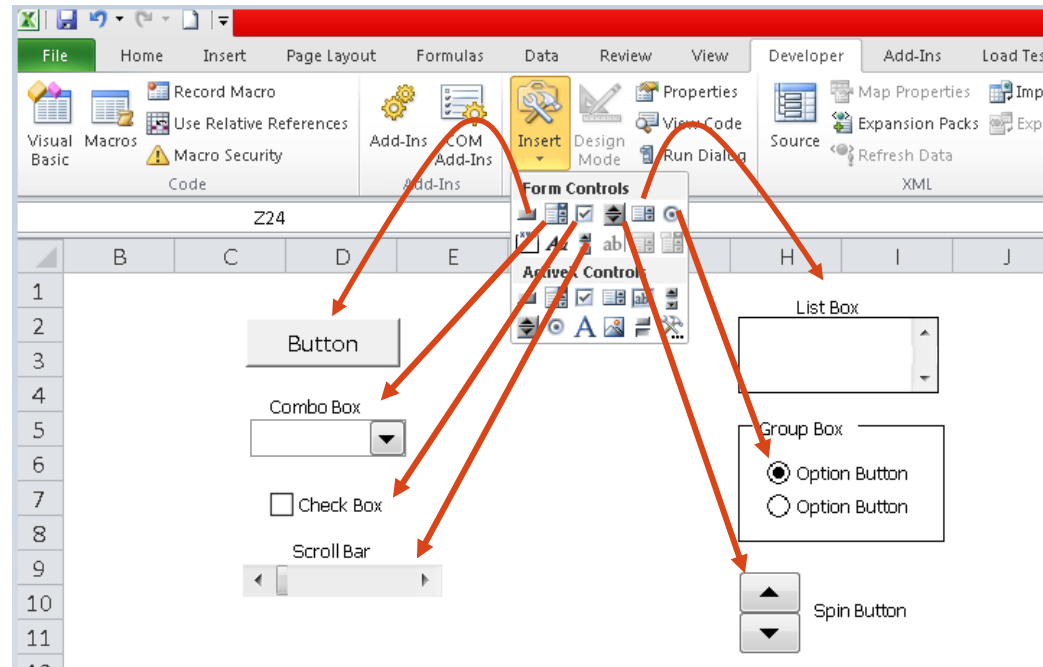


- ¿Cómo programar?

Generación Automática de Macros

Las macros también pueden ser asignadas a botones u objetos de Excel (formas, imágenes, etc.) y ejecutadas con un click de ratón o un cambio en su forma. En el caso de los botones:

Desarrollador => Insertar => Controles de formulario



Asignando la macro... Click derecho en el objeto => 'Asignar macro' => Seleccionar la macro

• ¿Cómo programar?

Ejercicio: Calcular la TIR de un bono a 10 años



En el libro '*E1_TIR_Bono_Macro*' se encuentra una valoración de un bono a 10 años. La idea del ejercicio es construir una subrutina (grabando una macro) que use la opción '**Buscar Objetivo**' (*Datos => Análisis de hipótesis => Buscar objetivo*) de Excel para calcular la Tasa Interna de Retorno (TIR). Esta macro debe ser asignada a una figura y probada cambiando el cupón del bono.

Recuerde...

La *Tasa Interna de Retorno (TIR)* es una tasa de descuento que hace el Valor Actual Neto de los flujos de caja igual a cero. Tal que la TIR es:

$$0 = \sum_{i=0}^n \frac{FC_i}{(1 + TIR)^i}$$

- FC_i es el flujo de caja del periodo i .
- FC_0 es la inversión inicial (precio del bono en nuestro caso).

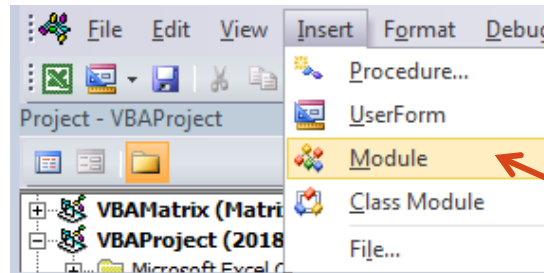
Debido a la naturaleza de esta fórmula, la TIR no puede ser calculada analíticamente y debe ser calculada mediante prueba y error o usando un software programado para ello.

• ¿Cómo programar?

Programación pura en Visual Basic

Lo primero de todo para crear un código en VBA es crear un modulo si todavía no existe.

- ✓ Se pueden insertar tantos módulos como necesitemos para organizar adecuadamente nuestro código.
- ✓ Cada modulo puede contener varias funciones y/o subrutinas.
- ✓ El nombre de los módulos se puede cambiar en la ventana de propiedades.



• ¿Cómo programar?

Programación pura en Visual Basic: Funciones



VBA nos ofrece numerosas funciones ya implementadas, que el usuario puede ejecutar, siempre respetando el formato de llamada. Estas funciones permiten desarrollar cualquier procedimiento en consonancia con las necesidades del usuario. (*Ver funciones VBA más comunes en el Apéndice I*)

Sin embargo, el usuario puede construir sus propias funciones VBA, llamándolas desde Excel u otros procedimientos de Función o Subrutina. Se debería de respetar la siguiente estructura:



Código	Descripción	¿Obligatorio?
Function NombreFuncion(Argumentos)	Es el comienzo de la función. El <i>NombreFuncion</i> es elegido por el usuario. Los <i>Argumentos</i> en paréntesis son las variables que necesitamos introducir. Si los paréntesis están vacíos entonces no hay necesidad de introducir ningún parámetro.	Sí
Dim Variables (CODIGO)	Declaración interna de variables. Se recomienda declarar todas las variables al comienzo.	No
NombreFuncion =	Indica el resultado que queremos devolver.	Sí
End Function	Indica el fin de la función.	Sí

- ¿Cómo programar?

Ejercicio: Creando nuestra primera función



En la hoja '*Deposito*' del libro '*E2_IntroduccionVBA*' encontrareis las características de un depósito a plazo (nominal, tipo de interés y plazo). Crear una función en VBA para calcular el valor del depósito a plazo a su vencimiento (Valor Futuro):

$$VF = Nominal * (1 + TI * Plazo)$$

Esta función debe tener tres argumentos y podría llamarse **Deposit_FV**.

- ¿Cómo programar?

Programación pura en Visual Basic: Subrutina

Subrutina es un procedimientos que ejecuta operaciones internas pero no devuelve ningún resultado. Se puede trabajar directamente con las celdas de Excel (leyendo o escribiendo datos), haremos uso de la siguiente estructura:

Código	Descripción	¿Obligatorio?
Sub <i>NombreSubrutina()</i>	Es el principio de la subrutina. El <i>NombreSubrutina</i> es elegido por el usuario. Debe estar seguido por paréntesis, normalmente vacío.	Sí
Dim <i>Variables</i> <i>(CODIGO)</i>	Declaración interna de variables. Se recomienda declarar todas las variables al comienzo	No
End Sub	Indica el fin de la subrutina.	Sí

• ¿Cómo programar?

Ejercicio: Creando nuestra primera subrutina



En la hoja '*Deposito*' del libro '*E2_IntroduccionVBA*' encontrareis las características de un depósito a plazo (nominal, tipo de interés y plazo). Esta vez crearemos dos subrutinas para calcular el valor del depósito a plazo a su vencimiento (Valor Futuro):

$$FV = Notional * (1 + IR * Term)$$

1. Crear una Subrutina (**Deposit_Sub_1**) con la fórmula de arriba usando la clave '*range*' para acceder a las celdas necesarias.
2. Crear una nueva subrutina (**Deposit_Sub_2**) que llame a la función creada en el ejercicio 3. Usar '*range + Offset*' para acceder a las celdas.

Usar un botón en la hoja de Excel para asignar las macros.

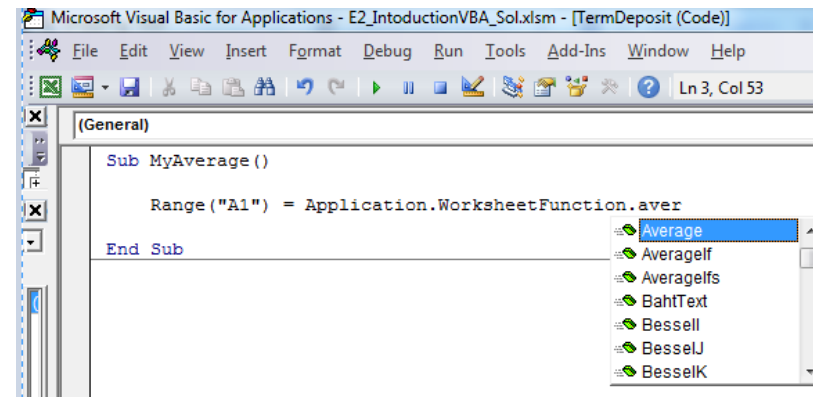
- ¿Cómo programar?
Funciones a la hoja de trabajar

Recordar que ...

VBA también nos permite usar **Funciones de Excel**. Para hacer esto, debemos escribir, '***Application.WorksheetFunction.***' y el nombre de la función (siempre en inglés). Acto seguido, debemos introducir los argumentos de la función separados por **comas**.



Application.WorksheetFunction.



5 | Condicionales en VBA

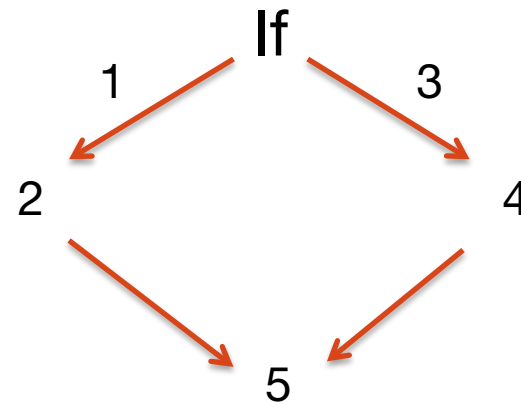
• Declaraciones condicionales en VBA

Condicional en VBA: If ... Then

If: Ejecuta condicionalmente un grupo de instrucciones, dependiendo del valor de una expresión. **Sintaxis:**

```
If <expresion> Then
    [instrucciones]
[Else]
    [Instrucciones_else]
End If
```

```
Function MyMax(inp1,inp2)
1   If inp1 >= inp2 Then
2       MyMax = inp1
3   Else
4       MyMax = inp2
5   End If
End Function
```



• Declaraciones condicionales en VBA

Condicional en VBA: If ... Then

Alternativas a la declaración "If":

Solo se quiere ejecutar un bloque, si se satisface una condición:

```
If <expresion> Then
    [instrucciones]
End If
```

Si se quiere escoger un bloque **entre 3 o más** posibilidades:

```
If <expresion1> Then
    [instrucciones1]
[Elseif <expresion2> Then]
    [Instrucciones_2]
[Elseif <expresion3> Then]
    [Instrucciones_3]
[Else]
    [Instrucciones_else]
End If
```

• Declaraciones condicionales en VBA

Operadores útiles en VBA

Para comparar

< menor que

> mayor que

= igual que

<= menor o igual que

>= mayor o igual que

<> distinto de

Lógicos

Expresión1 **And** Expresión2 (AMBAS condiciones deben cumplirse)
Expresión1 **Or** Expresión2 (al menos UNA condición se debe cumplir)
Not (Expresión) (Realiza una operación lógica negativa de la expresión)

• Declaraciones condicionales en VBA

Declaraciones condicionales en VBA: Select Case

Select Case: Ejecuta una de varias instrucciones grupales, dependiendo del valor de una expresión. **Sintaxis:**

```
Select Case test_expression
    [Case list_expression-n
        [instructions-n]]
    [Case Else
        [instructions-else]]
End Select
```

```
Function Returns(First, Second, Ret_Type)
    Select Case Ret_Type
        Case "Linear"
            Returns = (Second/First)-1
        Case "Log"
            Returns = Log(Second/First)
    End Select
End Function
```

6 | Bucles VBA

• Bucles VBA

Bucles: For ... Next

For: Repite un grupo de instrucciones un número **específico** de veces.

Sintaxis:

```
For <iterator = First To Last>  
    [instructions]  
Next iterator
```

```
Sub Returns_Sub()  
1   For i = 1 To 10  
2       Aux = (Range("A1").Offset(i,0).Value / Range("A1").Offset(0,0).Value) - 1  
3       Range("A1").Offset(i,1) = Aux  
4   Next i  
End Function
```



Los bucles pueden ser decrecientes:
For <iterator = **Last To First Step -1**>
 [instructions]
Next iterator

• Bucles VBA

Bucles : Do While ... Loop

Do While: Repite un bloque de instrucciones **mientras** se cumple una condición. **Sintaxis:**

Do While <expression>
[instructions]

Loop



El iterador debe ser inicializado. Cuidado con los **bluces infinitos**.

```
Sub LastCell()  
1  i = 1  
2  Do While Range("A1").offset(i,0) <> ""  
3    i = i + 1  
4  Loop  
5  Range("A1").Offset(i,0) = "LastCell"  
End Function
```

• Bucles VBA

Bucles : Do Until ... Loop

Do Until: Repite un bloque de instrucciones **hasta** que se cumple una condición. **Sintaxis:**

Do Until <expression>
[instructions]

Loop

```
Sub LastCell()  
1  i = 1  
2  Do Until Range("A1").offset(i,0) = ""  
3      i = i + 1  
4  Loop  
5  Range("A1").Offset(i,0) = "LastCell"  
End Function
```



El iterador debe ser inicializado. Cuidado con los **bluces infinitos**.



Si queremos el mismo resultado que con "**While**", es necesario cambiar el operador

- Bucles VBA

Ejercicio: Uso de Condicionales y Bucles



En la hoja 'SumarSi' del libro 'E2_IntroduccionVBA' encontrareis una base de datos con activos y nominales. El objetivo es crear una **Función** para sumar el nominal total de un activo:

Function MiSumaSi(activo)

1. Usar un **Bucle** para iterar en la primera columna del conjunto de datos (Recuerde **inicializar** el iterador antes de usar el bucle Do While/Until).
2. Usar un condicional para **añadir** el nuevo nominal al antiguo, almacenado en una variable (También inicializar la variable como 0).

• Bucles VBA

Ejercicio: Interpolando



En la hoja '*Interpolacion*' del libro '*E3_Interpolacion*' encontrareis un conjunto de datos con fechas y tipos. El objetivo es crear una **Función** para interpolar un tipo dada una fecha:

1. Usar condicionales para asignar un resultado predefinido si la fecha no está en el rango de fechas
2. Usar un bucle para hallar las fechas anterior y posterior de una fecha dada y evaluar la fórmula de interpolación

$$Tipo = Tipo_{Antes} + \frac{Tipo_{Después} - Tipo_{Antes}}{Fecha_{Antes} - Fecha_{Después}} (Fecha - Fecha_{Antes})$$



Consejo

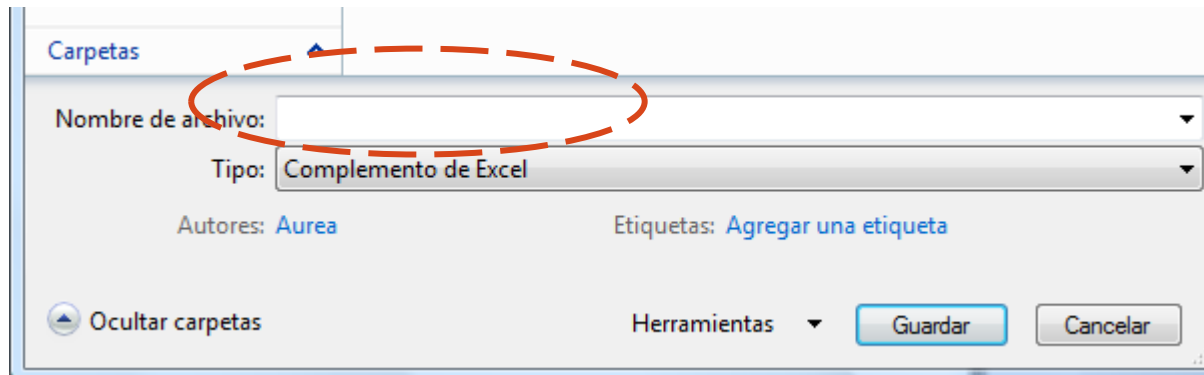
Para saber cuantas filas tiene un rango:
`MiRango.Rows.Count`

| 7 Generación de complementos .XLAM

• Generación de complementos .XLAM

GENERACIÓN DE UN COMPLEMENTO (FICHERO *.XLAM)

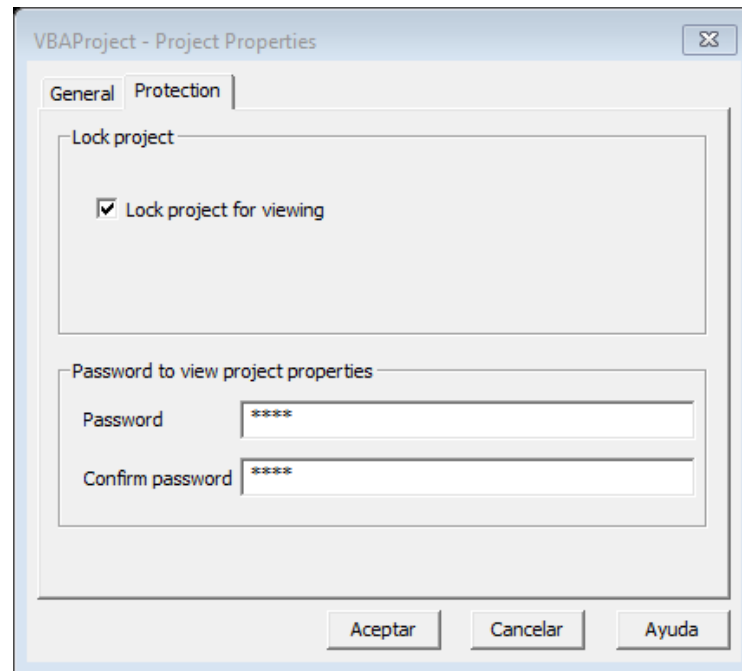
- Las macros definidas por el usuario, si están en un fichero .XLS (.XLSM) exigen que dicho fichero esté abierto en Excel para poder ser ejecutadas. Si tenemos en un fichero un conjunto de macros que usamos habitualmente y queremos evitar tener que abrir diariamente dicho fichero, podemos automatizar su carga mediante la generación de un complemento de tipo .XLAM.
- **GENERACIÓN DE UN COMPLEMENTO (FICHERO *.XLAM)**
- Teniendo el fichero donde se encuentran las macros abierto, se debe seguir la siguiente secuencia:
- Entorno Excel=> Archivo => Guardar Como => Otros formatos => Complemento de Excel



• Generación de complementos .XLAM

Protección de un complemento XLA

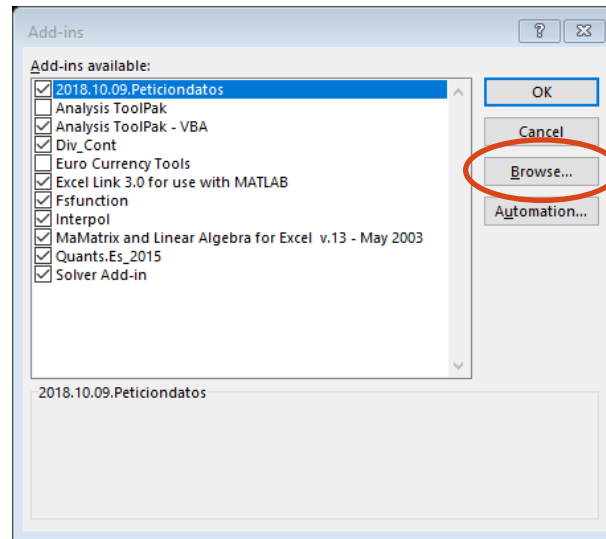
- Previamente a grabar el complemento, se pueden cambiar sus propiedades para que permanezca oculto y para ser visualizado exija una clave.
- Para configurar esto, esto seleccionar Herramientas => Propiedades de VBAProject
- En la lengüeta Protección, marcar la casilla “Bloquear el proyecto para visualización” e introducir una clave.



- Generación de complementos .XLAM

Inclusión de un nuevo complemento

- Archivo => Opciones => Complementos => Ir...



- Si aparece directamente lo seleccionamos.
- Si no aparece, pulsando el botón examinar (ver recuadro superior), buscamos el complemento en el directorio donde se ha creado. Tras pulsar aceptar, la próxima vez que arranquemos el Excel nos cargará automáticamente las macros del complemento incorporado

8 | Interacción con el usuario

- Interacción con el usuario

Interacción con el usuario



Diferentes formas de interactuar con el usuario:

- ✓ **Msgbox**: Comunicación predefinida
- ✓ **Inputbox**: Asignación de variable
- ✓ **UserForm**: Personalizar comunicación (VBA)

• Interacción con el usuario

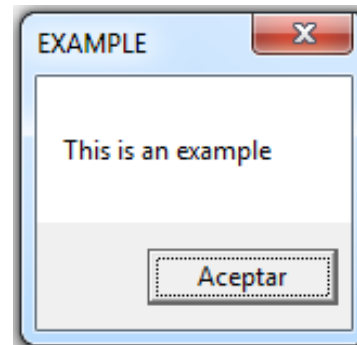
Msgbox

- ❑ Nos **mostrará** en la pantalla un mensaje que se introduce como un argumento.
- ❑ El usuario puede poner un título al msgbox rellenando el argumento Title.

Ejemplo


```
Sub Ex_Msgbox1()  
    MsgBox "This is an example", [Title]:="EXAMPLE"  
End Sub
```

Resultado



• Interacción con el usuario

Msgbox

- ❑ Para **interactuar** con Excel, son necesarios más argumentos.
- ❑ En código VBA, debemos asignar el resultado a una nueva variable.
- ❑ El argumento **obligatorio** en este caso es “**Buttons**”. 
- ❑ Los botones más comunes son:
 - **vbOKCancel**
 - **vbOKOnly**
 - **vbYesNo**
 - **vbYesNoCancel**

- Interacción con el usuario
Msgbox (Código)

```
Sub Ex_Msgbox2()
```

```
' Primero tenemos que saber qué es lo que quiere el usuario
```

```
output = MsgBox("This is an example", vbYesNoCancel, "vbYesNoCancel option")
```

```
' Después, Realizamos la acción en varios casos
```

```
If output = vbYes Then
```

```
    Range("A1") = "Yes"
```

```
Elseif output = vbNo Then
```

```
    Range("A1") = "No"
```

```
Elseif output = vbCancel Then
```

```
    Range("A1") = "Cancel"
```

```
End If
```

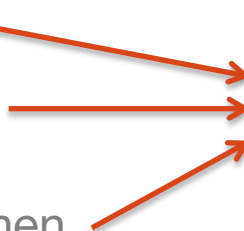
```
End Sub
```

Opción escogida



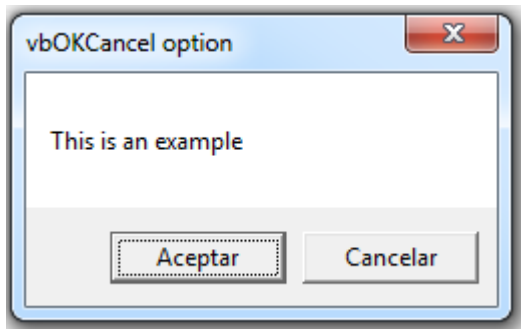
Nueva variable

Posibles outputs

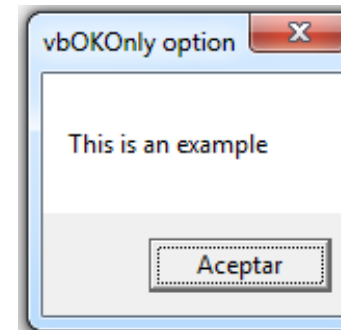


- Interacción con el usuario
Msgbox (Diálogo)

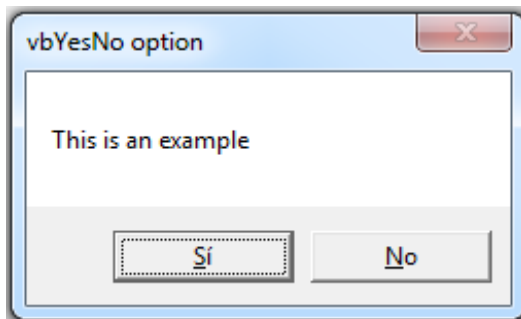
✓ **vbOKCancel**



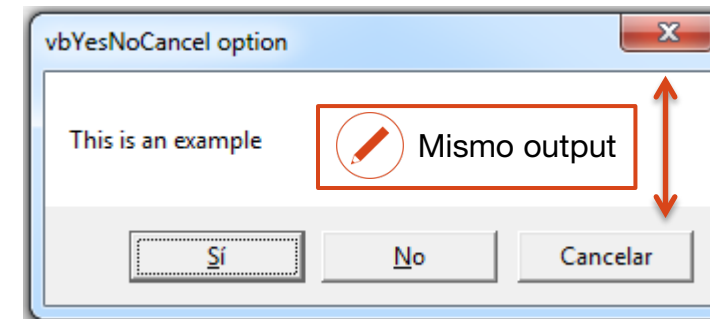
✓ **vbOKOnly**



✓ **vbYesNo**



✓ **vbYesNoCancel**



• Interacción con el usuario

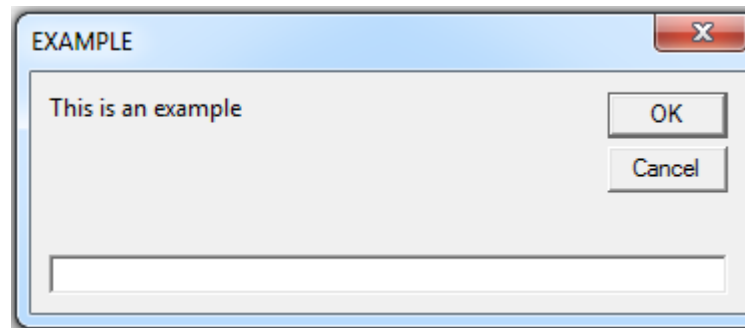
Inputbox

- ❑ Nos **mostrará** en la pantalla un mensaje que se introduce como un argumento.
- ❑ Esto pregunta al usuario por el input, que será **asignado a una nueva variable** en código VBA.

Ejemplo

```
Sub Ex_Inputbox()  
    variable = InputBox("This is an example", "EXAMPLE")  
    Range("A1") = variable  
End Sub
```

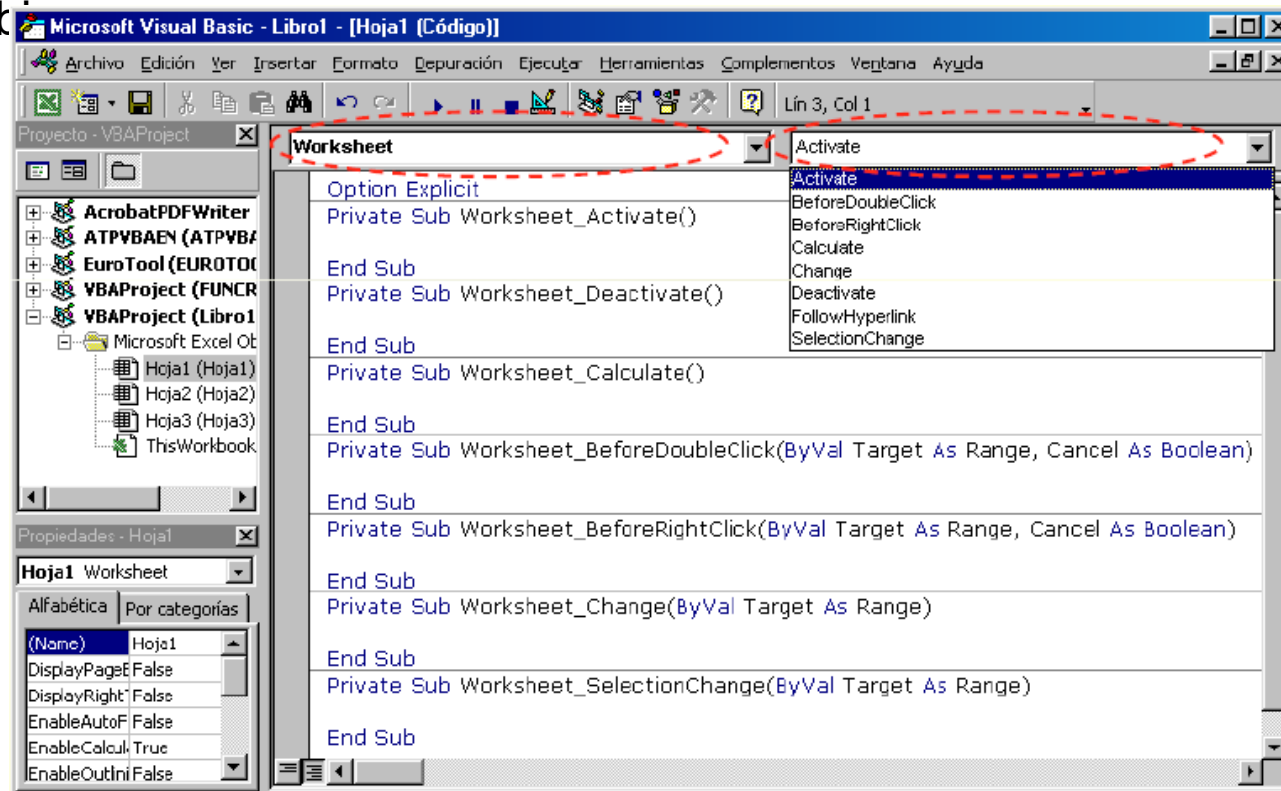
Resultado



9 | Eventos asociados a hojas y libro de Excel

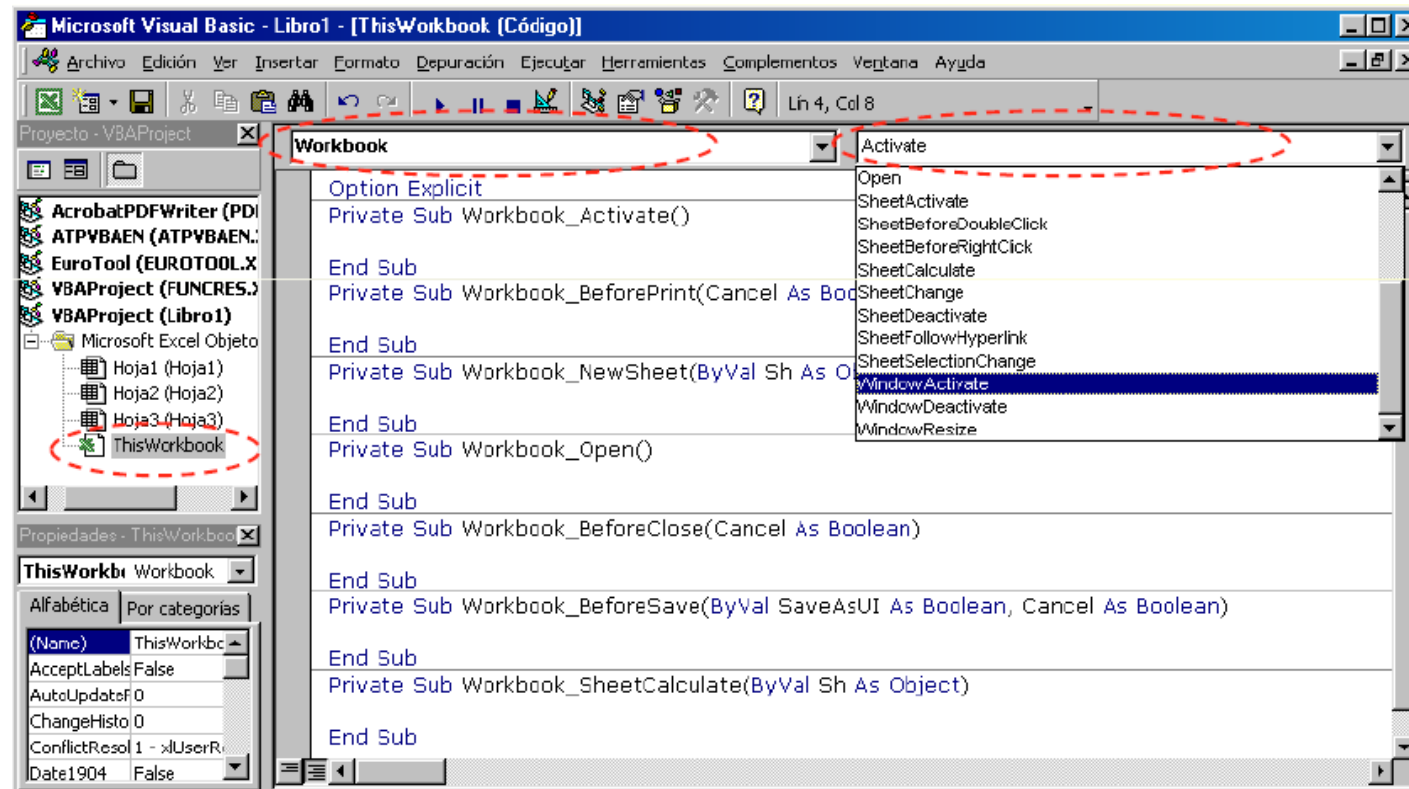
Eventos asociados a Hojas

- Cualquier hoja de un libro Excel puede tener asociada la ejecución de código Visual Basic ante ciertas acciones que ocurran en esa hoja. En el editor de Visual Basic, en cada Hoja podrán crearse subrutinas que se ajusten a ciertos nombres. Estos procedimientos pueden ser insertados directamente de los dos desplegables disponibles:



Eventos asociados a Libros

- Cada libro puede tener asociada la ejecución de subrutinas ante ciertas acciones. En el editor de Visual Basic, en el elemento ThisWorkbook del libro en cuestión se crearán dichas subrutinas (deberán ajustarse a ciertos nombres). Estos procedimientos pueden ser insertados directamente de los dos desplegables disponibles.



10 | Programación de tareas

Tareas programadas (I)

- Podemos programar la ejecución de una macro (procedimiento tipo Sub) a una hora determinada. Dicha macro puede realizar cualquier tipo de actividad como llamadas a otras macros (útil para archivar últimas cotizaciones diarias).
- Esta tarea será configurada mediante el método **Ontime** del objeto **Application**.
- **Sub CierreDia()**
 - 'Esta macro debe contener todo el código a ejecutar a la hora prevista de cierre.
 - 'Será ejecutada automáticamente a la hora indicada en el método Application.OnTime que se
 - ' activa en la macro ActivarReloj puesta más abajo.
 - MsgBox "Tarea Activada a las 18:00:00"
 - 'Introducir aquí todas las sentencias y rutinas a ejecutar.
- **End Sub**
- **Sub ActivarReloj()**
 - 'Esta macro debe ejecutarse una vez (por ejemplo al abrir el libro) para que se programe
 - Application.OnTime TimeValue("18:00:00"), "CierreDia"
 - ' El parámetro segundo es la macro que se ejecuta a las 18.00. OnTime admite más opciones.
 - ' Si pones el cursor sobre la palabra y pulsas F1 veras su ayuda y sus posibilidades.
- **End Sub**

Tareas programadas (II)

- Si deseamos ejecutar alguna tarea automáticamente al abrir un libro, podemos utilizar la macro **Auto_Open**. Si queremos ejecutar una tarea con un cierto retardo tras la apertura de un libro, podemos hacer uso del método **Ontime** del objeto **application**. Podremos programar la tarea a partir de la hora actual, que obtendremos llamando a la función **Time**. La hora devuelta es una cifra que varía entre 0 y 1 (1 para las 23:59:59).
- **Sub Apertura()**
 - 'Esta macro debe contener todo el código a ejecutar a la hora prevista.
 - 'Será ejecutada automáticamente a la hora indicada en el método Application.OnTime que se
 - ' activa en la macro Auto_Open puesta más abajo.
 - MsgBox "Tarea Activada a la hora actual + 5 minutos"
 - 'Introducir aquí todas las sentencias y rutinas a ejecutar.
- **End Sub**
- **Sub Auto_Open()**
 - 'Esta macro debe ejecutarse una vez (por ejemplo al abrir el libro) para que se programe
 - Dim t, minuto
 - Minuto = 1/ (24 * 60)
 - T = Time
 - Application.OnTime (t + 5 * minuto), "Apertura"
- **End Sub**

Tareas programadas (III)

- También se puede usar:
 - ***Now + TimeValue(tiempo)***: para programar alguna tarea con un cierto intervalo de tiempo.
 - Ejemplo: **Application.Ontime Now + TimeValue("00:00:15"), "Apertura"**
 - Ejecuta la macro "Apertura" dentro de 15 segundos.
 - ***TimeValue(tiempo)***: para programar alguna tarea a una cierta hora.
 - Ejemplo: **Application.Ontime TimeValue("00:00:15"), "Apertura"**
 - Ejecuta la macro "Apertura" a las 5 p.m.

11 | Tratamiento de errores

Tratamiento de errores

- En ocasiones, pueden aparecer errores en tiempo de ejecución, (sobre objetos de Excel), que no permiten continuar con la ejecución del código. Como ejemplo podremos provocar un error de división por cero:

- **Sub Errores()**

- 'Sin control de errores. Provoca un error de división por cero, aparece un mensaje
- 'del sistema que no nos permite continuar la ejecución del código
- Dim x, a
- a=5
- x=a/0 ' <= error de división por cero
- x= x + 33
- Cells(1,1) = x

Cuadro de diálogo del error
Tanto si pulsamos Finalizar como Depurar,
no se podrá continuar la ejecución
mientras exista el error



Tratamiento de errores

- Dentro de una macro, puede activarse el ignorar los errores del sistema, hasta el final de dicha macro. En esta modalidad, si en una línea se produce un error del sistema, **se ignora dicha línea y se continúa la ejecución en la línea siguiente.**
- **Sub IgnorarErrores()**
 - 'Macro que ignora los errores del sistema.
 - 'Si en una línea se produce un error, se ignora esa línea se salta a ejecutar la
 - 'siguiente línea. Este ignorado es efectivo hasta el final de la macro
 - Dim x, a
 - **On Error Resume Next** ' a partir de aquí se ignoran los errores que se produzcan
 - a=5
 - x=a/0 ' <= error de división por cero. Esta línea no será ejecutada
 - x= x + 33
 - Cells(1,1) = x
 - **End Sub**

Tratamiento de errores

- Podremos provocar que si se produce un error del sistema en una línea, **se ignora dicha línea y se salta a otra línea para hacer un tratamiento específico del error**. Podremos saber las características del error mediante el objeto **Err** (por ejemplo, el número de error está en **Err.Number**)
- **Sub ControlErrores()**
 - Dim x, a, c
 - **On Error GoTo ControlErr** 'activa control de errores (válido hasta el final del sub)
 - a=5
 - x=a/0 ' <= error de división por cero. Esta línea no será ejecutada y salta etiqueta ControlErr
 - x= x + 33
 - Cells(1,1) = x
 - Exit Sub 'debemos evitar que entre en el tratamiento de los errores
- **End Sub**

Tratamiento de errores

- Podremos provocar que si se produce un error del sistema en una línea, **se ignora dicha línea y se salta a otra línea para hacer un tratamiento específico del error**. Podremos saber las características del error mediante el objeto **Err** (por ejemplo, el número de error está en **Err.Number**)
 - Sub ControlErrores()**
 - Dim x, a, c
 - On Error GoTo ControlErr** 'activa control de errores (valido hasta el final del sub)
 - a=5
 - x=a/0 ' <= error de división por cero. Está línea no será ejecutada y salta etiqueta ControlErr
 - x= x + 33
 - Cells(1,1) = x
 - Exit Sub 'debemos evitar que entre en el tratamiento de los errores
 - ControlErr :**
 - c = MsgBox("Error num:" & Err.Number & ". Continuar?", vbYesNo)
 - If c = vbYes Then
 - 'podría hacer cualquier tipo de manipulación
 - Resume Next 'continua en la siguiente a la línea de error
 - End if
 - End Sub**
 - Nota:** si se desea desactivar control de errores: On Error GoTo 0



Apéndice I: Librería VBA de Funciones

- Librería VBA de Funciones

Funciones VBA más usadas

VBA tiene una librería de funciones a nuestra disposición. Aquí, mostramos algunas de las más importantes, especialmente para finanzas.

Función	Descripción
Abs (number)	Nos devuelve el valor absoluto de un número
Int (number)	Nos devuelve la parte entera de un número
Date	Nos devuelve la fecha actual del sistema
Now	Nos devuelve la fecha y hora actuales del sistema
Time	Nos devuelve la hora actual del sistema
DateSerial (year,month,day)	Nos devuelve una fecha (número) dados los valores de un año, mes y día
DateValue ("22/09/2007")	Nos devuelve el valor en serie de una fecha
Day (date)	Nos devuelve el día del valor de una fecha
Month (date)	Nos devuelve el mes del valor de una fecha
Year (date)	Nos devuelve el año del valor de una fecha
Weekday (date)	Nos devuelve el día de la semana del valor de una fecha, considerando el domingo como día 1

- Librería VBA de Funciones
Funciones VBA más usadas

Función	Descripción
Dateadd (interval, number, date)	<p>Añade un número de intervalos de tiempo a una fecha proporcionada y devuelve la fecha resultante. El intervalo podría ser: “yyyy” para años, “q” para trimestres, “m” para meses, “ww” para semanas y “d” para días.</p> <p>Ejemplo:</p> <p><code>Date2 = DateAdd(“yyyy”, 2, “22/10/2017”) => Date2 = “22/10/2019”</code></p>
Datediff (interval, date1, date2)	<p>Nos devuelve la diferencia entre dos fechas, basado en un intervalo específico. El intervalo podría ser: “yyyy” para años, “q” para trimestres, “m” para meses, “ww” para semanas y “d” para días.</p> <p>Ejemplo:</p> <p><code>Diff = DateDiff(“m”, “22/08/2019”, “22/10/2019”) => Diff = 2</code></p>

- Librería VBA de Funciones

Funciones VBA más usadas

Función	Descripción
Rnd	Nos devuelve un número aleatorio entre 0 y 1
Sqr (number)	Nos devuelve la raíz cuadrada de un número
Sgn (number)	Nos devuelve el signo de un número (representado como un entero: -1,0,1)
Log (number)	Nos devuelve el logaritmo natural de un número
Exp (number)	Nos devuelve el valor de la función exponencial de un número dado
Format (expression, format)	Aplica un formato específico a una expression y devuelve el resultado como una cadena. Ejemplo: Time = #20:12:15# TimeFormat = Format(Time, "hh:mm:ss AMPM") => "08:12:15 PM"
CDate	Convierte un valor a una fecha
CDbl	Convierte un valor a un decimal

- Librería VBA de Funciones

Funciones VBA más usadas

Función	Descripción
isArray + (Variable) isDate isEmpty isNumeric isMissing	Nos devuelve un valor booleano (Verdadero o Falso) que indica si la variable de entrada cumple o no cierta condición
Chdir path	Nos permite cambiar la carpeta o directorio actual
Shell pathname	Ejecuta un comando en el intérprete del sistema operativo
Lbound (array)	Nos devuelve el subíndice más pequeño de un determinado <u>vector</u>
Ubound (array)	Nos devuelve el subíndice más grande de un determinado <u>vector</u>
Len (text)	Nos devuelve el número de caracteres en un <u>texto</u>
Trim (text)	Nos devuelve el valor de un texto con el primer y último espacios borrados
Lcase (text)	Convierte un <u>texto</u> a minúsculas
Ucase (text)	Convierte un <u>texto</u> a mayúsculas

