MICROSOFT EXCEL

HERRAMIENTAS INFORMÁTICAS II

Prof. Ing. Norma Cañizares

Microsoft Excel

Visual Basic para Excel

Visual Basic para Excel

1. Introducción a Programación Orientada a Objetos
2. Terminología en VBA de Microsoft Excel
3. El Núcleo de VBA
4. El Editor de Visual Basic
5. Variables, Constantes, Tipos de Datos y Expresiones
6. Funciones de Conversión de tipos.
7. Operaciones de Entrada/Salida simples

Microsoft Excel

INTRODUCCIÓN A LA PROGRAMACIÓN ORIENTADA A OBJETOS

A la hora de trabajar con macros en Excel, deben tenerse claros ciertos conceptos de la Programación Orientada a Objetos (POO).

No nos extenderemos demasiado sobre la POO, pero si definiremos a continuación los conceptos de Objeto, Propiedades y Métodos.

OBJETO

En el mundo real un objeto puede ser cualquier cosa. Por ejemplo: objetos coche, objetos silla, objetos casa, objeto celular etc.

En POO, la generalización (o definición) de un objeto se llama Clase, así la clase coche seria como la representante de todos los coches del mundo, mientras que un objeto coche seria un coche en concreto. Cuando decimos que la clase coche representa a todos los coches del mundo significa que define como es un coche, cualquier coche.

Dicho de otra forma y para aproximarnos a la definición informática, la clase coche define algo que tiene cuatro ruedas, un motor, un chasis,... entonces, cualquier objeto real de cuatro ruedas, un motor, un chasis,... es un objeto de la clase coche.

OBJETO

Propiedades.

Cada objeto tiene características o propiedades, como por ejemplo el color, la forma, peso, medidas, etc. ***Estas propiedades se definen en la clase y luego se particularizan en cada objeto.***

Así, en la clase coche se podrían definir las propiedades Color, Ancho y Largo , luego al definir un objeto concreto como coche ya se particularizarían estas propiedades a, por ejemplo, Color = Rojo, Ancho = 2 metros y Largo = 3,5 metros.

OBJETO

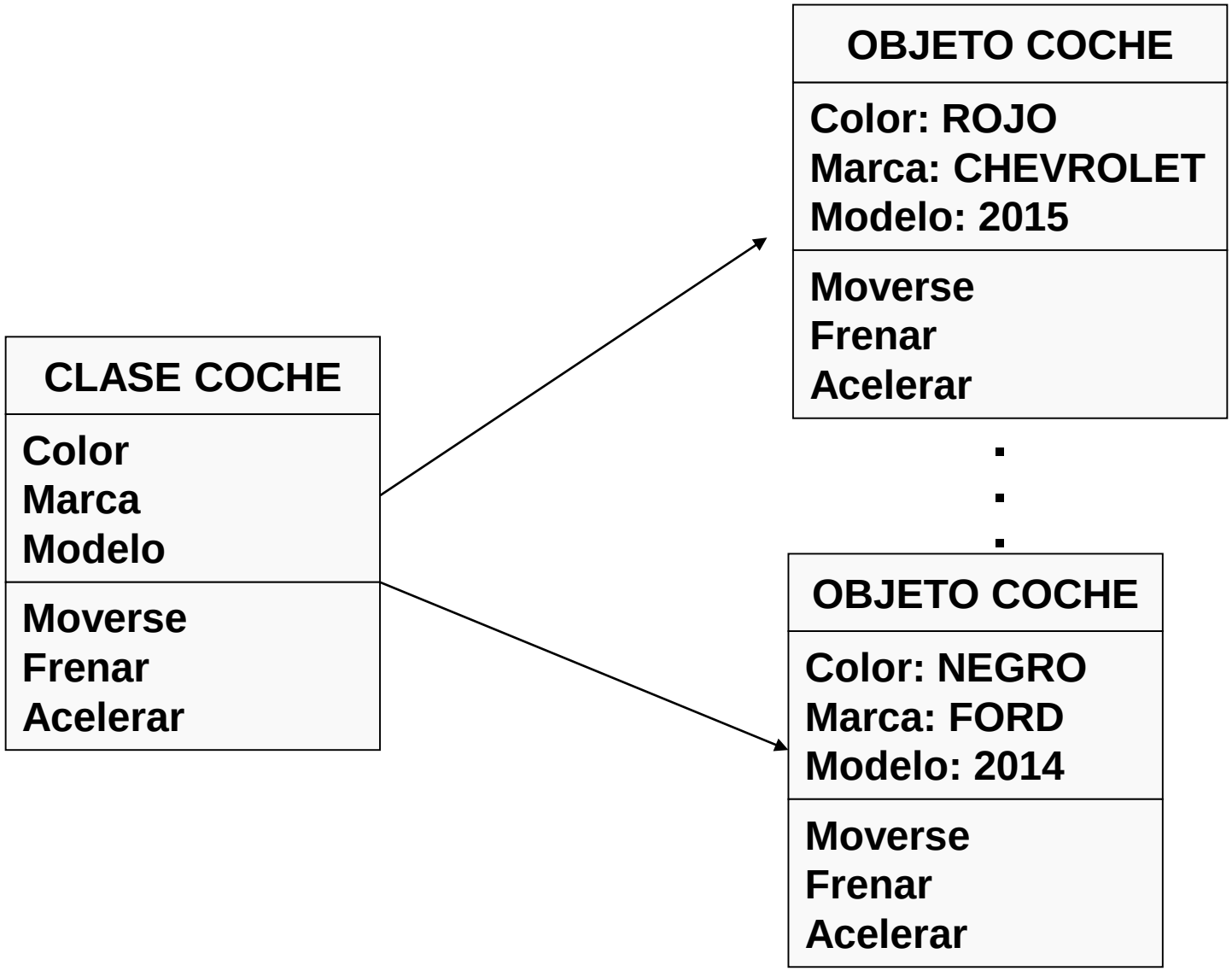
Métodos.

La mayoría de ***los objetos tienen comportamientos o realizan acciones***, por ejemplo, una acción evidente de un objeto coche es el de moverse o lo que es lo mismo, trasladarse de un punto inicial a un punto final.

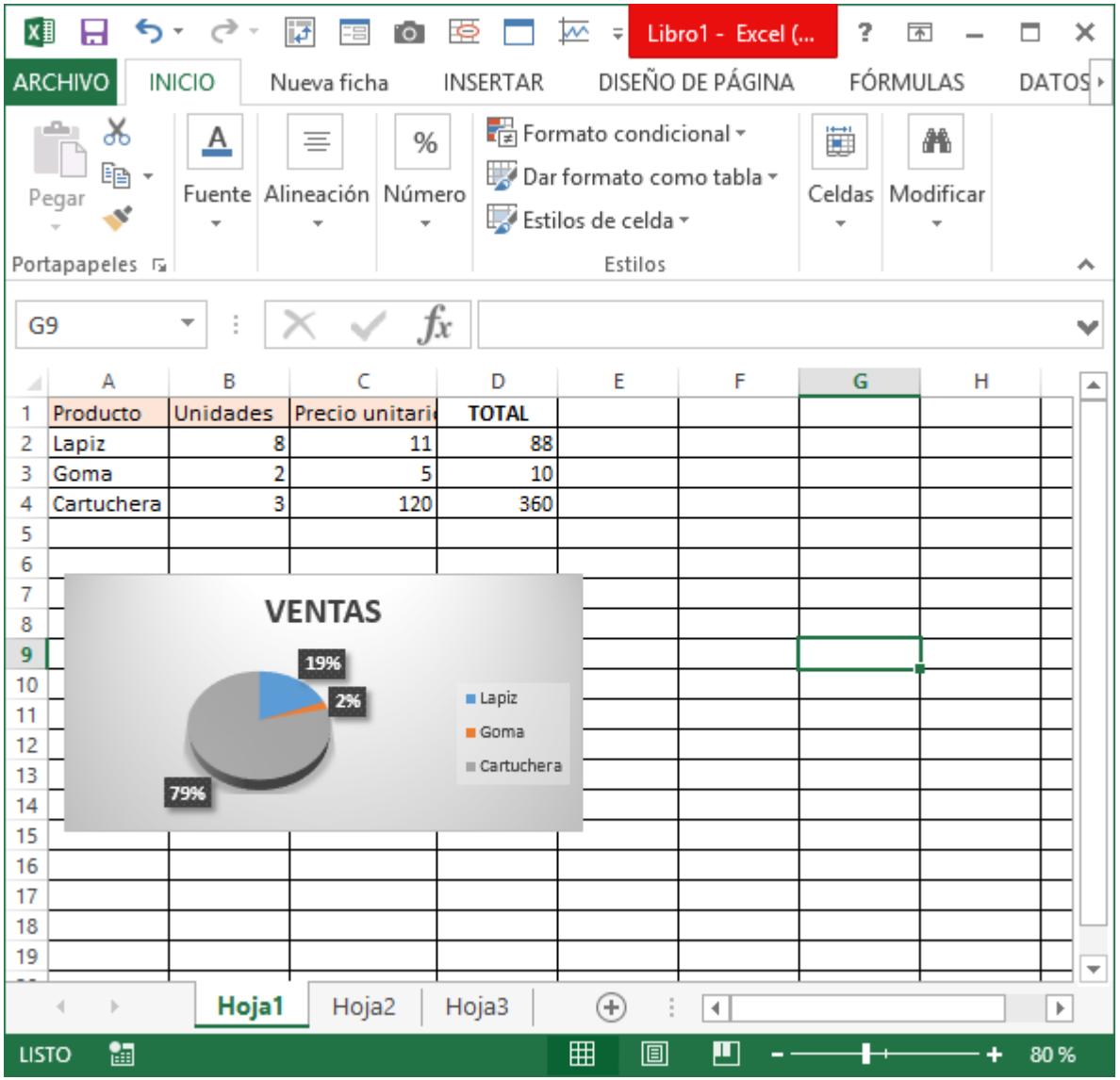
Cualquier proceso que implica ***una acción o pauta de comportamiento por parte de un objeto se define en su clase para que luego pueda manifestarse en cualquiera de sus objetos.***

Así, en la clase coche se definirían en el método mover, todos los procesos necesarios para llevarlo a cabo (los procesos para desplazar de un punto inicial a un punto final), luego cada objeto de la clase coche simplemente tendría que invocar este método para trasladarse de un punto inicial a un punto final, cualesquiera que fueran esos puntos.

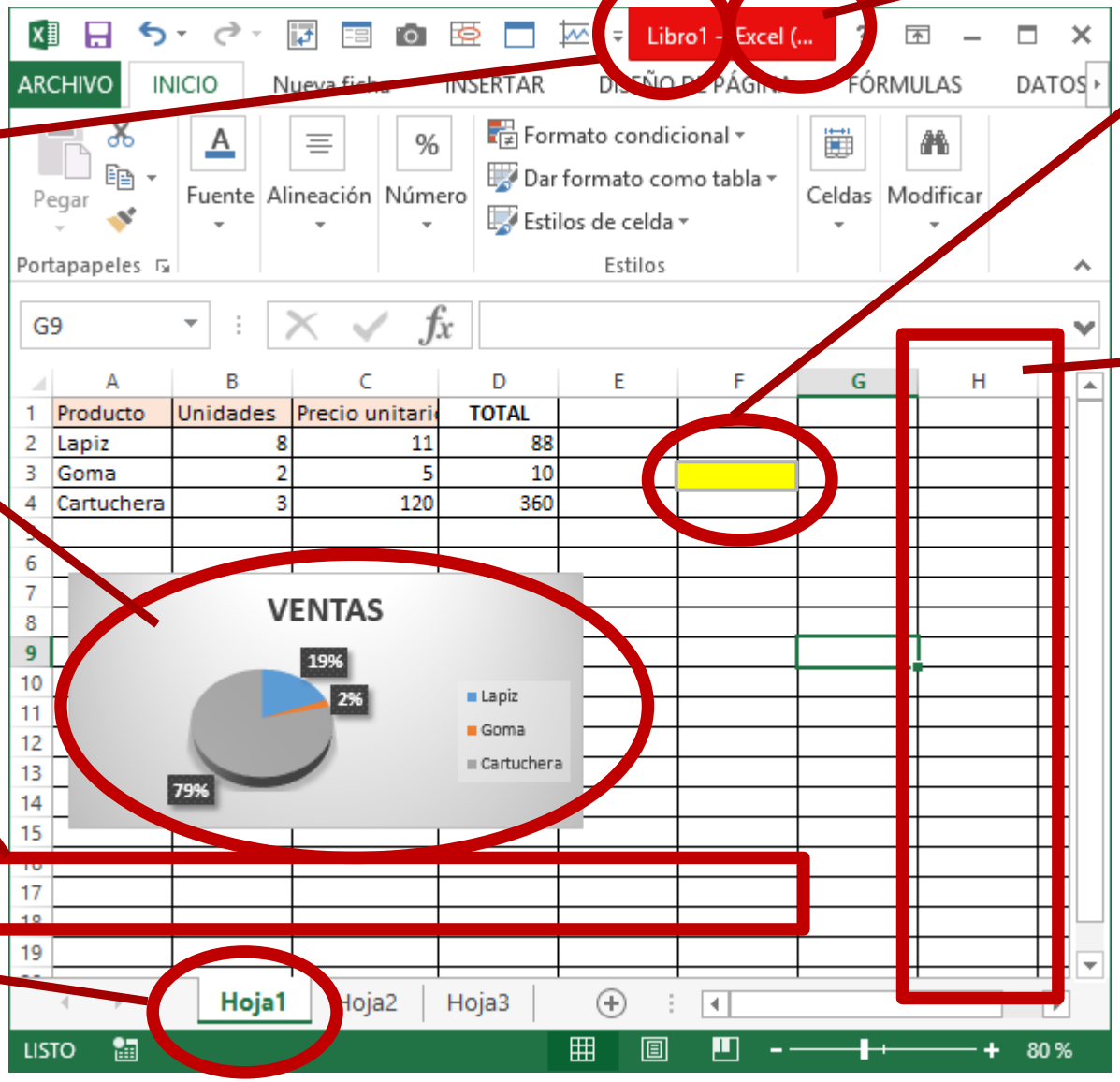
EJEMPLO: DEFINICIÓN DE OBJETOS



EJ. DE OBJETOS EN EL ENTORNO DE EXCEL



EJ. DE OBJETOS EN EL ENTORNO DE EXCEL



Objeto APLICACION

Objeto LIBRO

Objeto CELDA

Objeto COLUMNA

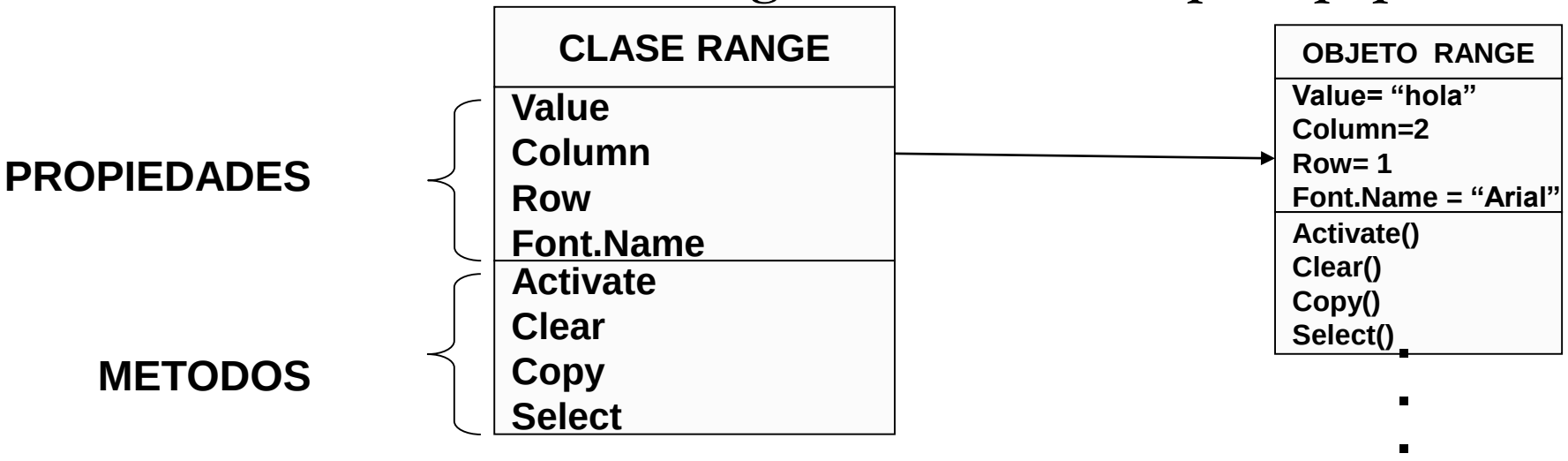
Objeto GRAFICO

Objeto FILA

Objeto HOJA

EJEMPLO DE CONCEPTOS DE POO EN EXCEL

- Un objeto Range está definido por una clase donde se definen sus propiedades. Entre las propiedades de un objeto Range están Value , que contiene el valor de la casilla , Column y Row que contienen respectivamente la fila y la columna de la casilla, Font que contiene la fuente de los caracteres que muestra la casilla, etc.
- Range, como objeto, también tiene métodos. Por ejemplo el método Activate, hace activa una celda determinada, Clear, borra el contenido de una celda o rango de celdas, Copy, copia el contenido de la celda o rango de celdas en el portapapeles,...



Visual Basic para Excel

1. Introducción a Programación Orientada a Objetos
2. Terminología en VBA de Microsoft Excel
3. El Núcleo de VBA
4. El Editor de Visual Basic
5. Variables, Constantes, Tipos de Datos y Expresiones
6. Funciones de Conversión de tipos.
7. Operaciones de Entrada/Salida simples

Microsoft Excel

TERMINOLOGIA EN VBA DE EXCEL

- Centrándonos en Office y en el lenguaje VBA, tenemos:
  - Objeto.
  - Colecciones.
  - Miembros.
  - Eventos.
- **Objeto.** Toda entidad de Office funcional, manejable y programable. En Office, prácticamente todo puede concebirse como objeto (botones, tablas, consultas, documentos, hojas de cálculo, etc.).
- **Colecciones.** Objetos que son del mismo tipo. Así, una colección es un grupo de objetos de la misma clase y esta colección por si misma es un objeto. **Siempre aparecen en plural** (Worksheets, Shapes, Workbooks, etc.). Por ejemplo, Worksheets y Workbooks representan, respectivamente, la colección o conjunto de hojas de cálculo abiertas y de libros abiertos en Excel.

TERMINOLOGIA EN VBA DE EXCEL

- **Miembros.** Conjunto formado por el estado y el comportamiento que se puede aplicar a un objeto concreto. Dentro de este apartado, podemos diferenciar dos conceptos:
    - **Propiedades.** Atributos o características propias de un objeto (tamaño, posición, forma, etc.).
    - **Métodos.** Acciones que pueden realizarse sobre o con un objeto (abrir, cerrar, eliminar, etc.)
- **Eventos.** Sucesos que tienen lugar en un momento dado, ante los cuales puede responder un objeto (pulsar tecla, pulsar botón derecho del ratón, etc.). Por defecto, ante un evento un determinado objeto no proporciona ninguna respuesta, cuya configuración queda en manos del usuario.

Visual Basic para Excel

1. Introducción a Programación Orientada a Objetos
2. Terminología en VBA de Microsoft Excel
3. El Núcleo de VBA
4. El Editor de Visual Basic
5. Variables, Constantes, Tipos de Datos y Expresiones
6. Funciones de Conversión de tipos.
7. Operaciones de Entrada/Salida simples

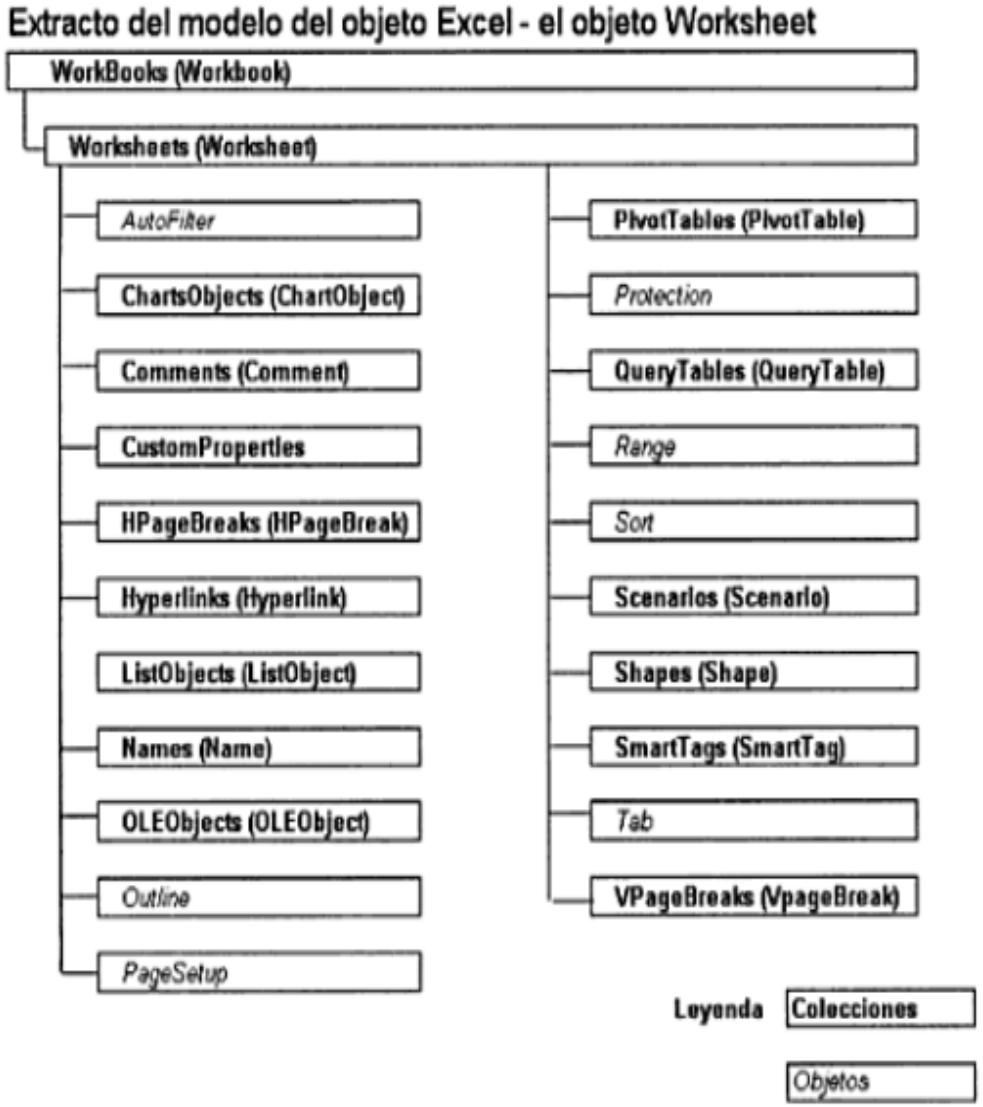
Microsoft Excel

EL NÚCLEO DE VBA

Esencialmente la programación de Excel se reduce a la manipulación de objetos (mediante la escritura de instrucciones en un lenguaje que Excel puede entender), mediante el Lenguaje Visual Basic For Application (VBA).

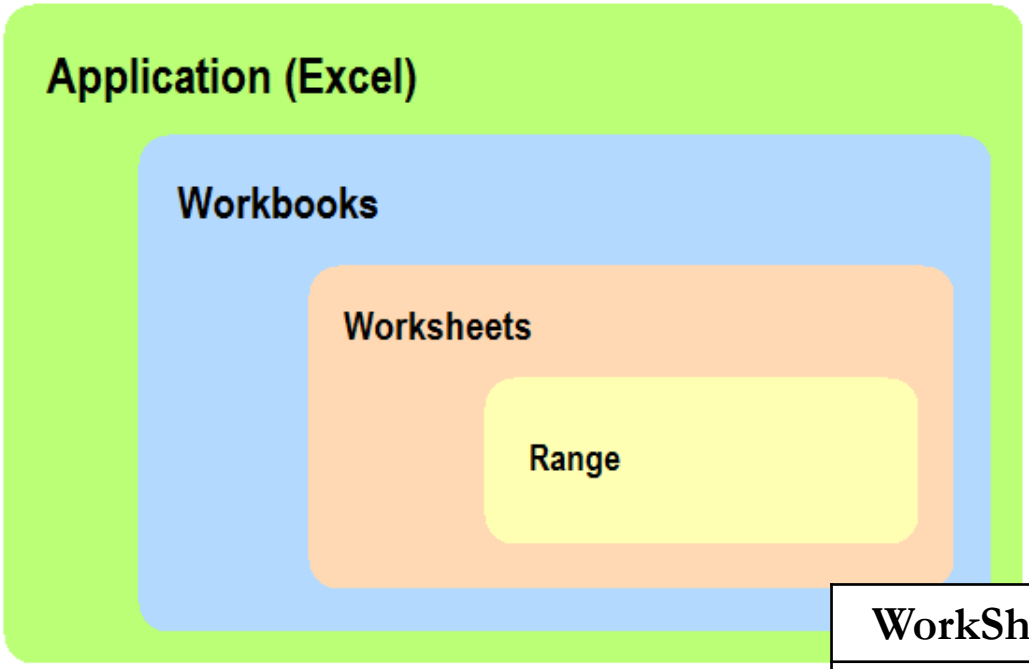
EL NÚCLEO DE VBA – MODELOS DE OBJETOS

- El secreto de usar VBA con otras aplicaciones reside en entender el MODELOS DE OBJETOS para cada Aplicación.
- VBA simplemente manipula objetos, y cada producto (Excel, Word, Access, PowerPoint y demás) posee un modelo de objeto único propio.



EL NÚCLEO DE VBA

JERARQUÍA DEL MODELO DE OBJETOS EN EXCEL

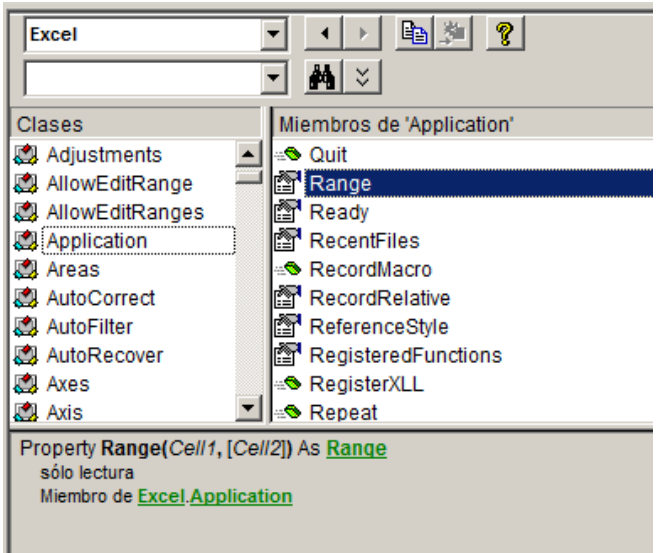


WorkSheet	Objeto hoja
Range	Objeto celda
Application	Objeto Excel
Workbook	Objeto Libro

NAVEGAR POR EL MODELO DE OBJETOS EN EXCEL

Navegar por el modelo de objetos en Excel, permite encontrar las diferentes categorías de objetos que se pueden utilizar para el desarrollo de una macro. Se tienen 2 alternativas para acceder a esta opción:

- Mediante el teclado: presionando el botón F2.
- Mediante la Barra de Menú: Menú Ver / Examinador de Objetos.



EL NÚCLEO DE VBA

Excel proporciona más de 100 clases de objetos para manipular.

Ejemplos de objetos son:

- Un libro de trabajo,
- Una hoja de cálculo,
- Un rango de una hoja de cálculo,
- Un gráfico
- Un rectángulo dibujado.

Las clases de objetos están ordenados jerárquicamente.

Los objetos pueden actuar como contenedores de otros objetos.

Por ejemplo,

Excel es un objeto llamado **Application** y contiene otros objetos como **Workbook**.
El objeto **Workbook** puede contener otros objetos como **Worksheet** y **chart**.
Un objeto **Worksheet** puede contener objetos como **Range**, **Pivottable** y demás.

NOS REFERIMOS AL ORDEN DE ESTOS OBJETOS COMO MODELO DE OBJETO DE EXCEL.

EL NÚCLEO DE VBA

Colecciones.

- Una colección es un conjunto de objetos del mismo tipo, es decir, es un Array de objetos. Por ejemplo, dentro de un libro de trabajo puede existir más de una hoja (Worksheet), todas las hojas de un libro de trabajo forman un conjunto, el conjunto Worksheets.

Para hacer referencia a un único objeto de una colección:

- Colocamos el **nombre del objeto** o el numero del índice entre paréntesis después del **nombre de la colección**, como en el siguiente caso:

Worksheets("hoja1")

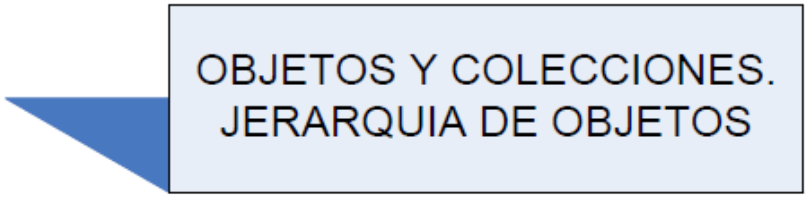
- Si hoja1 es la primera hoja de la colección, también podemos utilizar la siguiente referencia **WorkSheets(1)** y a la segunda hoja seria **WorkSheets(2)**
- También hay una colección especial llamada **Sheets** que esta compuesta por todas las hojas de un libro, tanto si son gráficos, hojas de calculo o no.

Sheets(1)

EL NÚCLEO DE VBA

HACER REFERENCIA A UN OBJETO EN LA JERARQUÍA DEL MODELO DE OBJETOS.

- Las objetos Excel como hemos visto se clasifican en clases y estas se ordenan en jerarquías.



Los objetos se manipulan utilizando código VBA

Los objetos pueden actuar como recipientes de otros objetos

La Unidad Básica de Excel es el Objeto Workbook

JERARQUIA →

Cuando queremos hacer referencia a un objeto (por ej. hoja de calculo) debemos **especificar su posición en la jerarquía utilizando el *punto* como separador entre el contenedor y el miembro.**

HACER REFERENCIA A UN OBJETO EN LA JERARQUÍA DEL MODELO DE OBJETOS: EJEMPLOS

- Podemos hacer referencia a un libro de Excel llamado primas.xlsx de la siguiente forma:

  Application.WorkBooks ("primas.xlsx")

  Esto hace referencia al libro primas.xlsx de la colección Workbooks que se encuentra dentro del objeto Application de Excel.
- Podemos también hacer referencia a la hoja 1 del libro primas.xlsx de la siguiente forma:

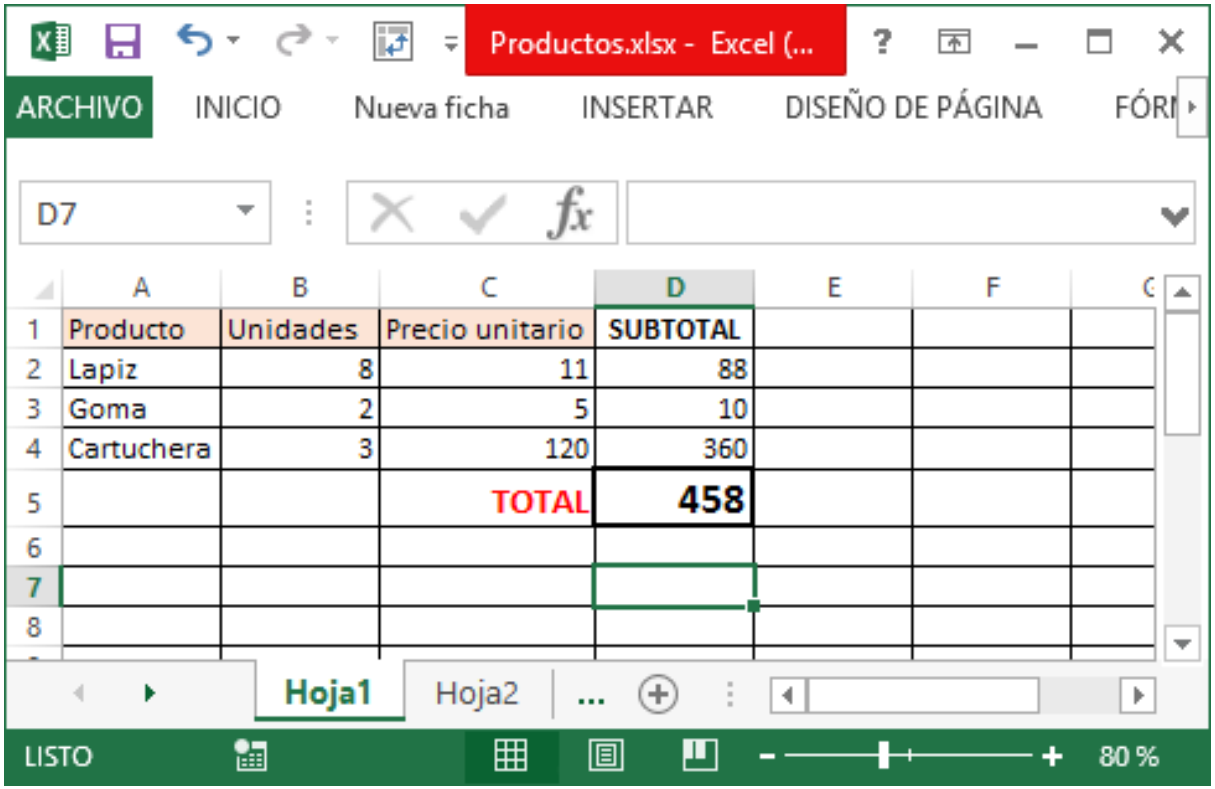
  Application. WorkBooks ("primas.xlsx"). WorkSheets ("hoja 1")
- Avanzando podemos llevarlo a un nivel de desarrollo más detallado y referirnos a la celda A1 de la hoja anterior, de la siguiente forma:

  Application.WorkBooks("primas.xlsx").WorkSheets("hoja 1").Range ("A1")
- Si se omiten una referencia especifica a un objeto Excel utiliza los **objetos activos**. Si el libro activo es primas.xlsx, la referencia anterior se puede hacer más simple como:

  WorkSheets("hoja 1").Range("a1")

EJEMPLO

Siguiendo la Jerarquía del Modelo de Objetos de VBA, ¿Cómo haría referencia al Objeto Celda que contiene el TOTAL de las ventas?



DIFERENTES FORMAS DE HACER REFERENCIA AL MISMO OBJETO

CASO:
Supongamos que tenemos un libro llamado Ventas y es el único libro abierto. Este libro contiene una hoja llamada Resumen. Podemos hacer referencia a la hoja de cualquiera de las siguientes formas:

WorkBooks("Ventas.xlsx").WorkSheets("Resumen")

WorkBooks(1).WorkSheets(1) **WorkBooks(1).Sheets(1)**

Application.ActiveWorkbook.ActiveSheet

ActiveWorkbook.ActiveSheet

ActiveSheet

- El método que utilizaremos estará condicionado normalmente por nuestro conocimiento del espacio de trabajo.
- Por ejemplo si hay mas de un libro abierto, el segundo y tercer método no son fiables.
- Si queremos trabajar con la hoja activa (la que sea), ninguno de los últimos tres métodos funcionara.
- Para estar absolutamente seguros de que estamos haciendo referencia a una hoja especifica de un determinado libro, el primer método es la mejor opción.

SINTAXIS PARA ACCEDER A LAS PROPIEDADES DE UN OBJETO

- Para hacer referencia a las propiedades de cualquier objeto se emplea la siguiente sintaxis:

Objeto.Propiedad = valor

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

Range("a2").Value = 27

SINTAXIS PARA IMPLEMENTAR EL MÉTODO DE UN OBJETO

- Para implementar los métodos de un objeto, se utiliza la siguiente sintaxis:

Objeto.Método

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

Range("a2").Select

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

Thisworkbook.Saveas filename:="empleados.xlsm"

RANGE

- Representa una celda, una fila, una columna, una selección de celdas que contienen uno o más bloques contiguos de celdas.

Propiedades

Nombre	Descripción
Cells	Devuelve un objeto Range que representa las celdas en el rango especificado.
Column	Devuelve el número de la primera columna de la primera área del rango especificado. Long de solo lectura.
End	Devuelve un objeto Range que representa la celda situada al final de la región que contiene el rango de origen. Equivale a presionar las teclas FIN+FLECHA ARRIBA, FIN+FLECHA ABAJO, FIN+FLECHA IZQUIERDA o FIN+FLECHA DERECHA. Objeto Range de solo lectura.
EntireColumn	Devuelve un objeto Range que representa toda la columna (o columnas) que contiene el rango especificado. Solo lectura.
EntireRow	Devuelve un objeto Range que representa toda la fila (o filas) que contiene el rango especificado. Solo lectura.
Font	especificado.
Offset	Devuelve un objeto Range que representa un rango que está desplazado del rango especificado.
Row	Devuelve el número de la primera fila de la primera área del rango. Long de solo lectura.
Value	Devuelve o establece un valor de tipo Variant que representa el valor del rango especificado.

Métodos

Nombre	Descripción
Activate	Activa una sola celda, que debe estar en la selección actual. Para seleccionar un rango de celdas, use el método Select.
Clear	Borra todo el objeto.
Copy	Copia el rango en el rango especificado o en el Portapapeles.
Cut	Corta el objeto y lo pega en el Portapapeles o en un destino especificado.
Delete	Elimina el objeto.
Find	Busca información específica en una hoja de cálculo.
Select	Selecciona el objeto.

WORKSHEETS

Propiedades

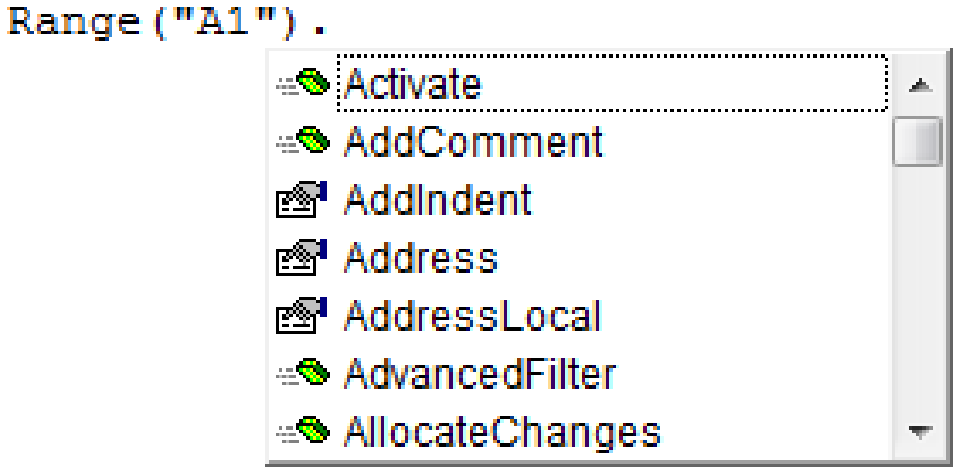
Nombre	Descripción
Index	Devuelve un valor de tipo Long que representa el número de índice del objeto dentro de una colección de objetos similares.
Name	Devuelve o establece un valor de tipo String que representa el nombre del objeto.
Visible	Devuelve o establece un valor XlSheetVisibility que determina si el objeto está visible.

Métodos

Nombre	Descripción
Activate	Convierte la hoja actual en la hoja activa.
Copy	Copia la hoja en otro lugar del libro.
Delete	Elimina el objeto.
Paste	Pega el contenido del Portapapeles en la hoja.
PrintOut	Imprime el objeto.
Select	Selecciona el objeto.

VER TODAS LAS PROPIEDADES Y MÉTODOS DE UN OBJETO

- Los objetos tienen muchas propiedades y métodos y a veces es difícil pensar que los llegaremos a memorizar todos por completo. Sin embargo, el **Editor de Visual Basic** es de gran ayuda porque justamente al momento de escribir nuestro código nos proporciona la lista completa de propiedades y métodos para un objeto.



- Esto sucede al momento de introducir el punto después del nombre del objeto. Puedes distinguir entre las propiedades y métodos porque tienen iconos diferentes. En la imagen de arriba los métodos son los que tienen el icono de color verde.

OBJETO RANGE. CELDAS Y RANGOS.

- Gran parte del trabajo de Excel se realiza sobre celdas y rangos de las hojas. Así el objeto Range esta contenido dentro de un objeto más amplio Worksheets y consta de una única celda o serie de celdas dentro de una hoja.
- Principal Propiedad del objeto Range: **Value**.

EJEMPLOS

Worksheets("Sheet1").Range("A1").Value=1

Esta instrucción introduce el valor 1 en la celda A1 de la hoja1 del libro activo.

ActiveSheet.Range("A1:B10").Value=2

En este caso se introduce el valor 2 en 20 celdas del rango especificado.

Range("A1;A3;A5;A7;A9")=4

En este caso se introduce el valor 4 en cinco celdas de rango no continuo. El punto y como sirve como operador de unión.

Worksheets("Sheet1").Range("Input").Value=1

Igual que la anterior pero destaca que en la propiedad Range también se reconoce nombres definidos en los libros.

Range("A1","B10")=2

Este ejemplo produce lo mismo que el anterior, solo que se omite la referencia a la hoja por lo que se presupone que se usa la hoja activa y también se omite la propiedad Value por lo que se utiliza la propiedad por defecto (que es Value para un objeto Range)

PROPIEDAD CELLS PARA ACCEDER A CELDAS Y RANGOS.

Otra forma de hacer referencia a un rango es utiliza la propiedad Cells.

- EJEMPLOS

Worksheets("Sheet1").Cells(1,1)=9

Esta instrucción introduce el valor 9 en la celda A1 de la Hoja1

ActiveSheet.Cells(1,1)=5

Esta instrucción introduce el valor 5 en la celda activa, la celda activa se trata como si fuera la celda A1 de la hoja activa.

ActiveSheet.Cells(3,4)=7

Esta instrucción introduce el valor 7 en la celda D3 (es decir, fila 3, columna 4)

ActiveSheet.Cells(2,1)=5

Esta instrucción introduce el valor 5 en la celda A2.

EJ. ACCEDIENDO A LAS PROPIEDADES DE LOS OBJETOS

Objeto RANGE:

- **Valor:** Range(“A1”).Value= 12
- **Formato de la Fuente Negrita:** Range(“A1”).Font.Bold = True
- **Formato de la Fuente Cursiva:** Range(“A1”).Font.Italic = True
- **Formato de la Fuente Subrayado:** Range(“A1”).Font.Underline = xlUnderlineStyleSingle
- **Tamaño de la Fuente:** Range(“A1”).Font.Size = 12
- **Color de la Fuente:** Range(“A1”).Font.Color=RGB(250,0,0)
- **Numero de Columna de una celda:** MsgBox ("N° de Columna: " & Range("A1").Column)
- **Numero de Fila de una celda:** MsgBox ("N° de Fila: " & Range("A1").Row)

Objeto WORKSHEETS:

- **Nombre de la Hoja de Calculo:** Worksheets("Hoja1").Name = "Productos“
- **Cantidad de hojas de calculo de un libro:** Msgbox(Worksheets.Count)

En este caso la propiedad Count devuelve un valor numérico, es por ello que utilizamos la función msgbox para visualizar el mismo.

EJ. IMPLEMENTACIÓN DE MÉTODOS DE OBJETOS

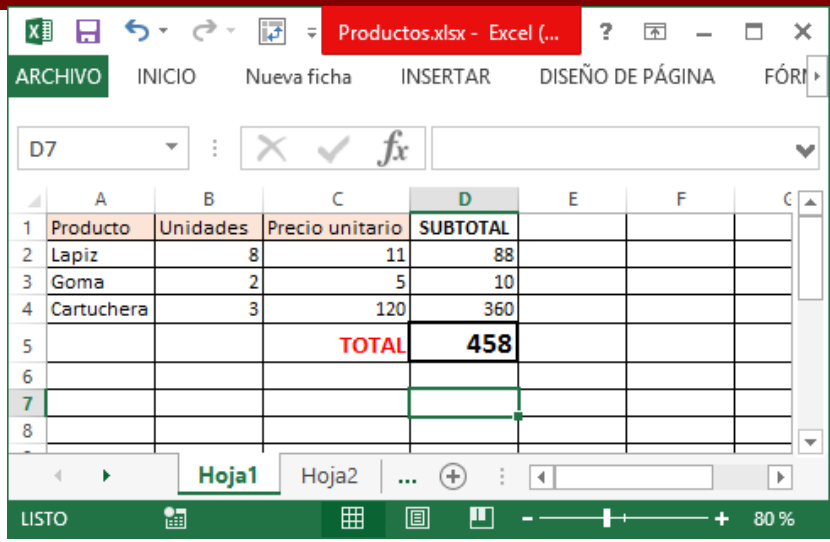
Objeto RANGE:

- **Copiar el contenido de una celda:** Worksheets("Hoja1").Range("A1").Copy Destination:=Worksheets("Hoja2").Range("A5")
- **Activar una celda:** Range("A1").Activate
- **Seleccionar un rango:** Range("A1:B2").Select
- **Eliminar una celda:** Range("A1").Delete
- **Borrar el contenido de una celda:** Range("A1").Clear

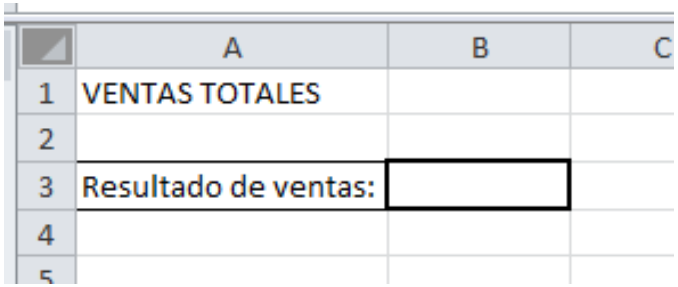
Objeto WORKSHEETS:

- **Agregar una nueva hoja de calculo:** Worksheets.Add
- **Agregar una nueva hoja de calculo con un nombre definido:** Worksheets.Add.Name = "Enero“
- **Eliminar una hoja de calculo:** Worksheets(“Hoja1”).Delete

EJERCICIO:



1. Modificar el nombre de la Hoja1 por VENTAS.
2. Modificar las unidades vendidas del producto Lápiz, a 35.
3. En la Hoja2 colocar el total de ventas y las etiquetas correspondientes (la imagen tiene la referencia de la ubicación de los datos):



4. Modificar el nombre de la Hoja2 por RESUMEN.

RESUMEN

Como hacer referencia a una Celda:

- Con RANGE:

WorkSheets(“Hoja1”).Range(“A1”).Value=“Hola”

- Con CELLS:

WorkSheets(“Hoja1”).Cells(1,1).Value=“Hola”

Como hacer referencia a un Rango:

- Con RANGE:

WorkSheets(“Hoja1”).Range(“A1:B7”).Value=“Hola”

- Con CELLS:

WorkSheets(“Hoja1”).Range(Cells(1,1),Cells(8,2)).Value=“Hola”

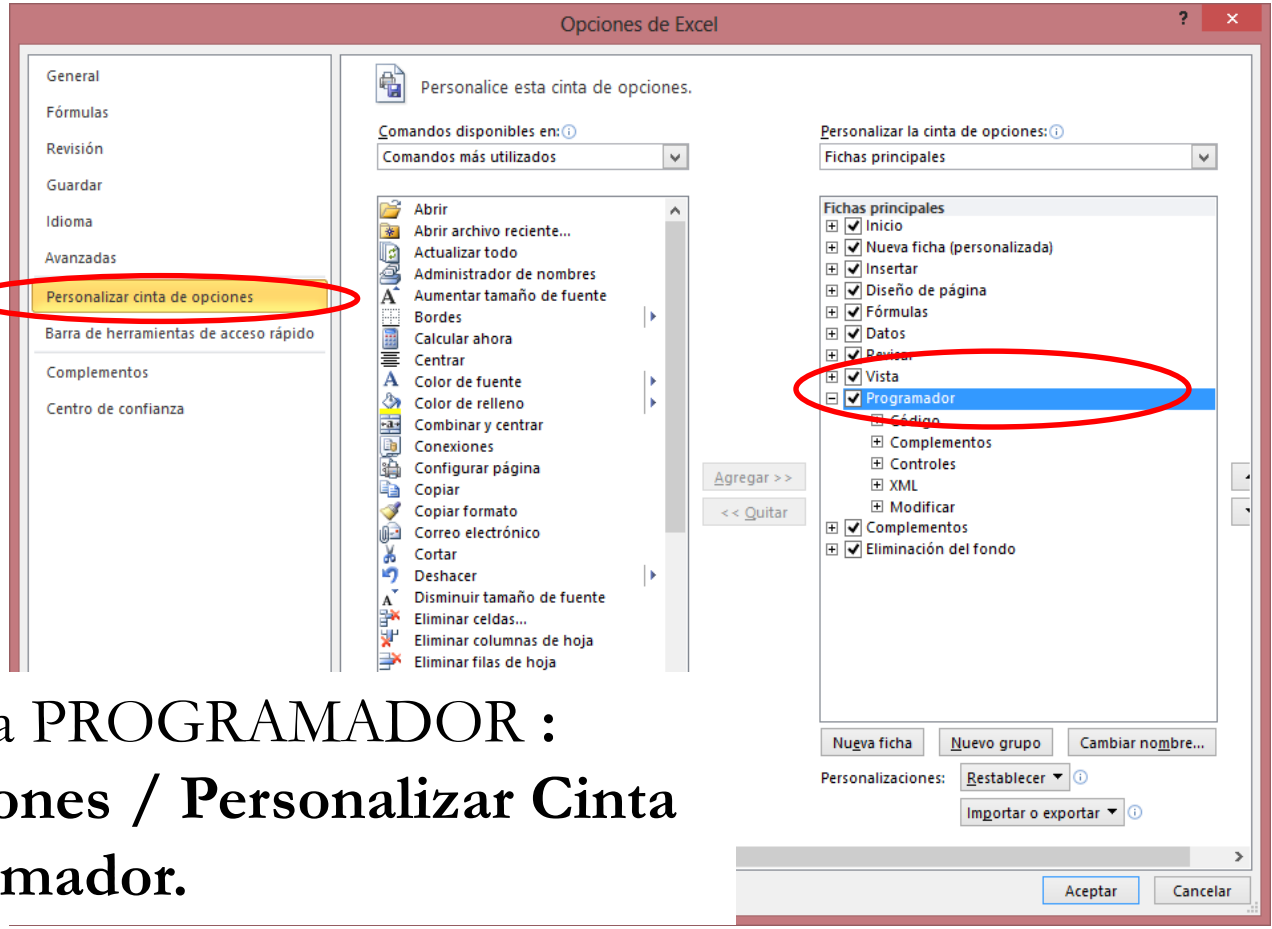
Visual Basic para Excel

1. Introducción a Programación Orientada a Objetos
2. Terminología en VBA de Microsoft Excel
3. El Núcleo de VBA
4. El Editor de Visual Basic
5. Variables, Constantes, Tipos de Datos y Expresiones
6. Funciones de Conversión de tipos.
7. Operaciones de Entrada/Salida simples

Microsoft Excel

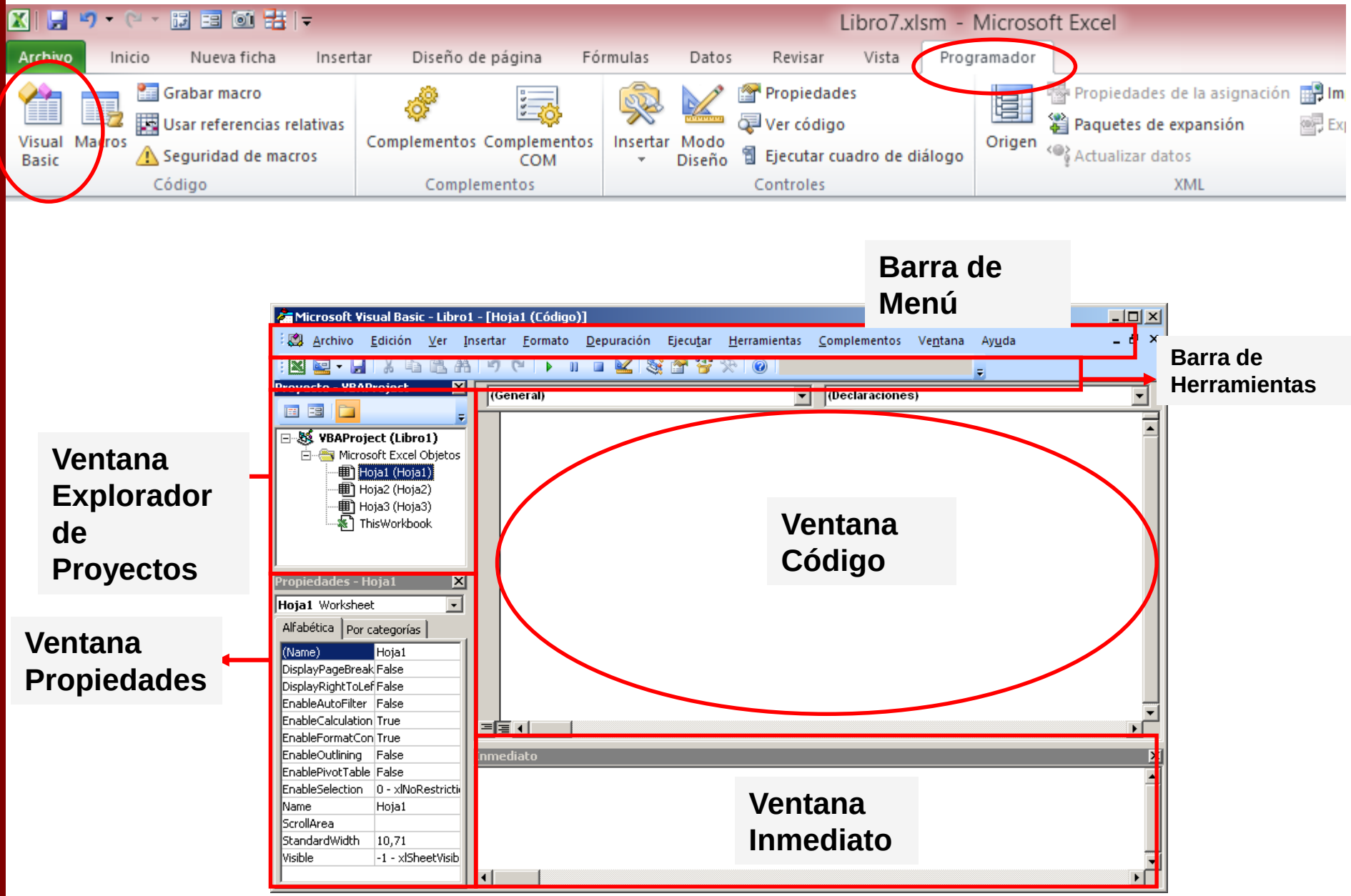
EL EDITOR DE VISUAL BASIC

- La ficha para interactuar con VBA es la FICHA PROGRAMADOR. Esta ficha es especial y no se encuentra disponible dentro de la instalación básica.



Ruta para activar la ficha PROGRAMADOR : **Ficha Archivo / Opciones / Personalizar Cinta de Opciones / Programador.**

VENTANAS DEL EDITOR DE VISUAL BASIC



ELEMENTOS DEL EDITOR DE VISUAL BASIC

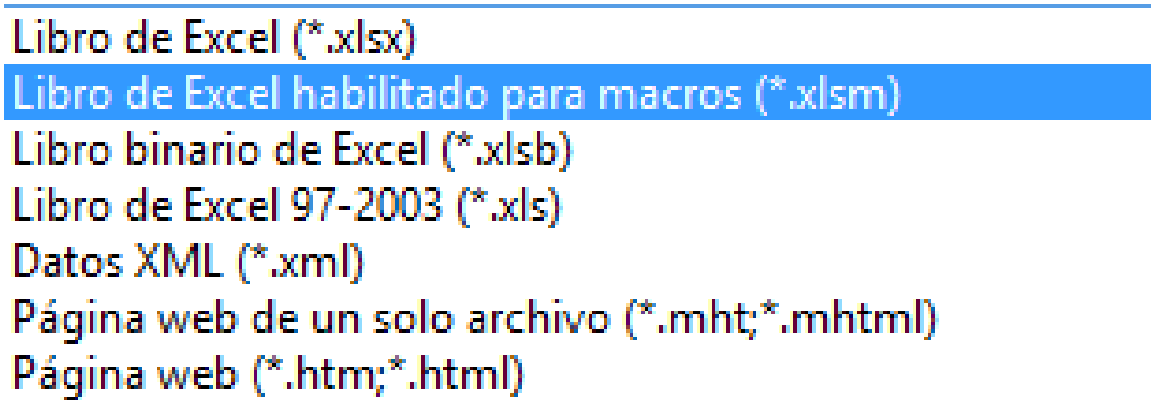
El editor de visual Basic está compuesto de varias ventanas partes :

- **Barra de menús**
- **Barras de herramientas**
- **Ventana explorador de proyectos.** Presenta un diagrama de árbol que contiene cada libro de trabajo que está actualmente abierto en excel. Cada libro de trabajo es un proyecto. Si esta ventana no esta abierta hay que presionar control+r. Para esconder la ventana, hay que hacer clic en el botón cerrar de su barra de título.
- **Ventana código.** Una ventana de código, o ventana de módulo, contiene un código VBA. Cada elemento de un proyecto tiene asociada una ventana de código. Para visualizar una ventana de código para un objeto, hay que hacer doble clic sobre el objeto en la ventana del explorador de proyectos.
- **Ventana inmediato.** Esta ventana es útil para ejecutar instrucciones de VBA directamente, probar las instrucciones y limpiar el código. Para abrir esta ventana presione control+g, para ocultarla basta hacer clic sobre el botón cerrar de su barra de título.

COMO SE GUARDAN LOS LIBROS QUE CONTIENEN CÓDIGO VBA?

NOMBRE_DEL_ARCHIVO.**XLSM**

Extensión de los libros con macros



MODULOS

- Las acciones de VBA se realizan mediante la ejecución del código VBA. El código VBA se escribe y se guarda en un **módulo** VBA.
- **Un módulo** sirve para agrupar procedimientos y funciones. Los **Procedimientos** y las **Funciones** son entidades de programación que sirven para agrupar instrucciones de código que realizan una acción concreta.
- Los módulos se guardan en un libro de trabajo de Excel pero se editan o visualizan en el editor de visual Basic.

INSERTAR O ELIMINAR UN MODULO EN EL EDITOR DE VISUAL BASIC

Insertar un nuevo módulo de VBA.

- Para insertar un nuevo módulo de VBA a un proyecto, hay que seleccionar el nombre del proyecto en la ventana explorador de proyectos y seleccionar menú insertar /modulo.
- Cuando se graba una macro, Excel inserta automáticamente un módulo VBA para contener el código grabado.

Quitar un modulo de VBA

- Hay que seleccionar el nombre del módulo en la ventana del explorador de proyectos y elegir menú archivo/quitar.

PROCEDIMIENTO SUB y FUNCTION

El código VBA que escribimos en el Editor de Visual Basic es reconocible por segmentos o grupos de códigos llamados procedimientos, existen dos de estos procedimientos:

Procedimiento Sub

- Es un conjunto de líneas de código que sirven para llevar a cabo alguna tarea específica en Excel.
- Sintaxis:

```
Sub Nombre_Procedimiento()
    .
    .
    .
End Sub
```

Procedimiento Function

- Es un conjunto de líneas de código que realizan un cálculo y que devuelve un valor.
- Sintaxis:

```
Function Nombre_Funcion ()
    .
    .
    .
End Function
```

EJEMPLOS DE PROCEDIMIENTOS SUB

Ejemplo 1:

```
Sub Primero()
    Range("a1").Value = "hola"
End sub
```

Siguiendo la jerarquía completa, el ejercicio anterior quedaría

```
Sub Primero()
    Application.Workbooks(1).Worksheets(1).Range("a1").Value = "hola"
End Sub
```

Ejemplo 2: En este segundo ejemplo simplemente ampliaremos la funcionalidad del ejemplo 1. Además de escribir "hola" en la celda A1, la pondremos en negrita y le daremos color al texto. Para ello utilizaremos las propiedades **Bold** y **color** del objeto **Font**.

```
Sub Segundo()
    Activesheet.Range("a1").Value = "hola"
    Activesheet.Range("a1").Font.Bold = true
    Activesheet.Range("a1").Font.Color = RGB(255,0,0)
End sub
```

- **True:** indica que la propiedad **bold** está activada. Si se desea desactivarla, se debe igualar al valor **false**.
- **La función RGB:** Para establecer el color de la propiedad se utiliza la función ***RGB***(Red, Green, Blue), los tres argumentos para esta función son valores del 0 a 255 que corresponden a la intensidad de los colores rojo, verde y azul respectivamente.

EJEMPLO DE PROCEDIMIENTO FUNCTION

Ejemplo 1: Función que calcula la suma de 2 valores y devuelve su resultado.

```
Function Suma(a As Integer, b As Integer) As Integer
    Suma = a + b
End Function
```

El procedimiento Tercero invoca a la Función Suma para efectuar el calculo de 2 valores pasados como parámetros de la funcion.

```
Sub Tercero()
    ActiveCell.Value = Suma(5, 4)
    Range("A5").Value = Suma(Range("A1").Value, Range("A2").Value)
End Sub
```

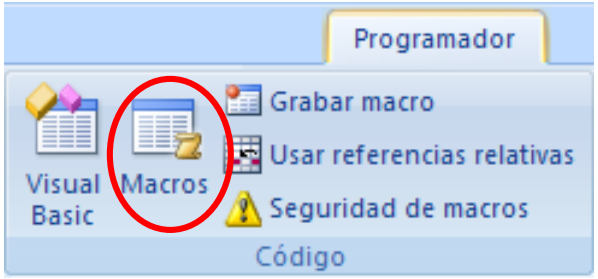
EJECUTAR UN PROCEDIMIENTO.

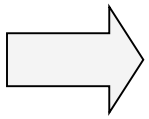
- **En el Editor de Visual Basic.**

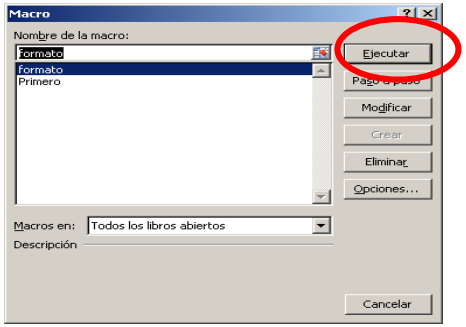
  1. Situar el cursor dentro del procedimiento.
  2. Activar la opción de la barra de menús **Ejecutar/ Ejecutar Sub/Userform**. También puede hacer clic sobre el botón o pulsar la tecla F5.

- **Para ejecutar el procedimiento desde la hoja de cálculo.** Debe estar en una hoja, no en el editor de Visual Basic

  1. Active opción macros de la ficha programador. Se despliega una ventana que muestra una lista donde estás todas las macros incluidas en el libro de trabajo.
  2. Seleccione la macro de la lista y pulse sobre el botón ejecutar.







Visual Basic para Excel

1. Introducción a Programación Orientada a Objetos
2. Terminología en VBA de Microsoft Excel
3. El Núcleo de VBA
4. El Editor de Visual Basic
5. Variables, Constantes, Tipos de Datos y Expresiones
6. Funciones de Conversión de tipos.
7. Operaciones de Entrada/Salida simples

VARIABLES

- El propósito principal de VBA es manipular datos. Algunos residen en objetos tales como rangos de hojas de cálculo. Otros se guardan en las variables que se crean.
- ***Las variables son "nombres"*** que pueden contener un valor, ya sea de tipo numérico como de cualquier otro tipo de dato.
- Una variable es una dirección "lugar" en memoria donde se guarda un valor o un objeto.
- VBA tiene algunas reglas relacionadas con los nombres de las variables:
  - ✓ Se pueden usar caracteres alfabéticos, números y algún carácter de puntuación, pero el primero de los caracteres debe ser alfabético
  - ✓ VBA no distingue entre mayúsculas y minúsculas
  - ✓ No se pueden usar espacios ni puntos
  - ✓ No se pueden incrustar en el nombre de una variable los siguientes símbolos: #, $, %,!
  - ✓ Los nombres de las variables pueden tener hasta 254 caracteres.

TIPOS DE DATOS INTEGRADOS EN VBA

(Referencia: Ayuda en línea de Visual Basic para Excel)

Tipo de datos	Tamaño De almacenamiento	Intervalo
Byte	1 byte	0 a 255
Boolean	2 bytes	True o False
Integer	2 bytes	-32.768 a 32.767
Long(entero largo)	4 bytes	-2.147.483.648 a 2.147.483.647
Single (coma flotante/ precisión simple)	4 bytes	-3,402823E38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
Double (coma flotante/ precisión doble)	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
Currency (entero a escala)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807
Decimal	14 bytes	+/-79.228.162.514.264.337.593.543.950.335 sin punto decimal; +/-7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es+/- 0,0000000000000000000000000001
Date	8 bytes	1 de enero de 100 a 31 de diciembre de 9999
Object	4 bytes	Cualquier referencia a tipo Object
String (longitud variable)	10 bytes + longitud de la cadena	Desde 0 a 2.000 millones
String(longitud fija)	Longitud de la cadena	Desde 1 a 65.400 aproximadamente
Variant(con números)	16 bytes	Cualquier valor numérico hasta el intervalo de un tipo Double
Variant(con caracteres)	22 bytes + longitud de cadena	El mismo intervalo que para un tipo String de longitud variable
Definido por el usuario (utilizando Type)	Número requerido por los elementos	El intervalo de cada elemento es el mismo que el intervalo de su tipo de datos.

DECLARACIÓN DE VARIABLES

Tenemos que decirle a VBA que "reserve" un espacio en memoria para poder guardar un valor u objeto, esto se consigue mediante la declaración de variables.

- Sintaxis para declarar una variable

  DIM NomVariable AS Tipo.

Siendo:

- **NomVariable:** el nombre que se asigna a la variable y
- **Tipo:** el tipo de dato (números, texto, fecha, boleanos,...) que se guardará en la variable.

- Ejemplos:

  DIM Contador As Integer

  DIM NombreyApellido As String

  DIM Pago As Boolean

DECLARACIÓN DE MÚLTIPLES VARIABLES

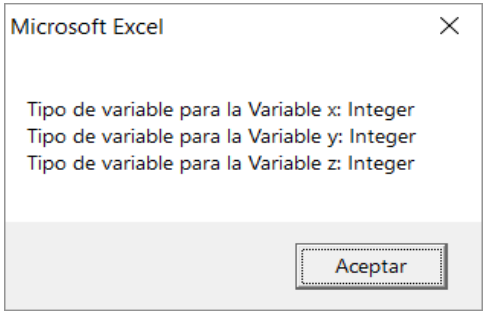
Cuando disponemos de varias variables, las podemos declarar en el proceso como:

```
Sub MiVariable()
Dim x As Integer
Dim y As Integer
Dim z As Integer

'Contenido del código del procedimiento
MsgBox ("Tipo de variable para la Variable x: " & TypeName(x) & Chr(13) & _
"Tipo de variable para la Variable y: " & TypeName(y) & Chr(13) & _
"Tipo de variable para la Variable z: " & TypeName(z))

End Sub
```

Resultado del MsgBox



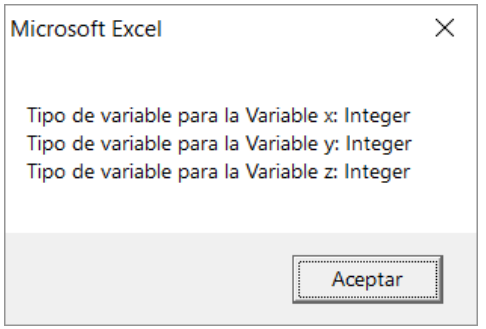
O también como:

```
Sub MiVariable2()
Dim x As Integer, y As Integer, z As Integer

'Contenido del código del procedimiento
MsgBox ("Tipo de variable para la Variable x: " & TypeName(x) & Chr(13) & _
"Tipo de variable para la Variable y: " & TypeName(y) & Chr(13) & _
"Tipo de variable para la Variable z: " & TypeName(z))

End Sub
```

Resultado del MsgBox



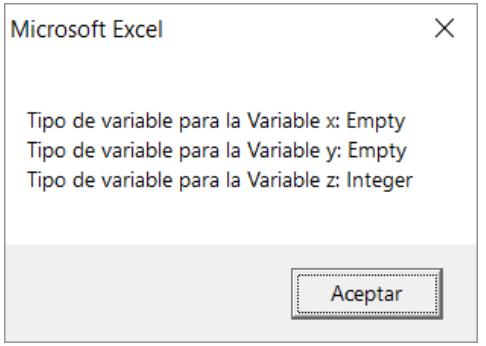
- **Esto no es válido:**

```
Sub MiVariable3()
Dim x, y, z As Integer
' Esto no es valido, z es la única variable
' definida como Integer, x e y se declaran como
' de tipo Variant.

'Contenido del código del procedimiento
MsgBox ("Tipo de variable para la Variable x: " & TypeName(x) & Chr(13) & _
"Tipo de variable para la Variable y: " & TypeName(y) & Chr(13) & _
"Tipo de variable para la Variable z: " & TypeName(z))

End Sub
```

Resultado del MsgBox



ASIGNACIÓN DE VALORES A UNA VARIABLE

- Cuando se declara una variable, VBA le asigna un valor por defecto, como lo muestra la siguiente tabla:

Valores por Defecto	
Tipo de Dato	**Valor por Defecto**
Boolean	False
Integer	0
Long	0
Single	0
Double	0
Currency	8
Date	8
String	""

EXPRESIONES DE ASIGNACIÓN

- Para asignar un valor a una variable, es decir, para especificar que valor se va almacenar en ese espacio de memoria, se utiliza el signo igual (=) como operador de asignación.
- Una expresión de asignación es una instrucción de VBA que realiza evaluaciones matemáticas y asigna el resultado a una variable o a un objeto.

```
X = 1
X = x + 1
X = (y * 2) / (z * 2)
Range("año"). Value = 1995
```

EJEMPLO

```
Sub Entrar_Valor()

Dim Texto As String

    ' Chr(13) sirve para que el mensaje se muestre en dos Líneas

    Texto = InputBox("Introducir un texto " & Chr(13) & "Para la casilla A1", "Entrada de datos")

    ActiveSheet.Range("A1").Value = Texto

End Sub
```

- Este ejemplo también se puede hacer sin variables.

```
Sub Entrar_Valor()

    ActiveSheet.Range("A1").Value = InputBox("Introducir un texto " & Chr(13) & "Para la casilla A1", "Entrada de datos")

End Sub
```

VARIABLES DE OBJETOS

Una variable de objeto sirve para hacer referencia a un objeto, esto significa que podremos acceder a las propiedades de un objeto (como puede ser un rango o una hoja de cálculo) e invocar a sus métodos a través de la variable en lugar de hacerlo directamente a través del objeto. La variable se declara como:

```
Sub MiVariableObjeto()
   Dim NombreVariable As Objeto

    'código del procedimiento

End Sub
```

Uso de Variables Objeto (Set)

Uso de la instrucción SET para asignar valores a una variable tipo objeto

Sintaxis

```
Set NombreVariable = NombreObjeto
```

Ejemplo:

```
Sub MiVariableObjeto()
   Dim MiCelda As Range
   Set MiCelda = Worksheets("Hoja1").Range("A1")
   MiCelda.Value=5
End Sub
```

LA SENTENCIA OPTION EXPLICIT.

SOBRE LA NECESIDAD DE DECLARAR VARIABLES

- En Visual Basic no es necesario declarar las variables, en los ejemplos anteriores se hubiera podido prescindir de las líneas

Dim texto as String

- Aunque podemos omitir la declaración de variables y utilizar cualquier nombre con cualquier tipo de dato, es necesario, por no decir que obligatorio, declarar las variables, no solo por ser una buena práctica sino para evitar inconvenientes en distintos momentos de la creación del código.
- Si no declara las variables al principio del procedimiento ocurrirán dos cosas.
  - ✓ Primero, las variables no declaradas son asumidas como tipo Variant (este es un tipo de datos que puede almacenar cualquier valor, número, fechas, texto, etc). El tipo Variant ocupa entre 16 y 22 bytes (para guardar, por ej, la edad de alguien, no son necesarios tantos bytes);
  - ✓ Segundo, reducirá considerablemente la legibilidad de sus procedimientos ya que las variables las irá colocando a medida que las necesite, esto, a la larga complicará la corrección o modificación del procedimiento.
- La sentencia **Option Explicit** al principio del módulo fuerza a que se declaren todas las variables. Si al ejecutar el programa, se encuentra alguna variable sin declarar se producirá un error y no se podrá ejecutar el programa hasta que se declare.
- ***Para activar la sentencia Option Explicit:*** **Herramientas -> Opciones -> Editor - Requerir Declaración de Variables**

EJEMPLO

```
Sub Entrar_Valor()
    Texto = InputBox("Introducir un texto " & Chr(13) &
    "Para la casilla A1", "Entrada de datos")
    ActiveSheet.Range("A1").Value = Testo
End Sub
```

El siguiente código presenta errores al ejecutarse???

EJEMPLO

```
Option Explicit

Sub Entrar_Valor()

    Dim Texto As String

    Texto = InputBox("Introducir un texto " & Chr(13) & "Para la casilla
    A1", "Entrada de datos")

    ActiveSheet.Range("A1").Value = Testo

End Sub
```

El siguiente código presenta errores al ejecutarse???

ÁMBITO DE LAS VARIABLES

- Un libro de Excel puede tener cualquier número de módulos VBA y estos módulo pueden tener cualquier número de procedimientos Sub y Procedimientos Functions.
- El ámbito de una variable determina el módulo y el procedimiento en el que se puede usar una variable.

Ámbito de Variables - Declaración	
Ámbito (Scope)	**Declaración (Palabra Reseravada)**
A Nivel de Procedimiento	Con las palabras reservadas **Dim** o **Static** dentro del procedimiento deseado
A Nivel de Módulo	Declara la Variablecon **Dim** antes del primer **procedimiento** en el módulo
En todo el proyeto	Utilizando la palabra reservada **Public** antes del primer procedimiento en un módulo

- Normalmente cuando un procedimiento termina todas la variables en el procedimiento se resetean.
- **Una variable declarada con la palabra reservada Static es un caso especial que retiene el valor almacenado, incluso cuando el procedimiento ha terminado. Las variables con Static se declaran a nivel de procedimiento.**

VARIABLES DEFINIDAS A NIVEL DE PROCEDIMIENTO

- Variables Locales y Variables Estáticas

Variables Locales	Variables Static
Variable declarada dentro de un procedimiento. Solo se usan dentro del procedimiento y cuando este acaba, la variable deja de existir y Excel libera la memoria. **Sub Ejemplo()** **Dim InteresAnual As Long** **'código del procedimiento** **End Sub**	Las variables estáticas son un caso especial. Se declaran a nivel de procedimiento y retienen su valor después de que el procedimiento finaliza. **Sub Ejemplo()** **Static InteresAnual As Long** **'código del procedimiento** **End Sub**

VARIABLES DEFINIDAS A NIVEL DE MODULO Y A NIVEL DE PROYECTO

- Variables a Nivel de Modulo y Variables Publicas

Variables a Nivel de Modulo	Variables Públicas
Algunas veces se deseara que una variable esté disponible para todos los procedimientos de un módulo. Para ello, se declara la variable antes del primer procedimiento del módulo (fuera de cualquier procedimiento o función). **Dim InteresAnual As Long** **Sub Ejemplo1()** **'código del procedimiento 1** **End Sub** **Sub Ejemplo2()** **'código del procedimiento 2** **End Sub**	Para que una variable esté disponible para todos los procedimientos de un proyecto de VBA, se declara la variable a nivel de modulo con el uso de la palabra Public. **Public InteresAnual As Long** **Sub Ejemplo()** **'código del procedimiento** **End Sub**

EJEMPLO: CÓDIGO PARA VALIDAR EL ÁMBITO DE UNA VARIABLE

```
Public ContPublica As Integer
Sub Variables()
Dim ContLocal As Integer
Static ContEstatica As Integer

Dim Casilla_Inicial As String
Dim Fila As Integer, Columna As Integer

Casilla_Inicial = InputBox("Introducir la casilla Inicial : ", "Casilla Inicial")
ActiveSheet.Range(Casilla_Inicial).Activate
Fila = ActiveCell.Row
Columna = ActiveCell.Column

Cells(Fila, Columna) = ContPublica
ContPublica = ContPublica + 1
Cells(Fila + 1, Columna) = ContPublica

Cells(Fila + 2, Columna) = ContLocal
ContLocal = ContLocal + 1
Cells(Fila + 3, Columna) = ContLocal

Cells(Fila + 4, Columna) = ContEstatica
ContEstatica = ContEstatica + 1
Cells(Fila + 5, Columna) = ContEstatica

End Sub
```

MÉTODOS PARA LIMPIAR UNA VARIABLE

- En el Editor de Visual Basic: **En la Barra de Menú Opción Ejecutar/Restablecer.**
- Incluir la palabra reservada **End** en cualquier lugar del código.

De otra forma únicamente las variables declaradas a nivel de procedimiento serán receteadas en memoria después que se termina la ejecución del código VB.

Las Variables Static, a nivel de módulo y las Variables Public son retenidas en memoria de una ejecución a otra.

TIPO DE OPERADORES

Operadores aritméticos

+ Suma, - resta, * multiplicación, / división, \ división entera, mod resto, ^ exponencial, & concatenación

Operadores comparativos

= Igual, < menor, > mayor, <= menor o igual, >= mayor o igual, <> distinto

Operadores lógicos

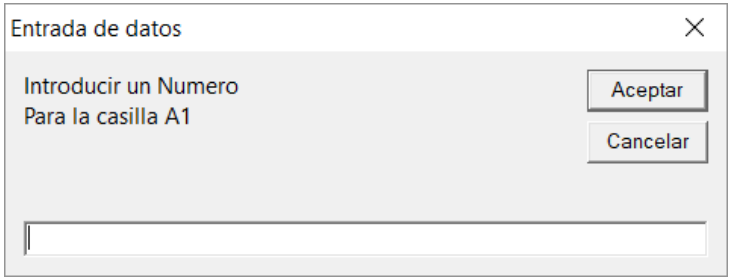
NOT (negación lógica, AND (conjunción lógica), OR (disyunción lógica), XOR (exclusión lógica), EQV (equivalencia en dos expresiones), IMP (implicación lógica)

ERRORES TÍPICOS EN VBA EN LA ASIGNACIÓN DE VARIABLES (ERROR 6 Y 13)

Ejecutar el siguiente código con los siguientes valores como entrada de datos:

Caso 1: Ingresar un texto.

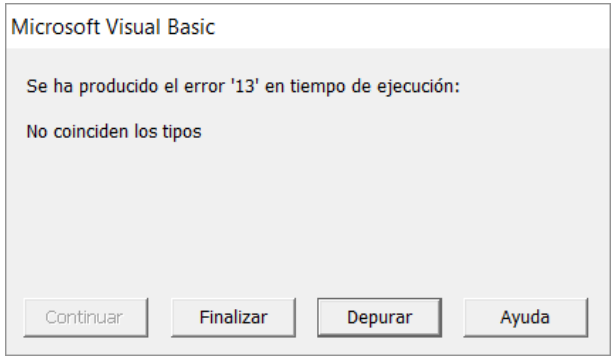
Caso 2: Ingresar un valor mayor a 32767.



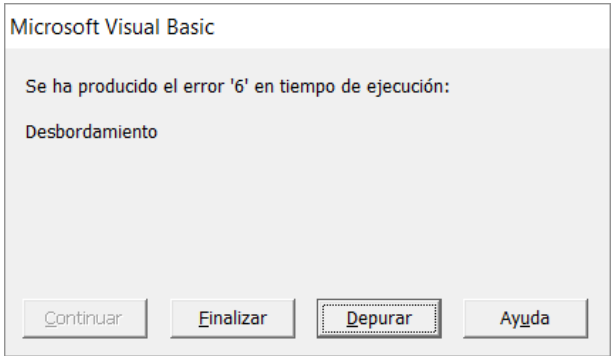
Codigo

```
Sub Entrar_Valor1()
Dim Numero As Integer
Numero = InputBox("Introducir un Numero " & Chr(13) & "Para la casilla A1", "Entrada de datos")
ActiveSheet.Range("A1").Value = Numero
End Sub
```

Resultado Caso 1:

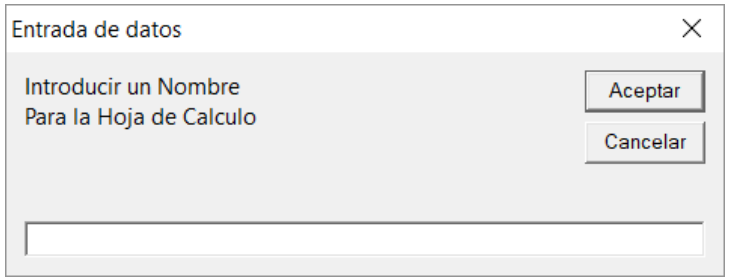


Resultado Caso 2:



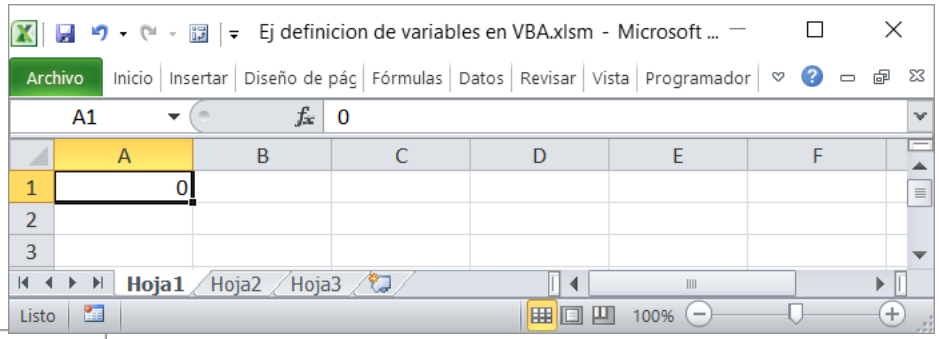
ERRORES TÍPICOS EN VBA EN LA ASIGNACIÓN DE VARIABLES (ERROR 9)

Ejecutar el siguiente código:

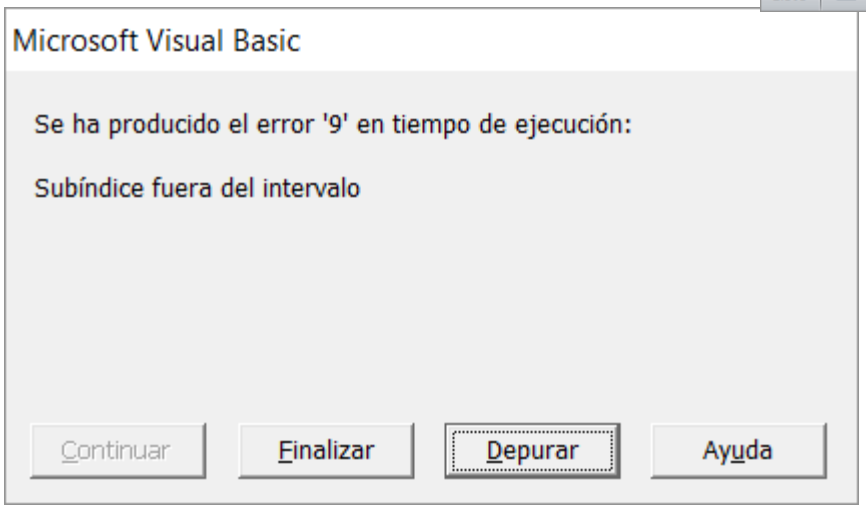


Codigo

```
Sub Cambiar_Nombre_Hoja()
Dim Texto As String
Texto = InputBox("Introducir un Nombre " & Chr(13) & "Para la Hoja de Calculo", "Entrada de datos")
Worksheets("Hoja4").Name = Texto
End Sub
```



Resultado:



El error se produce porque la Hoja4 a la que se hace referencia No Existe.

FUNCIONES DE CONVERSIÓN DE TIPOS.

- Val(cadena). Convierte la cadena a un valor numérico.
- Str(número). Convierte el número a una expresión cadena.

Ejemplo

```
Option Explicit
Sub Sumar()
    Dim Numero1 As Integer
    Dim Numero2 As Integer
    Numero1 = Val(InputBox("Entrar el primer valor", "Entrada de datos"))
    Numero2 = Val(InputBox("Entrar el segundo valor", "Entrada de datos"))
    ActiveSheet.Range("A1").Value = Numero1 + Numero2
End Sub
```

La función Val(Dato String), convierte una cadena de caracteres a valor numérico. Si la cadena a convertir contiene algún carácter no numérico devuelve 0. Así, si al pedir un valor se teclea "Hola", la función Val, devolverá un cero.

Visual Basic para Excel

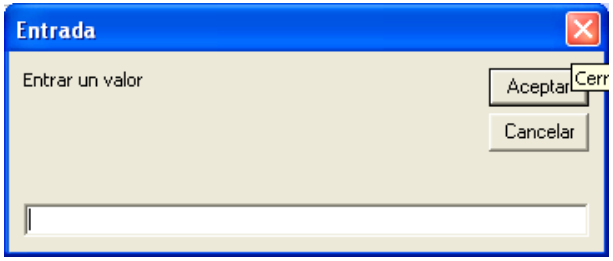
1. Introducción a Programación Orientada a Objetos
2. Terminología en VBA de Microsoft Excel
3. El Núcleo de VBA
4. El Editor de Visual Basic
5. Variables, Constantes, Tipos de Datos y Expresiones
6. Funciones de Conversión de tipos.
7. Operaciones de Entrada/Salida simples

Operaciones de Entrada/Salida simples

LA FUNCIÓN INPUTBOX.

- Esta función muestra una ventana para que el usuario pueda teclear datos. Cuando se pulsa sobre aceptar, los datos entrados pasan a la variable a la que se ha igualado la función. Ejemplo.

Texto = Inputbox("entrar un valor", "entrada").



Sintaxis de inputbox.

Inputbox (mensaje, título, valor por defecto).

- **Mensaje** : es el mensaje que se muestra en la ventana.
- **Título** : es el título para la ventana inputbox. Es un parámetro opcional.
- **Valor por defecto**: es el valor que mostrará por defecto el cuadro donde el usuario entra el valor.

Operaciones de Entrada/Salida simples

LA FUNCIÓN MSGBOX.

Esta función muestra un mensaje en un cuadro de diálogo hasta que el usuario pulse un botón. La función devuelve un dato tipo Integer en función del botón pulsado por el usuario.

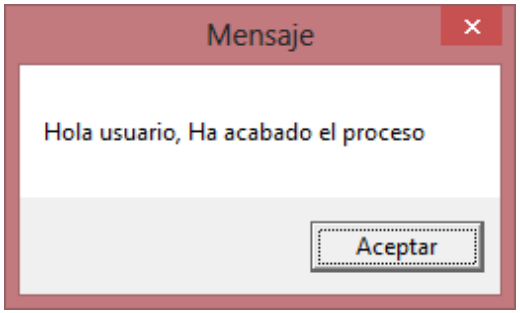
Sintaxis de msgbox.

Msgbox(mensaje, botones, título)

- Msgbox (“mensaje”)
- Nombre_variable = MsgBox (“mensaje”, botones, “título”)

Ej:

X= MsgBox ("hola usuario, ha acabado el proceso", Vbok, “mensaje")



Constante	Valor	Descripción
VbOK	1	Aceptar
VbCancel	2	Cancelar
VbAbort	3	Anular
VbRetry	4	Reintentar
VbIgnore	5	Ignorar
VbYes	6	Sí
VbNo	7	No

DIFERENTES MODOS DE REFERIRSE A LIBROS, HOJAS Y CELDAS:

Workbook : Libro de trabajo.	
ActiveWorkbook	Libro activo
Workbooks(2)	El segundo libro abierto
Workbooks("Libro1.xls")	Llamada al libro de nombre Libro1
Workbooks(milibro)	Llamada a un libro cuyo nombre se encuentra en una variable llamada 'milibro' (*) *-Si el nombre del libro se encuenta en una variable, NO lleva comillas Previamente asignamos nombre, por ej: milibro=ActiveWorkbook.name

WorkSheet : Hoja de trabajo	
ActiveSheet	Hoja activa
WorkSheets("Enero")	Hoja de nombre 'Enero'
WorkSheets(1)	Número de hoja según el orden de las pestañas.
Sheets("Enero")	Hoja de nombre 'Enero'
Sheets(3)	Número de hoja según el orden de las pestañas.
[Hoja2]	La 2da hoja según orden de las pestañas

Range o Cells : rango o celda	
Activecell	la celda activa
Range("A2")	la celda A2
Cells(2,1)	la celda de fila 2 y columna 1 = A2 .
Cells(3,"D")	la celda de fila 3 y col D = D3 *-Nótese que mientras en Range se introduce la celda en el orden Col,Fila, en Cells es a la inversa: Cells(fila,col)
Range("A5:B10")	rango de celdas desde A5 hasta B10 inclusive
Range("E:E")	columna E
Range("2:2")	fila 2
[A3]	la celda A3
Range("A" & fila)	celda de la col A y fila según valor de variable