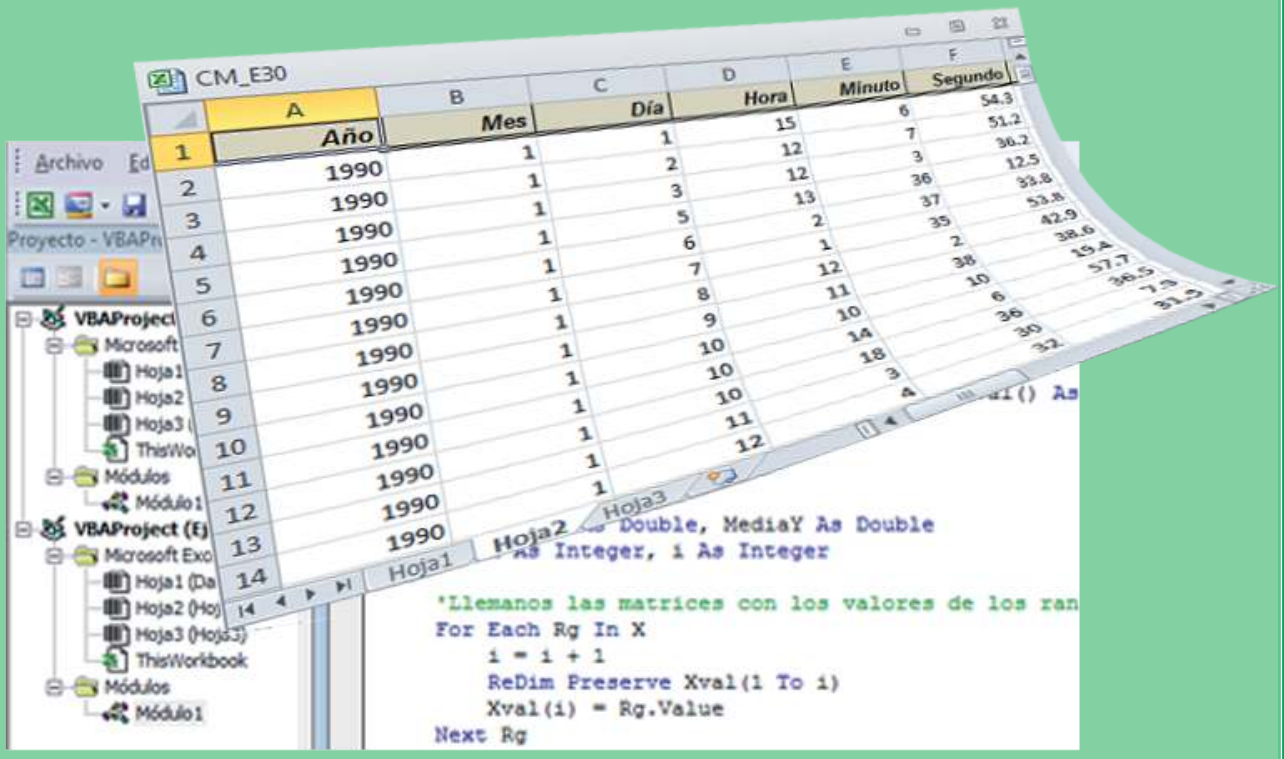
Postgrado oficial
Ciencias de la Tierra

Introducción a las Macros para Microsoft Excel®



J. V. Pérez Peña

Libro-Manual del curso impartido dentro del programa oficial de
DOCTORADO EN CIENCIAS DE LA TIERRA

Datos de catalogación bibliográfica

José Vicente Pérez Peña

Introducción a las Macros para Microsoft Excel

Granada, 2012.

ISBN: 978-84-615-7245-8

Materias BIC: UFBC UMN UMP

Formato Electrónico. Páginas: 76

José Vicente Pérez Peña

Introducción a las Macros para Microsoft Excel®

1ª Edición: Febrero, 2012
I.S.B.N.: 978-84-615-7245-8

Diseño de la portada; Pérez Peña, J.V.

Edición; Grupo RMN 148 (Junta de Andalucía)
Con la colaboración de los proyectos;
CGL2008-03249 y CGL2011-29920 del Ministerio de Ciencia e Innovación.

Como citar este libro:

Pérez-Peña, J.V. (2012). Introducción a las Macros para Microsoft Excel®. Grupo RMN 148 (Junta de. Andalucía), Granada, España.

Prólogo

El presente libro es un intento de sintetizar una temática compleja como son las macros y el lenguaje de programación Visual Basic for Applications (VBA) integrado en gran parte de las aplicaciones de escritorio de Microsoft Windows ®. Este libro-manual es fruto del curso de postgrado "Macros para Ms Excel" impartido dentro del programa de doctorado de Ciencias de la Tierra de la Universidad de Granada, y nace como la necesidad de plasmar los contenidos de dicho curso en un único volumen que pueda servir de referencia bibliográfica a los alumnos del mismo. Debido a la temática de este curso, es posible encontrar manuales teóricos en ambos extremos; sumamente detallados (dirigidos prácticamente a desarrolladores), o excesivamente escuetos (en los que no es posible entender lo que se está realizando). Este es un puente entre ambos extremos; ofreciendo una base teórica precisa y actualizada, así como ejemplos de uso práctico.

El curso de postgrado impartido en la Universidad de Granada, pese a contar hasta la fecha con solamente dos ediciones, cuenta con un grado de aceptación muy elevado entre sus estudiantes, que entienden este tipo de formación práctica como parte esencial para su formación.

A pesar de que el presente manual esta específicamente enfocado a las macros para Ms Excel ®, haciendo uso de los objetos COM de Microsoft, el lector encontrará una introducción a la programación en leguaje VBA, así como una presentación a la programación orientada a objetos que le podrá servir a modo de base para realizar script y aplicaciones para cualquier otro tipo de software que admita el lenguaje VBA.
Este libro se completa con los archivos de los ejercicios resueltos que se encuentran en el CD adjunto.

Agradecimientos

Me gustaría agradecer especialmente al coordinador del programa de Doctorado de Ciencias de la Tierra de la Universidad de Granada, Antonio García Casco, por haber hecho posible la impartición del curso "Macros para Ms Excel". Así mismo agradecer al Departamento de Geodinámica de la Universidad de Granada por facilitar sus instalaciones para impartir las clases. También mi más sincero agradecimiento a los alumnos tanto de la primera como de la segunda edición del citado curso, por animarme a realizar sucesivas ediciones y la elaboración de un manual que sintetizase la materia vista en el curso.

Índice de contenidos:

Parte I: INTRODUCCIÓN A LAS MACROS

0. INTRODUCCIÓN **10**

Parte II: CONCEPTOS BÁSICOS DE PROGRAMACIÓN

1. VARIABLES **18**

1.1. Concepto de variable 18

1.2. Declaración de variables en VBA 18

1.3. Tipos de variables primitivos en VBA 19

1.4. Operadores básicos 20

1.5. Ámbito de las variables 20

1.6. Ejercicios de declaración de variables 21

2. INSTRUCCIONES **22**

2.1. Instrucción lógica If …Then 22

2.2. Bucle For …Next 25

2.3. Bucle Do … Loop 27

2.4. Sentencia SelectCase … 30

3. MATRICES (ARRAYS) **32**

3.1. Concepto de matriz 32

3.2. Declaración de matrices en VBA (Dinámicas – Estáticas) 32

Parte III: PROGRAMACIÓN ORIENTADA A OBJETOS. OBJETOS DE EXCEL

4. INTRODUCCIÓN A LOS OBJETOS **38**

4.1 La programación orientada a objetos 38

4.2. Concepto de objeto. Propiedades – Métodos - Eventos 38

4.3. Declaración de una variable objeto 39

4.4. Objetos propios de Excel 40

5. MODULOS Y FORMULARIOS **43**

5.1. Concepto de Módulo 43

5.2. Tipos de Módulos 43

5.3. Creación de módulos en VBA 43

5.4. Trabajando con formularios 44

6. PROCEDIMIENTOS Y FUNCIONES **52**
6.1. Concepto de Procedimiento 52
6.2. Concepto de Función 55
7. TRABAJANDO CON OBJETOS PROPIOS DE EXCEL **57**
7.1. Objeto Workbook 58
7.2. Objeto Worksheet 61
7.3 Objeto Range 64

PARTE I:
INTRODUCCIÓN A LAS MACROS

0. INTRODUCCIÓN

¿Qué es una macro?

La palabra macro es una abreviatura de "macroinstrucción", es decir un conjunto de instrucciones que se ejecutan secuencialmente. Hablando más coloquialmente, una macro es un pequeño programa que realiza una tarea específica a través de una serie de instrucciones. Este pequeño programa se encuentra dentro de la aplicación principal, y por lo tanto puede acceder a todas las funcionalidades de la misma. Es decir, una macro dentro de MS Word podrá acceder a los tipos de letra, párrafo, tablas, … de Word; una macro dentro de Excel podrá acceder a los libros, hojas, formulas, celdas, … de la aplicación principal.

¿Cómo se crea una macro?

Una macro está compuesta por una serie de instrucciones que se ejecutan secuencialmente para realizar una tarea determinada. Estas instrucciones están escritas en lenguaje de programación, por lo que para crear una macro deberemos de escribir el código necesario para realizar las instrucciones deseadas. Las macros pueden aumentar muy considerablemente las posibilidades de muchas aplicaciones y adaptarlas de una manera mucho más eficiente a necesidades específicas de sus usuarios.

En muchas aplicaciones que trabajan en entorno Windows (Microsoft Office entre otras muchas) está disponible el editor de *"Visual Basic for Applications"* (*VBA*) para crear macros. El lenguaje VBA es un subconjunto casi completo del lenguaje Visual Basic 6, casi cualquier cosa que se pueda programar en este lenguaje, se podrá programar también dentro de una aplicación que tenga el editor de VBA integrado. Es por este motivo que las nuevas versiones de Office son cada vez más restrictivas a la hora de ejecutar macros de autores desconocidos. Con una macro podríamos hacer que un simple documento de Word se comportara como un virus simplemente ejecutando las instrucciones adecuadas.

¿Qué se puede hacer con una macro en Excel?

Como he indicado anteriormente, las aplicaciones de la familia de software de Microsoft Office tienen un editor de VBA integrado e independiente. MS Excel tiene también este editor, con lo que podemos aumentar considerablemente sus funcionalidades (ya de por si amplias). Algunas de estas funcionalidades serian;

- ***Automatización de tareas y procesos que involucran muchos pasos***

Imaginemos una tarea rutinaria con Excel que hagamos a menudo y que involucre varios pasos (incluso decenas). Con una macro adecuada podríamos reducir esta rutina a un solo clic!

- ***Creación de nuevas funciones a medida***

Excel trae incorporadas unas 330 funciones estándar, sin embargo suele suceder que justo la función que necesitamos no existe. Mediante el uso de macros podemos programar funciones a medida y que hagan exactamente lo que nosotros queremos. Estas nuevas funciones se comportarán igual que las ya incorporadas por Excel (aparecerán en el menú de funciones en la categoría que nosotros indiquemos, tendrán sus respectivos argumentos, etc.).

- ***Creación de nuevos comandos, complementos y menús.***

Los complementos Excel también están creados con macros. Si vamos a menú Herramientas > Complementos vemos una lista de los que están instalados en nuestro Excel. Su utilidad reside en agregar alguna funcionalidad extra al Excel.

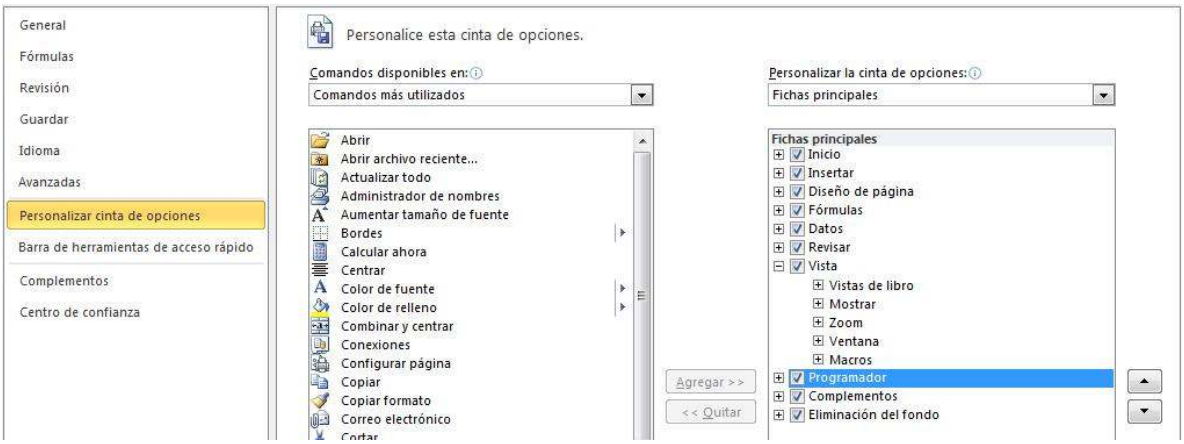
- ***Creación de completas aplicaciones a medida.***

Las macros nos pueden permitirán construir complejas y elegantes aplicaciones para cualquier uso que queramos. ¿El límite?, Solo la imaginación.

El editor de VBA (Creación de nuestra primera macro)

El editor de VBA viene integrado en numerosas aplicaciones de MS Office. A partir de la versión 2007 de MS Excel tenemos dos opciones, para acceder a él tenemos dos opciones; en la cinta de ***Programador – Visual Basic***, o mediante el teclado **Alt + F11**.

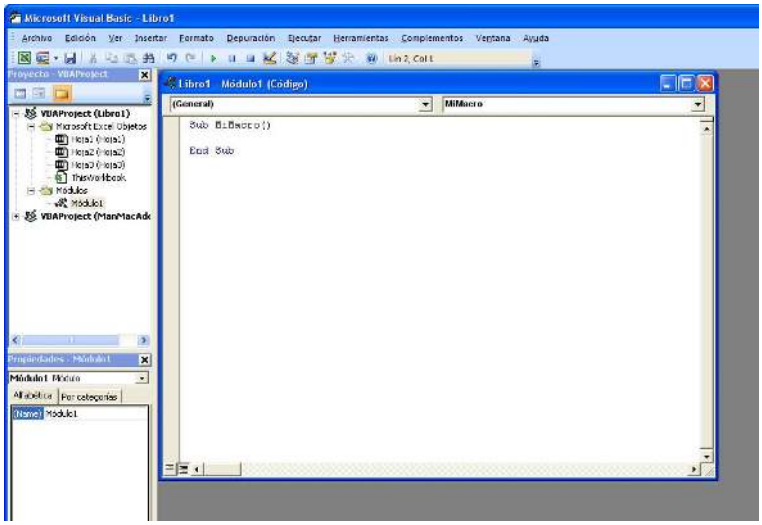
> En las versiones de Excel 2007 y 2010, debemos de activar la ficha de ***programador***, vamos a Opciones de Excel (botón derecho en lugar vacío en barra de ***Menú***) y vamos a la opción de ***Personalizar cinta de opciones***, y en el cuadro **fichas principales** activamos la casilla ***Mostrar ficha de programador*** *en la cinta de opciones.*



En versiones anteriores, iremos a Menú – Herramientas – Macros

Para entender mejor el editor vamos a crear una macro en blanco. Para hacer esto vamos a la pestaña Vista – Macros, aunque también podemos entrar desde la ficha de Programador – Macros, y aún tenemos otro modo de entrar con **Alt+F11**. En la ventana nos aparecerán las macros que hemos creado en el documento. Vemos que no aparece ninguna macro en el cuadro inferior, porque aún no hemos creado ninguna. Para crear una nueva macro escribimos en el cuadro debajo de *"Nombre de la macro"*. Le ponemos de nombre "*MiMacro*" y le damos a crear.

Haciendo esto hemos creado nuestra primera macro, y hemos entrado en el editor de VBA.

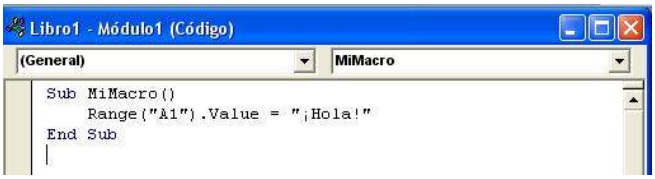


El editor contiene 3 ventanas principales: la Ventana Proyecto (parte izquierda superior), la Ventana de Código (parte derecha) y la Ventana Propiedades (parte izquierda inferior). Además, como muchos otros programas de Windows, también tiene un Menú y debajo una barra de herramientas.

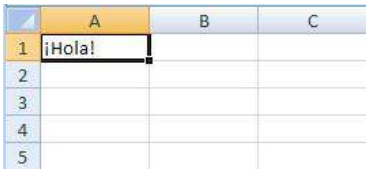
- ✓ ***La Ventana Proyecto***; esta ventana muestra los libros (*.xls) y los complementos abiertos. Usualmente verás nombres del tipo "VBAProject" y entre paréntesis el nombre del archivo o complemento Excel. Veamos el caso de VBAProject (Libro1); simplemente significa que tienes abierto un libro Excel llamado Libro 1. Además tenemos 2 sub-carpetas; Microsoft Excel Objetos (con Hoja1, Hoja2, Hoja3, y ThisWorbook) y Módulos (En esta última vemos Módulo1, que es donde hemos creado nuestra primera macro).
- ✓ **Ventana de Código:** esta es el lugar donde escribiremos el código propiamente dicho de las macros. Como vemos, en la barra de título de esta ventana pone "Libro1 – Módulo1 (Código)". Esto quiere decir que la ventana de código corresponde a un módulo estándar llamado Módulo1 que se ha creado en dentro del Libro1 (libro que tenemos actualmente abierto). Dentro de esta ventana de cogido vemos que el programa ha escrito **Sub** MiMacro() … **End Sub**. Lo que hay entre estas dos instrucciones es la macro que acabamos de crear. Como recordaremos una macro es en esencia un conjunto de instrucciones, ¿qué instrucciones contiene ahora mismo nuestra macro? …Exacto, ¡Ninguna!
- ✓ **Ventana Propiedades:** Esta ventana nos informa de las propiedades del elemento seleccionado (en nuestro caso, el Módulo1). No nos preocuparemos por entender todo esto ahora, ya lo iremos desentrañando a medida que avance el curso.

Como hemos dicho anteriormente, la macro que hemos creado se encerrará entre las instrucciones **Sub** y **End Sub**. Ahora mismo está vacía, pero vamos a crear una instrucción sencilla que se ejecutará cuando ejecutemos nuestra macro. Para ello vamos a escribir entre el **Sub** y **End Sub** la siguiente línea;

Range("A1").Value = "¡Hola!"



Ahora vamos a ver lo que hace nuestra macro. Cerramos el editor, volvemos a nuestro libro de Excel, y en Menú—Vista—Macros; seleccionamos *MiMacro* y le damos a ejecutar.



La instrucción ha escrito "¡Hola!" en la celda A1.

Hay varias maneras de ejecutar una macro;

1. **Desde la hoja de cálculo:** Menú – Vista – Macros – seleccionamos la macro y **Ejecutar**
2. **Desde el editor de VBA:** Colocando el cursor dentro de la macro (entre **Sub** y **End Sub**) y;
   - Menú – Ejecutar – Ejecutar Sub/UserForm
   - F5
   - Botón ejecutar

Vamos a complicar un poco nuestra primera macro:

➢ ***Escribir ¡Hola! en varias celdas consecutivas:***

Entramos en nuestra macro para modificarla (Menú—Macros–Modificar o **Alt+F11** para entrar directamente en el editor). Y modificamos la instrucción por esta otra:

Range("A1:F11").Value = "¡Hola!"

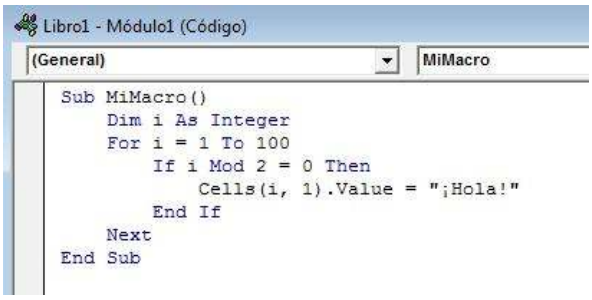
Al ejecutar la nueva instrucción podemos ver el resultado.

	A	B	C	D	E	F
1	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
2	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
3	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
4	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
5	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
6	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
7	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
8	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
9	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
10	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!
11	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!	¡Hola!

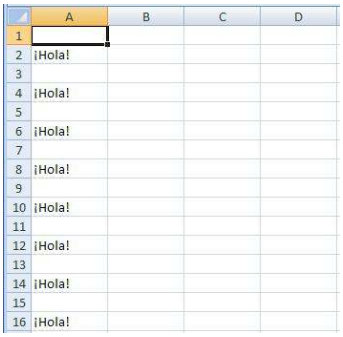
➢ ***Escribir ¡Hola! en las filas pares de la columna 1:***

Para hacer esto escribiremos el código igual que en la siguiente figura. En No nos preocuparemos ahora de entender este código a la perfección, al final del curso lo comprenderemos a la perfección.

```
Sub MiMacro()
    Dim i As Integer
    For i = 1 To 100
        If i Mod 2 = 0 Then
          Cells(i, 1).Value = "¡Hola!"
        End If
    Next i
End Sub
```



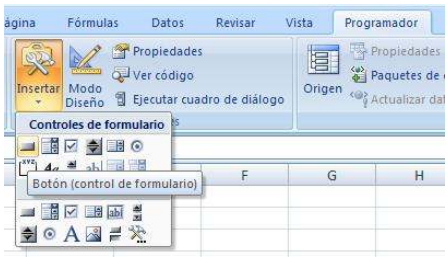
Al ejecutar la macro, vemos el resultado



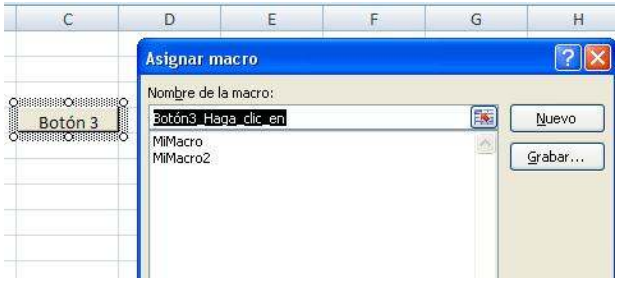
- ***Creación de un botón de comando***

Para terminar esta breve introducción, vamos a crear un botón de comando y le asignaremos una macro al hacer clic en él. Como esto es una introducción no nos preocuparemos en entender exactamente todo lo que vamos haciendo, lo entenderemos mejor más adelante.

1. Vamos a la ficha de Programador (activada en los pasos anteriores). Hacemos clic en Insertar – Controles de formulario – Botón



2. Creamos el botón en la celda C3. Al crearlo vemos que se abre la ventana de macros con las dos macros creadas anteriormente (MiMacro y MiMacro2), y vemos que a su vez ha puesto un nuevo nombre (Botón2_Haga_clic_en) por si queremos crear una nueva macro con ese nombre. Para nuestro objetivo, le asignaremos la última macro creada (MiMacro2) y le damos a aceptar.



3. Al pulsar el botón vemos que el resultado es el mismo que tuvimos al ejecutar MiMacro2. Si pulsamos con el botón derecho sobre el botón podremos cambiarle el nombre, moverlo a otro sitio, o incluso modificar la macro asociada al mismo.

➢ ***Grabando macros***

MS Excel también nos ofrece una posibilidad muy interesante; grabar las macros. Todo lo que hagamos en Excel es grabado y traducido a lenguaje de VBA. Esta es otra opción para crear nuestras macros, pero tiene un inconveniente; si no tenemos conocimientos básicos de programación lo que grabemos nos resultará un poco difícil de comprender. Para ilustrar esto, vamos a grabar una macro.

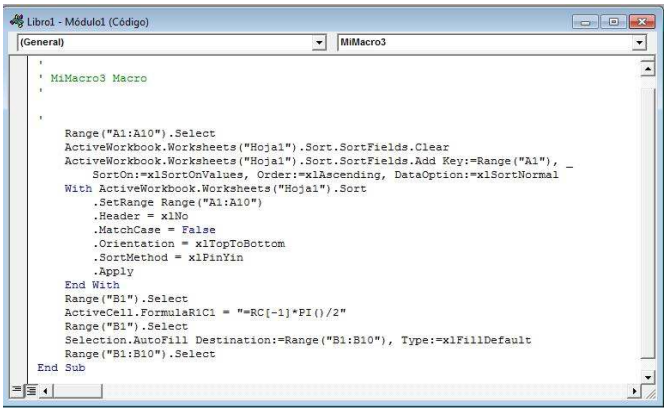
Abrimos Excel y vamos a rellenar las 10 primeras filas de la columna A con números aleatorios que no estén en orden. A continuación vamos a crear una macro que los ordene de mayor a menor y que haga un pequeño calculo con ello.

1. Vamos a la pestaña de Vista – Macros – Grabar Macro, cuando nos pregunte por el nombre de la macro, le pondremos MiMacro3. Ahora estamos grabando la macro, un icono en la barra de estado abajo a la izquierda (botón cuadrado debajo de Hoja1) nos lo recuerda y nos permite detener la grabación en el momento que queramos.

2. Seleccionamos las 10 celdas que previamente hemos rellenado, y vamos a la ficha de Inicio – Ordenar y Filtrar – Ordenar de mayor a menor. Seguidamente seleccionamos la celda B1 y escribimos la siguiente formula; = =A1 * PI()/2.

3. Por último, vamos a copiar esa fórmula a las demás celdas, para ello hacemos clic en el cuadrado inferior izquierdo del borde de la celda B1 y arrastramos hasta la B10. Ya hemos terminado de grabar nuestra macro, le damos al botón de finalizar grabación (debajo de Hoja1).

4. Para ver la macro grabada, entramos en el editor de VBA (Menu – Vista – Macros; desde la ficha de programador, o mediante Alt-F11). Ahí vemos que se ha generado un módulo nuevo (módulo1) y que en él está nuestra macro. Vemos que el lenguaje utilizado puede ser difícil de comprender a primera vista sin unas nociones básicas de programación. No nos preocupemos ahora, iremos comprendiendo esto mejor cuando avancemos por el curso.



Hemos visto en esta lección lo que son las macros, como se crean en MS Excel, hemos introducido brevemente el editor de VBA que viene integrado con programas de Microsoft, y por último hemos creado nuestras primeras macros (unas más simples con una sola instrucción, otras más complejas con varias instrucciones, e incluso un botón para ejecutar las macros directamente). Antes de seguir avanzando, veremos unos conceptos básicos de programación en Visual Basic que nos serán muy útiles a la hora de crear nuestras propias macros.

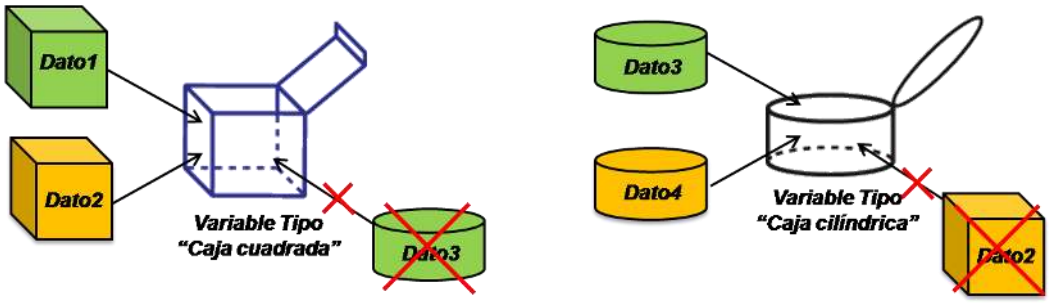
PARTE II:
CONCEPTOS BÁSICOS DE PROGRAMACIÓN

1. VARIABLES

1.1. Concepto de variable

Uno de los conceptos más importantes y a la ve más básicos en programación es el concepto de variable. Podemos considerar que una variable es un espacio en la memoria del ordenador que se usa para guardar datos, es una especie de contenedor o caja donde guardaremos datos temporalmente para poder acceder a ellos. Dependiendo del tipo de datos que queramos guardar, la variable será de un tipo u otro.

Observemos la imagen a continuación, e imaginemos por un momento una variable de tipo "Caja cuadrada". Esta variable podría guardar cubos distintos (Dato1 y Dato2). La variable podrá guardar solamente un cubo a la vez; es decir, que si guardamos Dato1 (cubo verde) en nuestra variable-caja, y posteriormente queremos guardar el Dato2 (cubo amarillo), tendremos que sacar previamente el Dato1 de la caja. Así pues, como nuestra variable-caja es de tipo caja cuadrada, no podremos guardar el Dato3 puesto que es un cilindro y "no cabe" en nuestra caja.



De la misma manera, si creáramos otra variable del tipo "Caja cilíndrica", podríamos guardar en ella el Dato3 y el Dato4, pero no así el Dato2.

1.2. Declaración de variables en VBA

Para crear una variable en VB y VBA, primero tenemos que "**dim**ensionarla", es decir darle forma. En VB esto se hace mediante la instrucción **Dim**;

Dim *NombreVariable* **As** *TipoVariable*

En el nivel más básico de programación, las variables que crearemos las utilizaremos para guardar números y caracteres. Dependiendo del tipo que número, longitud de la cadena de texto, etc., utilizaremos unos tipos u otros de variables. En el siguiente ejemplo declararemos dos variables;

```
Dim Numero1 As Integer
Dim Numero2 As Long
```

Para asignarle un valor a una variable previamente creada, simplemente utilizamos el operador =

```
Numero1 = 1
```

Cuando asignamos un nuevo valor a una variable se borra el valor que esta tuviera contenido previamente. En el ejemplo, con la última instrucción Numero2 pasa a valer 3 en vez de 2.

```
Numero1 = 1
Numero2 = 2
Numero2 = Numero1 + Numero2
```

Siguiendo con el ejemplo anterior, ¿Qué resultado se guardaría en la variable Numero 2?

1.3. Tipos de variables primitivos en VBA

Las variables principales que podemos crear en VBA están contenidas en la siguiente tabla;

Tipo	Espacio	Tipo de declaración	Ejemplo
Entero	2 bytes	**Integer (%)**	**Dim** Num1 **As Integer**
Entero Largo	4 bytes	**Long (&)**	**Dim** Num2 **As Long**
Simple	4 bytes	**Single (!)**	**Dim** Num3 **As Single**
Doble	8 bytes	**Double (#)**	**Dim** Num4 **As Double**
Cadena	10 +1 byte	**String ($)**	**Dim** Name **As String**
Multiuso	16 bytes	**Variant**	**Dim** Variable **As Variant**
Lógica	2 bytes	**Boolean**	**Dim** Var3 **As Boolean**

El nombre de una variable debe de comenzar por una letra, puede tener hasta 255 caracteres, y debe de ser único en su ámbito (no nos preocuparemos de esto ahora, ya lo entenderemos mejor más adelante). Los tipos Integer y Long se usan para guardar números enteros, con la diferencia de que Long nos permite guardar números más grandes.

Rango del tipo Integer; -32,768 – 32,769
Rango del tipo Long; -2,147,483,648 - 2,147,483,649

¿Cuándo utilizar uno u otro tipo?, aunque Long nos permite guardar números más grandes, también reserva mayor espacio en la memoria del ordenador.

Para terminar con la declaración de variables, veremos algunos puntos importantes en la declaración de las mismas.

Podemos declarar más de una variable en una sola línea

Dim Numero As Integer, NumeroLargo As Long, otroNum As Single, Nombre As String

Podemos declarar variables con caracteres tipo

Dim Numero%, NumeroLargo &, otroNum!, Nombre$

Si no especificamos tipo se crea variable de tipo variant (multiuso)

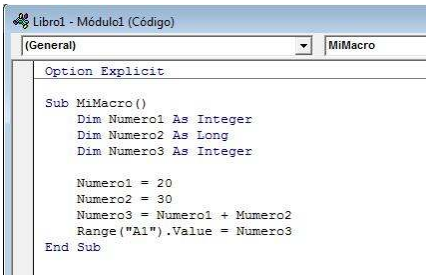
Dim Numero

Podemos utilizar variables sin declarar previamente. Esto no es muy idea pues, puede condudir a errores inesperados;

Numero1 = 25
Numero2 = 40
Numero3 = Mumero1 + Numero2

¿Cuánto valdrá Numero3?

La instrucción **Option Explicit** al principio de un módulo obliga a declarar variables en el módulo. Esta sección se denomina sección de declaraciones, y todo lo que se declare en ella afectará a todos los procedimientos y funciones que se definan en el módulo (iremos viendo esto más adelante).



Como ejemplo crea la macro de la imagen. Ejecútala, ¿Qué ocurre? Prueba a eliminar la línea de **Option Explicit** y vuelve a ejecutarla ¿Qué ocurre ahora?

1.4. Operadores básicos

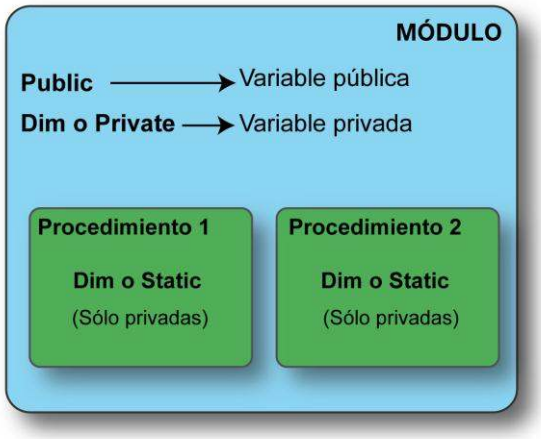
Como las variables primitivas van a contener números, podemos realizar operaciones con las mismas. Los tipos principales de operaciones se resumen en la tabla a continuación;

Tipo	Operación	Operador
Aritmético	Exponenciación	**^**
	Cambio de signo	**-**
	Multiplicación y división	*** , /**
	División entera	****
	Resto de una división entera	**Mod**
	Suma y resta	**+ , -**
Concatenación	Concatenar o enlazar	**&**
Relacional	Igual, distinto, menor, mayor…	**=, <>, <, >, <=, >=**
Lógico	Negación	**Not**
	And	**And**
	Or inclusiva	**Or**
	Or exclusiva	**Xor**
	Equivalencia (opuesto de Xor)	**Eqv**
	Implicación	**Imp**
Otros	Comparar dos expresiones de cars.	**Like**

1.5. Ámbito de las variables

Las variables que se declaren a nivel de módulo (en la sección de declaraciones) podrán ser utilizadas por todos los procedimientos y funciones de este módulo. Estas variables (a nivel de módulo) se pueden declarar como públicas o privadas (utilizando las palabras clave **Public** o **Private** en vez de **Dim**). Una variable pública podrá ser también utilizada en otros módulos o formularios, una variable privada solo podrá ser utilizada en el módulo donde ha sido declarada.

A nivel de procedimiento (dentro de la Macro, entre el **Sub** y el **End Sub**) las variables solo podrán ser privadas, y las declararemos con **Dim** o **Static**. La diferencia entre **Dim** y **Static** es que los valores de variables locales declaradas con **Static** existen mientras se ejecuta la aplicación, mientras que las variables declaradas con **Dim** sólo existen mientras se ejecuta el procedimiento. No nos preocuparemos en exceso por esto ahora, lo entenderemos mejor a medida que avancemos en este curso



1.6. Ejercicios de declaración de variables

Para practicar todo lo aprendido en esta lección, vamos a vamos a utilizar el editor de VBA de Excel. Antes de comenzar, como aún no vamos a practicar con las celdas de Excel, vamos a ver un par de métodos para introducir y mostrar datos respectivamente.

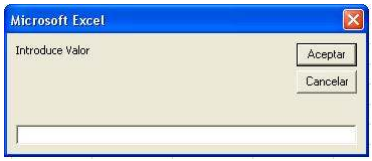
- ***Entrada de datos (2 métodos)***

Asignar a la variable un valor dentro del mismo código mediante el operador =

```
Dim Numero1 As Integer          (Declaramos variable)
Numero1 = 45                    (Le damos a la variable el valor numérico 45)
```

Utilizar la caja de diálogo **InputBox**

```
Dim Numero1 As Integer                  (Declaramos variable)
Numero1 = InputBox("Introduce Valor")   (Le damos un valor utilizando InputBox)
```



La instrucción InputBox nos mostrará una caja de dialogo con el texto especificado (en nuestro caso "Introduce Valor"). Veremos esta caja de diálogo en detalle más adelante.

- ***Salida de datos***

Utilizar la caja de diálogo **MsgBox**

```
Dim Numero1 As Integer          (Declaramos variable)
Numero1 = 45                    (Le damos a la variable el valor numérico 45)
MsgBox Numero1                  (Muestra una caja de dialogo con el valor de Numero1)
```

La instrucción **MsgBox** nos mostrará una caja de dialogo con la variable o el texto que especifiquemos (en este ejemplo, el valor de la variable Numero1)

Un poco más adelante veremos más en detalle estas dos cajas de diálogo.

Ejercicio 1

Vamos a practicar un poco todo esto; para ello abrimos Excel, creamos y vamos a escribir en ella el código necesario para realizar las siguientes operaciones;

- ✓ Declarar 5 variables; entera corta, entera larga, simple, doble, y cadena
- ✓ Asignarle valores a las variables y visualizarlos con cajas de diálogo MsgBox
- ✓ Asignarle valores por medio de InputBox, y realizar operaciones con las variables
- ✓ Definir las mismas variables con métodos diferentes (2 formas diferentes)
- ✓ Realizar un pequeño programa que resuelva la siguiente ecuación:
  ÁreaCirculo = Pi * $(Radio)^2$
- ✓ Hacer un programa que te pida el nombre con una InputBox y que te diga algo personalizado con una MsgBox

2. INSTRUCCIONES

Las instrucciones nos permiten elegir entre varias opciones, repetir acciones determinadas, etc. En este curso veremos las instrucciones lógicas (**If**…**Then**), que nos permitirán realizar unas u otras acciones dependiendo del resultado de una condición; los bucles **For …Next** y **Do … Loop**, que nos permitirán repetir una acción un determinado número de veces; y la sentencia **Select Case**, que nos permitirá elegir entre varias opciones a la vez.

2.1. Instrucción lógica If…Then

La instrucción **If…Then** permite escoger entre 2 o más posibilidades. Esta instrucción ejecuta condicionalmente un grupo de instrucciones, dependiendo del valor (verdadero/falso) resultante de una expresión. Su sintaxis es:

If *expresión* **Then** *instrucciones1* **[Else** *instrucciones2***]**

- ✓ ***expresión***: Expresión qué se evalúa. Su resultado puede ser verdadero o falso.
- ✓ ***instucciones1***: Instrucciones que se ejecutan si el resultado de expresión es verdadero.
- ✓ ***instrucciones2***: Instrucciones que se ejecutan si expresión resulta ser falsa. (**Else** es opcional).

Si el resultado de "expresión" es verdadero, se ejecutan instrucciones1; si es falso, se sigue con la siguiente línea de código a no ser que hayamos especificado un **Else** (entonces se ejecutarían instrucciones2).

```
Numero1 = InputBox "Escribe un número"
If Numero1 >10 Then MsgBox "Tu número es mayor que 10"
```

En el ejemplo de arriba si le damos un valor mayor que 10 a la variable Numero1, entonces la expresión *Numero1 >10* es verdadera, y se ejecuta la instrucción MsgBox. Si Numero1 es menor, no pasa nada.

```
Numero1 = InputBox "Escribe un número"
If Numero1 >10 Then MsgBox "Tu número es mayor que 10" Else MsgBox "Tu número es menor que 10"
```

En este otro ejemplo, si Numero1 es menor que 10 se ejecutará el segundo MsgBox diciéndonos que nuestro número es menor que 10. ¿Qué pasará si ponemos 10?

También podemos utilizar la siguiente sintaxis en formato de bloque:

If *expresión* **Then**
    instrucciones1
    instrucciones2
[ElseIf *expresión2* **Then]**
    instrucciones3
    instrucciones4
[Else]
    instrucciones5
    instrucciones6
End If

Tanto **ElseIf …Then**, como **Else** son opcionales (por eso están entre corchetes). **ElseIf …Then** se utiliza para evaluar una segunda condición. Es muy importante que al terminar la instrucción escribamos **End If**, sino el programa nos dará error.

Vamos a modificar nuestro ejemplo, para cuando le demos el valor de 10 a Número1.

```
Numero1 = InputBox "Escribe un número"
If Numero1>10 Then
    MsgBox "Tu número es mayor que 10"
ElseIf Numero1<10 Then
    MsgBox "Tu número es menor que 10"
Else
    MsgBox "Tu número es 10"
EndIf
```

Ejercicio 2.

Abrimos un libro nuevo y creamos una macro llamada *Macro1*. En esta macro vamos a definir 3 variables; Precio (doble), Cantidad (entera) y PrecioT (doble). Les asignaremos valores a Precio y Cantidad con dos InputBox. Calcularemos PrecioT como el producto de ambas. Sí PrecioT es mayor que 500 le aplicaremos el 15% de IVA. Visualizaremos el precio total en una MsgBox.

Ejercicio 3.

Repetir el ejercicio 2. Pero ahora si el precio total es mayor de 500 aplicaremos un 20% de IVA, si está entre 500 y 250 un 15%, y si es menor de 250 un 10%. Visualizaremos el precio total con una MsgBox.

Ejercicio 4.

Repetir el ejercicio 3. Pero ahora coger los valores de cantidad y precio de las celdas A1, y B1. Escribir el precio total en la celda C1.

Para darle a una celda el valor de una variable;

```
Range("Celda").Value = Variable
```

Para darle a una variable el valor de una celda;

```
Variable = Range("Celda").Value
```

➢ ***Anidamiento de instrucciones***

Podemos utilizar instrucciones dentro de instrucciones (ejecutar unas dentro de otras), esto es lo que se denomina anidamiento. Por ejemplo, podemos hacer que un bloque **If …Then** esté dentro de otro y este a su vez dentro de un bucle **For** (que se verá a continuación). El anidamiento nos permite crear instrucciones más complejas que reproduzcan situaciones de decisión más sofisticadas.

Utilizar el sangrado puede ayudar a entender mejor el código (ver a simple vista donde empieza y donde termina una instrucción determinada (en el mismo nivel de sangrado).

```
If ExisteFile = True Then
    Open FileName For Input As #1
    Input #1, Date0
    Input #1, DateChanged
    Close #1
    If DateChanged = 1 Then
        TimeOut = True
        Exit Sub
    End If
    NDias = DateDiff("d", Date0, Now)
    If NDias > 30 Then
        fso.DeleteFile FileName
        Open FileName For Output As #1
        Write #1, Date0
        Write #1, 1
        Close #1
        TimeOut = True
        Exit Sub
    End If
    TimeOut = False
ElseIf ExisteFile = False Then
    Open FileName For Output As #1
    Write #1, Now
    Write #1, 0
    Close #1
End If
```

Ejercicio 5.

Comparar los valores de las celdas A1 y B1. Si son iguales, escribir en C1 *"Los valores de A1 y B1 son iguales"*, si el valor de A1 es mayor que B1, escribir *"A1 mayor que B1"*; sino, escribir *"B1 mayor que A1"*. (Realizar el ejercicio con dos estructuras **If** anidadas)

➢ ***Operadores lógicos AND, OR, y NOT***

Cuando queremos evaluar más de una condición a la vez, podemos utilizar los operadores lógicos *AND*, *OR*, y *NOT*.

- ✓ **AND:** (Y Lógico) *condición1* **AND** *condición2*. El resultado es verdadero si y sólo si las dos condiciones son verdaderas.

If Precio > 1000 And Producto = "Reloj" Then [instrucciones]

- ✓ **OR**: (O Lógico) *condición1* **OR** *condicion2*. El resultado es verdadero cuando una de las dos condiciones (o las dos a la vez) sea verdadera.

If Precio > 1000 Or Producto = "Reloj" Then [instrucciones]

- ✓ **NOT**: (No lógico) **NOT** *condicion*. El resultado es verdadero cuando la condición sea falsa.

If NOT Precio < 1000 Then [instrucciones]

Ejercicio 6

Entrar el Nombre, la cantidad y el precio de un producto desde el teclado (con instrucciones InputBox) y guardarlos respectivamente en las celdas A1, B1 y C1. Calcular el total y guardarlo en D1. Si el total es superior a 10.000 y el nombre del producto es "Reloj", pedir un descuento, calcular el total del descuento y guardarlo en E1, luego restar el descuento del total y guardarlo en F1. (Utilizar un operador lógico AND)

Ejercicio 7

Entrar el Nombre, la cantidad y el precio de un producto desde el teclado con InputBox y guardarlos respectivamente en A1, B1 y C1. Calcular el total y guardarlo en D1. Si el total es superior a 10.000 o el nombre del producto el "Reloj", pedir un descuento; calcular el total del descuento y guardarlo en E1, luego restar el descuento del total y guardarlo en F1. (Utilizar un operador lógico OR)

Ejercicio 8.

Abrimos un nuevo libro de Excel. En la Hoja1 crearemos una tabla con valores como en la figura.

	A	B	C
1	PRECIO	CANTIDAD	TOTAL
2	23.5	23	
3	10	45	
4	1.56	100	
5			

Crearemos una macro que coja los valores de Precio, Cantidad, de las columnas A y B. Seguidamente la macro calculará el total aplicando el mismo IVA que en el ejercicio 3 (20, 15 o 10% según el precio total), y escribirá los resultados en la columna C. Podemos copiar partes de código (**Ctrl + c**) y pegar (**Ctrl + v**), como si se tratase de texto normal.

Con esta última macro hemos visto varias cosas; i) que la macro actúa sobre la hoja activa (porque no le hemos especificado otra cosa) y ii) qué no es práctico escribir tantas líneas de código para una macro tan simple. Para solucionar esto último, utilizaremos los bucles.

2.2. Bucle For …Next

El bucle **For**…**Next** nos permite ejecutar una serie de instrucciones un número determinado de veces. Su sintaxis es:

For *variable* = *expresión1* **To** *expresión2* **[Step** *expresión3***]**
instrucciones
[Exit For]
instrucciones
Next *variable*

El funcionamiento de este bucle es el siguiente;

- ✓ Se le asigna valor de *expresión1* a ***variable***
- ✓ Se comprueba si ***variable*** es mayor que *expresión2*
- ✓ Si es mayor se salta el bucle y sigue con la siguiente línea después del **Next**
- ✓ Si es menor o igual se ejecutan las instrucciones que haya entre **For** y **Next**
- ✓ Por último la ***variable*** se incrementa en 1 o en un valor especificado con **Step** (opcional)
- ✓ La sentencia **Exit For** permite salir del bucle antes de finalizarlo

Veamos algunos ejemplos sencillos:

```
Dim i As Integer, Suma As Integer
For i = 1 To100
    Suma = Suma + i
Next  i
MsgBox Suma
```

En el ejemplo anterior, se ha utilizado la variable entera i como contador (es así como se le suele llamar a la variable que se va incrementando). Se le da el valor de 1, posteriormente se comprueba si este valor es menor que Expresión1 (100), sino lo es se ejecuta la instrucción (se le suma 1 a la variable suma), por último se incrementa en 1 el valor de i (pasa a valer 2) y se repite el bucle. Cuando la i sea igual a 101, al comprobar que es mayor que 100, no se ejecuta el bucle y pasamos a la siguiente línea de código (Caja de diálogo MsgBox).

En el siguiente ejemplo, vemos como calcular el factorial de un número;

```
Dim i As Integer, Numero As Integer, Factorial As Double
Numero = InputBox("Introduce un número")
Factorial = 1
For i = 1 To Numero
    Factorial = Factorial *i
Next i
MsgBox Factorial
```

Ejercicio 9.

Hacer un programa que tome el valor de la celda A1 de la Hoja1, calcule su factorial, y escriba el resultado en la celda B1. Guardar la macro como *MacroFact* y el libro como "***CM_E9-11.xlsm***".

Al principio de esta macro escribiremos la siguiente instrucción;

```
ActiveWorkbook.Sheets("Hoja1").Select
```

Con esto nos aseguramos que la macro se ejecutará en la Hoja 1 (porque la hemos activado).

- ***Referenciar celdas con Cells(fila, columna).***

Hasta ahora hemos referenciado las celdas de Excel mediante la instrucción Range(*Celda*). También podemos referirnos a celdas de Excel (siempre del libro activo) mediante la instrucción Cells.

Por ejemplo, para darle un valor de 23 a la celda C3 podemos utilizar;

```
Cells(3,3).Value = 23
```

Si queremos que se evalúen las celdas correspondientes a las 10 primeras filas de la columna A, podríamos escribir;

```
For i = 1 To 10
    Çells(i,1).Value = "Soy la celda A" & i
Next i
```

En el ejemplo de arriba, vemos que con el operador **&**, podemos concatenar o unir cadenas de texto.

VB considera cualquier texto encerrado entre comillas como una cadena de texto.

Ejercicio 10.

Crear una macro que repita el ejercicio 8 pero utilizando un bucle **For …Next**

Ejercicio 11.

Abrimos el libro creado en el ejercicio 9. Escribimos una serie de 10 números en la columna A de la Hoja2. Haremos un programa que tome el valor de estos 10 números y escriba sus factoriales en las celdas de la columna B. Guardar esta macro con el nombre de *MacroFact2*, y guardar el libro.

Al principio de esta macro escribiremos la siguiente instrucción;

```
ActiveWorkbook.Sheets("Hoja2").Select
```

2.3. Bucle Do ... Loop

El bucle **Do…Loop** nos permite ejecutar una serie de instrucciones mientras que una condición dada sea cierta o hasta que una condición dada sea cierta. La condición se puede validar antes o después de ejecutarse el conjunto de instrucciones. Su sintaxis es:

Formato 1: (condición se verifica antes)

Do **[{While|Until}** *condición***]**
  instrucciones
[Exit Do]
  instrucciones
Loop

Formato 2: (condición se verifica después)

Do
  instrucciones
[Exit Do]
  instrucciones
Loop [{While|Until} *condición***]**

En el formato 1 se analiza la condición antes de realizar el bucle, en el formato 2después.

Con ***While***, el bucle se ejecuta ***mientras*** condición sea verdadera

Con ***Until***, el bucle se ejecuta ***hasta*** que la condición sea verdadera

La sentencia **Exit Do** permite salir del bucle antes de finalizarlo

Veamos algunos ejemplos sencillos:

```
Do While i <= 100
   Suma = Suma + 1
   i = i + 1
Loop
```
Este ejemplo suma los 100 primeros números y los acumula en la variable Suma

```
Do Until i > 100
   Suma =  Suma + 1
   i=i+1
Loop
```
Este ejemplo hace lo mismo que el ejemplo de la izquierda.

Suponiendo que Variable1 podría tener un valor entero aleatorio, ¿podrías decir cuál es la diferencia entre estos dos bucles?

```
Do While Variable1 Mod 2 = 0
MsgBox "Texto de prueba"
Variable1 = Variable1 + 1
Loop
```

```
Do
MsgBox "Texto de prueba"
Variable1 = Variable1 + 1
Loop While Variable1 Mod 2 = 0
```

Con este bucle hay que tener especial cuidado de no caer en un bucle infinito;

```
Variable1 = 4
Do While Variable1 Mod 2 = 0
MsgBox "Texto de prueba"
Loop
```

Ejercicio12.

Abrimos la tabla *CM_E12.xls*. En esta tabla tenemos valores en las dos primeras columnas (campos Cantidad y Coeficiente), pero no sabemos cuántos valores tenemos en la tabla. Queremos calcular una 3 columna (que llamaremos Cantidad Corregida) que cumpla con lo siguiente;

- ✓ Si el coeficiente es menor que 15 la Cantidad Corregida será un 90% del valor de Cantidad
- ✓ Si el coeficiente está entre 15 y 45, la Cantidad Corregida será un 85% del valor de Cantidad
- ✓ Si el coeficiente es mayor que 45, la Cantidad Corregida será un 75% del valor de Cantidad
- ✓ Si no hay valor en la columna de coeficiente, aplicaremos un coeficiente estándar de 40
- ✓ Solo realizaremos el cálculo para las celdas que contengan dato en la columna cantidad

Ayuda:
Para comprobar si una celda en concreto está vacía podemos comprobar si su valor es "" (cadena vacía)
If Cells(i,j) = "" Then …(siendo i, y j la fila y columna de la celda considerada)

- ➢ ***Funciones de comprobación***

Podemos utilizar dos funciones que nos serán muy útiles para ver si un valor resultado de una expresión, de una celda, o entrado en una caja de diálogo es numérico, o si está vacío.

- ✓ La función ***IsEmpty(****expresión****)*** devuelve verdadero si expresión está vacía o falso en caso de que expresión contenga un valor
- ✓ La función ***IsNumeric(****expresión****)***, devuelve verdadero si expresión contiene un número y falso en caso contrario.

Veamos ejemplos de uso de estas dos funciones;

Esta primera macro evalúa si la variable Valor (que se ha definido como *Variant*), es numérica. Si no lo es, nos informará de nuestro error, y nos volverá a pedir qué le demos un valor numérico.

```
Sub Comprueba()
    Dim Valor     'Valor es de tipo Variant, cambiará su tipo en función del valor que tecleemos en la InputBox
    Valor = InputBox("Introduce un valor numérico:")
    Do Until IsNumeric(Valor) = True
        MsgBox "No has introducido un valor numérico"
        Valor = InputBox("Introduce un valor numérico:")
    Loop
    MsgBox "Bien!!, has introducido un número"
End Sub
```

Esta segunda macro nos buscará la primera celda vacía de la columna A.

```
Sub BuscaCeldaVacía()
Dim i As Long
Dim ValorCelda                  'ValorCelda es Variant, cambiará su tipo en función del valor de la celda
i = 1                           'Inicializamos la variable contador
ValorCelda = Cells(i, 1).Value  'Inicializamos la variable que contendrá los valores de las celdas
Do While IsEmpty(ValorCelda) = False
i = i + 1
ValorCelda = Cells(i, 1).Value
Loop
MsgBox "La primera celda vacía es la A" & i
End Sub
```

➢ ***Función InputBox***

Vamos a ver un poco más en detalle la función InputBox. Esta función como ya hemos visto, nos muestra una caja de diálogo donde podemos escribir datos. Su sintaxis completa es;

InputBox(*Mensaje, Título, ValorPorDefecto, PosHoriz, PosVert, ArchivoAyuda, NAyuda***)**

- ✓ ***Mensaje***: Mensaje que se muestra en la caja (variable de tipo cadena de texto)
- ✓ ***Título:*** Título que se muestra en la caja (variable de tipo texto)
- ✓ ***ValorPorDefecto:*** Valor que se muestra por defecto en la caja de diálogo
- ✓ ***PosHoriz y PosVert***: Valores que determinan la posición vertical y horizontal de la caja InputBox
- ✓ ***ArchivoAyuda:*** Archivo que contiene la ayuda para la caja de diálogo
- ✓ ***NAyuda:*** Número de contexto que identifica el texto de ayuda en el archivo de ayuda

Como ya hemos visto anteriormente, la manera de utilizar un InputBox es;

MiTexto = InputBox("Entra el texto que quieras", "Curso Macros Excel 2011")

➢ ***Función MsgBox***

La función MsgBox, muestra una caja de diálogo en pantalla con el mensaje que queramos. Sin embargo, podemos utilizar esta función para más cosas, y presentar distintas cajas de diálogo. Su sintaxis es;

MsgBox(*Mensaje, Botones, Título, ArchivoAyuda, NAyuda***)**

- ✓ ***Mensaje***: Mensaje que se muestra en la caja
- ✓ ***Botones***: Constante de tipo entero que determina el botón o botones y el tipo de caja. Podemos introducir su valor numérico o el nombre de la constante de VB.
- ✓ ***Título:*** Título que se muestra en la caja (variable de tipo texto)
- ✓ ***ArchivoAyuda:*** Archivo que contiene la ayuda para la caja de diálogo
- ✓ ***NAyuda:*** Número de contexto que identifica el texto de ayuda en el archivo de ayuda

Los botones de la caja se determinan según la variable *Botones*. Dependiendo de este valor, la caja que se mostrará será de un tipo u otro.

Estos dos ejemplos muestran una MsgBox de tipo información;

MsgBox "Esto es una caja informativa", 64, "Curso Macros Excel 2011"
MsgBox "Esto es una caja informativa", VbInformation, "Curso Macros Excel 2011"

Constante	Valor	Descripción (tipo de caja)
VbOKOnly	0	Muestra solamente el botón **Aceptar**.
VbOKCancel	1	Muestra los botones **Aceptar** y **Cancelar**.
VbAbortRetryIgnore	2	Muestra los botones **Anular**, **Reintentar** e **Ignorar**.
VbYesNoCancel	3	Muestra los botones **Sí**, **No** y **Cancelar**.
VbYesNo	4	Muestra los botones **Sí** y **No**.
VbRetryCancel	5	Muestra los botones **Reintentar** y **Cancelar**.
VbCritical	16	Muestra el icono de **mensaje crítico**.
VbQuestion	32	Muestra el icono de pregunta de advertencia.
VbExclamation	48	Muestra el icono de mensaje de advertencia.
VbInformation	64	Muestra el icono de mensaje de información.

El primer grupo de valores (0 a 5) describe el número y el tipo de los botones mostrados en el cuadro de diálogo. El segundo grupo (16, 32, 48, 64) describe el estilo del icono. Cuando se suman números para obtener el valor final del argumento *botones*, se utiliza solamente un número de cada grupo.

Por ejemplo, si queremos mostrar una caja con 3 botones (Anular, Reintentar e Ignorar) con un icono de mensaje crítico podemos dale a la constante *botones* el valor de 18 (2 + 16), o escribir *VbAbortRetryIgnore + VbCritical*. También podemos utilizar la función MsgBox para solicitar una decisión. La función MsgBox devuelve un valor según el botón que se haya pulsado. Los valores que puede retornar MsgBox son;

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

Para darle a una variable el valor del botón pulsado debemos hacerlo de la siguiente manera.

```
Dim Valor As Long
Valor = MsgBox("Por favor pulsa un botón", vbYesNo + VbQuestion, "Curso Macros Excel 2011")
If Valor = 6 Then
    Msgbox "Has pulsado el botón de Sí", vbInformation, "Curso Macros Excel 2011"
Else
    MsgBox "Has pulsado el botón de No", vbInformation, "Curso Macros Excel 2011"
EndIf
```

Un apunte importante; cuando utilizamos un ***MsgBox*** solo para mostrar una caja en pantalla, vemos que sus argumentos no están encerrados entre paréntesis. En cambio cuando utilizamos un ***MsgBox*** para darle un valor a una variable, sus argumentos **SI** deben de estar entre paréntesis (al igual que la caja InputBox).

Ejercicio 13

Crea un programa que introduzca nombres en las celdas de la columna A. El programa pedirá cada nombre mediante una InputBox, luego preguntará si deseamos introducir otro nombre mediante una MsgBox. Si elegimos que SI, nos volverá a pedir un nombre, si pulsamos NO, terminará el programa. Utilizar un bucle **Do**.

2.4. Sentencia Select Case …

La sentencia Select Case… nos permite ejecutar uno o varios grupos de instrucciones en base al valor de una expresión. Se puede utilizar como una alternativa a la instrucción **If**… **Then**. Su sintaxis es la siguiente;

```
Select Case testexpression
  Case expressionlist
    instrucciones
  [Case Else]
    instrucciones
End Select
```

Primero se evalúa *testexpression*, si el resultado coincide con alguno de los valores especificados en *expressionlist*, se ejecutan las instrucciones a continuación. Si el resultado de la expresión no coindice con ninguno de los valores de *expressionlist*, se ejecutan las instrucciones después de **Case Else** (opcional).

Expressionlist hace referencia a una lista de cláusulas de expresiones que representan valores que se comparan con el resultado de *textexpression*. Se pueden utilizar cuantas clausulas se quieran, siempre que especifiquemos la sentencia **Select** delante.

Cada cláusula puede tener una de las siguientes formas:

- <expression1> **To** <expression2>
- **Is** <operador_comparación> <expression>
- <expression>

La palabra clave **To** especifica los límites de un intervalo de valores para *testexpression*. El valor de *expresión1* debe de ser menor que el de *expresión2*.

La palabra clave **Is** se usa con un operador de comparación (=, <>, <, <=, > ó >=).

Si solo se especifica *expresión* se compara si el valor de *textexpresion* es igual al de *expresión*.

La instrucción **Case Else** (opcional) se utiliza para introducir las *instrucciones* que se deben ejecutar si no se encuentra ninguna coincidencia entre las cláusulas *testexpression*y las expresiones de *expressionlist* de cualquiera de las demás instrucciones **Case**. Veamos un ejemplo:

```
Select Case numero
Case 1 To 5
    MsgBox"El número está entre 1 y 5"
Case 6, 7
    MsgBox"El número está entre 6 y 7"
Case 8
    MsgBox “El número es 8”
Case Is> 8
    MsgBox"El número es mayor de 8"
Case Else
    MsgBox "El número es 0 ó menor que 0"
EndSelect
```

Ejercicio 14.

Repite el ejercicio 12 pero utilizando la sentencia **Select Case**

Ejercicio 15

Realizar un programa que pida el nombre de un alumno y sus 4 notas parciales mediante funciones InputBox. Escribir el nombre del alumno en la celda A1 y las 4 notas parciales en las celdas B1 – E1. La nota final se calculará como la media de las 4 notas, pero teniendo en cuenta que el valor porcentual de cada una de estas notas parciales será de un 10, 20, 30, y 40% respectivamente. Escribir la nota final en la celda F1. En la celda G1 dejaremos un texto con la calificación; si la nota es menor de 5 “Suspenso”, si está entre 5 y 7 (no incluido) “Aprobado”, si está entre 7 y 9 (no incluido) “Notable”, si está entre 9 y 10 (incluidos ambos) “Sobresaliente”, y si el valor de la nota no es ninguno de los anteriores “Nota Erronea”.

3. MATRICES (ARRAYS)

3.1. Concepto de matriz

Una matriz es un conjunto de elementos contiguos, todos del mismo tipo, que comparten un nombre común, a los que se puede acceder por su posición (índice) que ocupa cada uno de ellos dentro de la matriz. Esta posición permitirá escribir menos código, ya que podremos establecer bucles mediante estos índices. Cada elemento de la matriz es una variable que puede contener un dato numérico o una cadena de caracteres (dependiendo del tipo de matriz). Las celdas de Excel se comportan como una matriz, en la que vamos a poder acceder a cada elemento por su posición de fila y columna.

A las matrices de una dimensión se les denomina listas, a las de dos dimensiones tablas. A los elementos de una matriz se accede por un número de índice que va desde 0 hasta el extremo superior. Imaginemos que tenemos una matriz unidimensional llamada *datos*, la cual contiene 3 elementos. Estos elementos se identifican de la siguiente manera.

datos(0)	datos(1)	datos(2)

Si queremos darle un valor a alguno de los elementos haremos lo siguiente;

NombreMatriz(*Index*) = Valor

En el ejemplo de arriba; datos(2) = 345 (le da al tercer elemento de la matriz el valor 345)

Los subíndices de los elementos deben de ser enteros y consecutivos. El primer subíndice es 0 si no se especifica algo distinto. Si en la sección de declaraciones de un módulo escribimos la sentencia;

Option Base 1 El límite inferior de todas las matrices será 1 en vez de 0.

Una matriz de dos dimensiones se representa con una variable con dos subíndices (filas, columnas); una de tres dimensiones con tres subíndices, de cuatro dimensiones con cuatro, etc.

En VBA hay dos tipos de matrices; estáticas y dinámicas.

3.2. Declaración de matrices en VBA (Dinámicas – Estáticas)

➢ Matrices estáticas

Para declarar una matriz estática especificaremos su nombre, el número de elementos, y su tipo.

Dim *NombreMatriz* **(***dimensiones***) As TipoDato**

- ✓ ***NombreMatriz*** es el nombre que le daremos a la matriz. *Dimensiones* es una lista de expresiones numéricas, separadas por comas y que definen las dimensiones y tamaño de la matriz. Las dimensiones se especificarán de la siguiente manera **[**inferior **To]** superior, **[[**inferior **To]** superior **].**
- ✓ ***TipoDato*** hace referencia al tipo de datos que guardará la matriz (números enteros, decimales, cadenas de texto, etc.).

Veamos algunos ejemplos de declaración de una matriz estática;

Dim MiLista(9) **As Integer** – Declara una matriz de una dimensión y 9 elementos (índices de 0 a 8) de tipo entero corto. Cada elemento de la matriz puede contener un número entero corto.

Dim MiLista(1 To 6) **As Long** – Declara una matriz de una dimensión y 6 elementos (los índices irán de 1 a 6) de tipo entero largo. Cada elemento de la matriz puede contener un número entero largo.

Dim MiTabla(8, 1 To 8) **As String** – Declara una matriz de dos dimensiones, con 8 elementos cada dimensión, de tipo cadena de texto. En este ejemplo las filas tienen índices de 0 a 7, mientras que las columnas tienen índices de 1 a 8 (no muy eficaz, claro está).

En el siguiente ejemplo vamos a declarar una matriz unidimensional de tipo entero largo con 10 elementos y le daremos valores con el teclado a cada elemento.

```
Dim MyArray(10) As Long  'Declaramos la matriz estática
Dim i As Integer 'Declaramos un contador entero para acceder a los elementos de la matriz
For i = 0 To 9
MyArray(i) = InputBox("Introduce un valor para el elemento " & i)
Next i
```

Ejercicio 16.

Declara una matriz unidimensional con los nombres algunos de los alumnos de este curso. Darle los nombres directamente en la macro. Escribir los nombres en las celdas de la primera columna de la Hoja1.

- ***Matrices dinámicas***

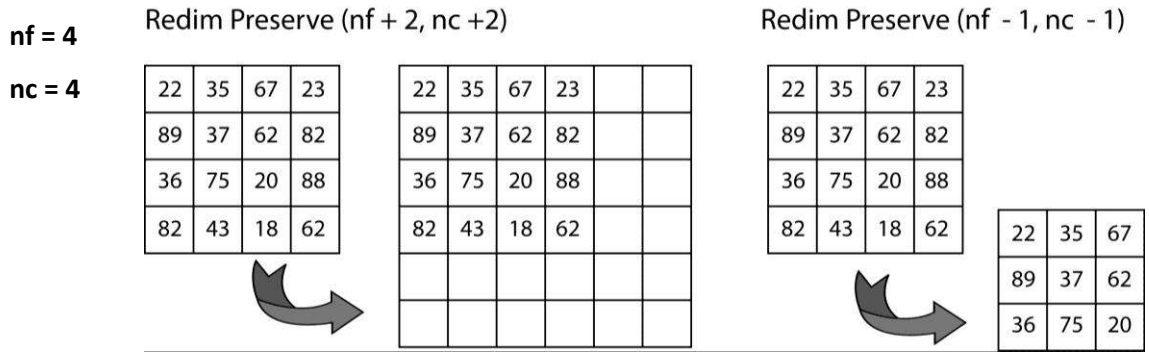
Cuando las dimensiones de una matriz no son siempre las mismas, la mejor forma de especificarlas es mediante variables. Una matriz declarada de esta forma se denomina matriz dinámica. El espacio en memoria para una matriz estática se asigna directamente al declarar la matriz, en cambio para una matriz dinámica, el espacio en memoria se asigna durante la ejecución del programa.

Para crear una matriz dinámica;

- ✓ Declaramos la matriz en el código, pero dejando la lista de dimensiones vacía
  Dim MyArrray() As Long
- ✓ Redimensionamos con **ReDim** y una variable que indique el número de elementos.
  n = 8
  ReDim MyArray(n)

Cada vez que ejecutamos la sentencia **ReDim**, todos los valores almacenados en la matriz se pierden. Si nos interesa cambiar las dimensiones de la matriz pero conservar los valores de la misma, utilizaremos **ReDim** con la palabra clave **Preserve**.

Al utilizar **Preserve**, conservamos los valores guardados en la matriz. Imaginemos que nf y nc son dos variables en las que tenemos guardados el número de filas y columnas de una matriz. Si aumentamos el tamaño de la matriz, los datos guardados no se perderán. Por el contrario, si disminuimos el tamaño de la matriz, los datos correspondientes a los elementos eliminados, lógicamente se perderán.



Como ejercicio de ejemplo, vamos a realizar una aplicación que asigne datos a una matriz m de dos dimensiones (*nf* filas y *nc* columnas), y a continuación escriba las sumas correspondientes a las filas de la matriz.

1. Declaramos la matriz, así como las variables necesarias.

```
Dim m( ) As Double  'Matriz dinámica de tipo doble
Dim i As Integer, j As Integer  'Índices para acceder a los elementos de la matriz
Dim nf As Integer, nc As Integer  'Variables que especificarán el número de filas y columnas de la matriz
```

2. Especificamos el número de filas y columnas para redimensionar la matriz

```
nf = InputBox("Introduce el número de filas de la matriz")
nc = InputBox("Introduce el número de columnas de la matriz")
ReDim m(1 To nf, 1 To nc)
```

3. Damos valores a los elementos de la matriz con dos bucles anidados **For …Next**

```
For i = 1 to nf
    For j = 1 to nc
      m(i, j) = InputBox( "m(" & i & ", " & j & ") = " )
    Next j
Next i
```

4. Calculamos la suma de los valores de las filas y la suma de los valores de filas y los escribimos en la primera columna de la tabla Excel.

```
For i = 1 To nf
    SumaFila = 0  'Se reinicia la variable
    For j = 1 To nc
      SumaFila = SumaFila + m(i, j)
    Next j
      Cells(i, 1).Value = "Suma de la fila " & i & " = " & SumaFila
Next i
```

Ejercicio 17

Abrir el libro de CM_E17.xlsx. Hacer una macro para guardar los datos de las columnas A, B, y C en una matriz. Las dimensiones de la matriz se las daremos con dos InputBox. En la columna E, se calcularán las sumas de cada fila.

Ejercicio 18

Abrimos el libro CM_E18.xlsx. Vamos a hacer dos macros;

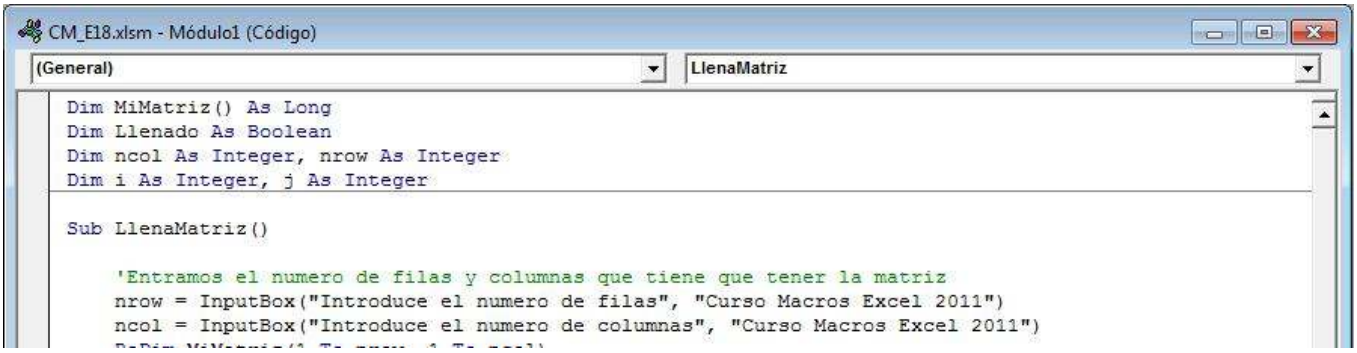
-Una primera macro, llamada "*LlenaMatriz*", que coja los datos de las columnas A, B, y C y los guarde en una matriz (podemos utilizar el mismo código que en el ejercicio17). La matriz la definiremos fuera de la Macro, en la sección de declaraciones del Módulo, así no se borrará cuando termine la macro.

-Una segunda macro, llamada "*DetectaCambios*", la cual dibujará en rojo cualquier celda a la que le hayamos cambiado el valor después de ejecutar "*LLenaMatriz*"

Nota; para pintar una celda lo haremos de la siguiente manera; [cells(i,j) es la celda que queremos pintar].

```
Dim Rango1 As Range
Set Rango1 = Cells(i, j)
Rango1.Interior.Color = vbRed
```

Nota; para evitar que las variables se borren cada vez que ejecutemos una macro, las definiremos todas en la sección de declaraciones del Módulo (fuera de cualquier macro).



Ejercicio 19. Construir y llenar una matriz de pesos para un análisis estadístico

Abrimos el libro CM_E19.xlsx. En el vemos datos de 3 columnas; X, Y, y Z. Estos datos corresponden a datos experimentales (columna Z) de localizaciones espaciales (determinadas por las columnas X, e Y). Unos de los pasos previos a la realización de un análisis de autocorrelación espacial es la de generar una matriz de pesos ($W_{(i,j)}$). Esta matriz W, es una matriz cuadrada de dimensiones [1 To n], [1 To n] (siendo n el número de datos). Los valores de los elementos de esta matriz serán $1/d_{i-j}$(siendo d_{i-j} la distancia entre los elementos i y j). Es decir, se compara cada dato experimental con el resto de datos, y se le asigna un peso proporcional a la distancia al mismo.

Realizar un programa que calcule y genere esta matriz de pesos ($W_{(i,j)}$).

Escribir la matriz completa en la Hoja2.

Consejos;

1. Crea 3 matrices para guardar los datos de X, Y, y Z.
2. Redimensiónalas (1 To n, 1 To n) siendo n el número de datos. Y toma los datos de las columnas A, B, y C.
3. Para acceder a los elementos de la matriz haz dos bucles For … Next anidados (contadores i y j).
4. Para calcular la distancia entre dos elementos (**i** y **j**) partiendo de sus coordenadas X e Y (proyectadas), y suponiendo que tenemos tres matrices X(), Y(), y Z() con los valores, podemos utilizar la ecuación;

$\left(d = \sqrt{(\Delta x)^2 + (\Delta y)^2}\right)$ ⟶ *d= Sqr(((X(j) - X(i)) ^ 2) + ((Y(j) - Y(i)) ^ 2))*

PARTE III.
PROGRAMACIÓN ORIENTADA A OBJETOS.
OBJETOS DE EXCEL

4. INTRODUCCIÓN A LOS OBJETOS

4.1 La programación orientada a objetos

Este curso no pretender dar una introducción exhaustiva de la programación orientada a objetos (POO), sino más bien una serie de indicaciones que nos ayudarán a comprender los objetos propios de MS Excel y trabajar mejor con ellos.

La POO no es un lenguaje de programación en sí, es un paradigma, un modelo de programación, una forma de programar. Es una forma de intentar llevar el código al mundo real. La POO usa objetos y sus interacciones, para diseñar aplicaciones y programas informáticos. Está basada en varias técnicas, incluyendo herencia, abstracción, polimorfismo y encapsulamiento (que no veremos en este curso).

La programación tradicional se basa en el concepto de programación estructurada o secuencial. La resolución de problemas consiste en descomponer el problema en sub-problemas y más sub-problemas hasta llegar a acciones muy simples y fáciles de codificar.

En la POO vamos a descomponer la realidad en objetos; vamos a fijarnos no en lo que hay que hacer en el problema, sino en cuál es el escenario real del mismo, y vamos a intentar simular ese escenario en nuestro programa. Se van a crear objetos con características y comportamientos específicos, los cual utilizaremos para resolver nuestros problemas.

Por ejemplo, MS Excel tiene sus propios objetos con sus propiedades, métodos y eventos propios. Nosotros no tendremos que preocuparnos por el código de estos objetos, simplemente utilizaremos los mismos.

4.2. Concepto de objeto. Propiedades – Métodos - Eventos

Un **objeto** se puede definir como una encapsulación genérica de DATOS y procedimientos para manipular los mismos. Un Objeto tiene una serie de propiedades y métodos, y sobre él pueden ocurrir determinados eventos.

Aunque en este curso no vamos a trabajar con clases, ni con módulos de clase, vamos a introducir una serie de definiciones que ayudarán a entender mejor la POO.

Una **clase** será la descripción genérica de un objeto, pero no un objeto concreto. Para hacernos una idea, un objeto sería un flan y una clase el molde para hacer flanes. Imaginemos que tenemos una clase llamada "*coche*", esta clase será la descripción genérica de un coche, como un molde, pero no un coche concreto. Este molde nos dirá que el coche deberá tener una serie de propiedades como color, marca, tipo, numero de puertas, marchas, tipo de motor, etc. Un objeto será un coche en concreto; imaginemos un Seat Ibiza Rojo, de 5 puertas, con 110 cv, etc.

Como hemos apuntado anteriormente, un objeto va a tener una serie de propiedades, métodos, y eventos.

- ***Propiedades.***

Cualquier 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. Aunque ya lo veremos más adelante, para referirnos a las propiedades de un objeto, lo hacemos de la siguiente manera;

NombreObjeto.**Propiedad** = *Valor*

- ***Métodos.***

La mayoría de 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. Para invocar un método, lo hacemos de la siguiente manera;

NombreObjeto.**Metodo**

Propiedades:

- ***Color*** = Rojo
- ***Llantas*** = Aleación
- ***ABS*** = Si
- ***Motor*** = 1900 cc

Metodos:

- ***Mover_Volante*** (Coche cambia de dirección)
- ***Acelerar*** (Coche acelera)
- ***Luces*** (Luces del coche se encienden)

- ***Eventos***

Los eventos hacen referencia a aquellas acciones que puedan ocurrir sobre el objeto. Podemos definir el comportamiento del objeto cuando suceda tal acción o evento.

4.3. Declaración de una variable objeto

En VBA declararemos una variable objeto de la misma manera que declaramos una variable normal de otro tipo, con la salvedad de que tendremos que especificar el tipo de objeto.

Por ejemplo, el objeto de Excel llamado "*Range*" (visto anteriormente) se usa para controlar rangos de celdas dentro de una hoja de Excel. Veremos cómo definir un variable de tipo Range.

```
Dim MiRango As Range
```

Una vez definida la variable de tipo "*Range*", es necesario asignarle un contenido. Es lo mismo que cuando declaramos una variable, hay que darle un valor. Para las variables objeto esto se hace de manera un poco diferente a las demás variables. Utilizaremos la palabra clave **Set.**

Set *VariableObjeto* = [Referencia a objeto]

Set *MiRango* = ActiveSheet.Range("A1:B10")

Ahora nuestra variable MiRango está haciendo referencia a un rango concreto de una hoja concreta (celdas A1 a las B10 de la hoja activa).

Ahora podremos utilizar las propiedades y métodos de la variable MiRango, y esto se traducirá a que las propiedades de las celdas A1:B10 se modificarán. Para acceder tanto a propiedades como a métodos, utilizamos un punto entre el nombre de la variable objeto y el nombre de la propiedad o método. Veamos un ejemplo.

```
Dim MiRango As Range   'Definimos una variable objeto de tipo Range
Set MiRango = ActiveSheet.Range("A1:B10")   'Asignamos un rango de celdas concreto a la variable objeto
MiRango.Value = "Hola"   'Utilizamos una propiedad del objeto Range
MiRango.Interior.Color = vbCyan   'Utilizamos otra propiedad del objeto Range
```

Por si no nos hemos dado cuenta, hemos utilizado el objeto *ActiveSheet*, que hace referencia a la hoja activa. Y este objeto *ActiveSheet* tiene a su vez una propiedad que es *Range* (que es a su vez un objeto). Hay muchas propiedades de objetos que nos devuelven objetos a su vez. Veremos esto con más detalle en los siguientes puntos.

4.4. Objetos propios de Excel

Repasemos a continuación todos estos conceptos pero ahora desde el punto de vista de algunos de los objetos que nos encontraremos en Excel como *WorkSheet* (Objeto hoja de cálculo) o *Range* (Objeto celda o rango de celdas. Un objeto *Range* está definido por una clase donde se definen sus propiedades. Recordemos que una propiedad es una característica, modificable o no, de un objeto. 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, recordemos que los métodos sirven llevar a cabo una acción sobre un objeto. 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,...

- ***For Each … Next***

Una instrucción muy útil con objetos y colecciones de objetos es **For Each …Next**. Esta instrucción repite una serie de instrucciones para cada elemento de una matriz o colección de objetos. Su sintaxis es muy parecida a **For …Next**;

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

- ✓ ***Elemento;*** variable que se utiliza para iterar por los elementos del conjunto o grupo de elementos. Elemento solo puede ser una variable de tipo Variant, una variable de objeto genérica o cualquier variable de objeto especificada.
- ✓ **Grupo;** nombre de un conjunto de objetos o de una matriz

La entrada al bloque **For Each** se produce si hay al menos un elemento en *grupo*. Una vez que se ha entrado en el bucle, todas las instrucciones en el bucle se ejecutan para el primer elemento en *grupo*. Después, mientras haya más elementos en *grupo*, las instrucciones en el bucle continúan ejecutándose para cada elemento. Cuando no hay más elementos en el *grupo*, se sale del bucle y la ejecución continúa con la instrucción que sigue a la instrucción **Next**.

Se pueden colocar en el bucle cualquier número de instrucciones **Exit For**. Esta instrucción nos permite salir del bucle en cualquier momento.

Con un objeto de tipo *Range* en Excel, podemos utilizar esta instrucción para movernos por cada uno de sus elementos. Cuando definimos un objeto de tipo rango y lo referenciamos a un rango concreto, por ejemplo;

```
Dim MiRango As Range
Set MiRango = ActiveSheet.Range("A1:C20")
```

En realidad podemos ver ese rango como un conjunto de rangos más pequeños (como si lo descompusiéramos). Podríamos utilizar **For Each …Next** para movernos por cada celda (rango unidad) de ese rango;

```
Dim Rg As Range
For Each Rg In MiRango
    instrucciones
Next Rg
```

Vamos a ver un ejemplo con objetos de Excel, vamos a rellenar las 40 filas de las columnas A, B y C con números aleatorios entre 100 y 600.

```
Dim NumAl as Integer    'Variable que guardará el número aleatorio
Dim MiRango As Range    'Variable objeto que referenciará el rango A1:C40 (40 primeras filas de A, B, y C)
Dim Rg As Range    'Variable objeto para referenciar cada rango unidad en MiRango (cada celda)

Set MiRango = ActiveSheet.Range("A1:C40")    'Referenciamos la variable MiRango a un rango concreto
For Each Rg In MiRango    'Para cada elemento de tipo Range que haya en el rango MiRango…
    Randomize    'Se inicializa  el generador de números aleatorios basado en el reloj del sistema
    NumAl = Int((600 – 100 + 1) * Rnd + 100)    'Genera un número aleatorio entero entre 100 y 600
    Rg.Value = NumAl
Next Rg
```

FunciónRnd

En este último ejemplo hemos utilizado la función ***Rnd***. Esta función devuelve un número aleatorio mayor o igual a 0 pero menor de 1. Para generar un número entero aleatorio entre dos límites especificados, haremos;

MiNumero = Int ((limiteSuperior – LimiteInferior + 1) * **Rnd** + LimiteInferior)

- ***With … End With***

Otra instrucción importante a utilizar con objetos y con propiedades de los mismos es **With… End With**. Esta instrucción tiene la siguiente sintaxis;

```
With NombreObjeto
.propiedad = valor
.propiedadn = valor
End with
```

- ✓ ***Nombre Objeto***, hace referencia a un objeto que hayamos definido previamente
- ✓ ***propiedad1, propiedadn***; hacen referencia al nombre de propiedades o métodos del objeto.

Esta instrucción es muy útil cuando trabajamos con objetos de objetos de objetos, y necesitamos escribir mucho código en cada línea. Por ejemplo, si queremos modificar la fuente de un rango de celdas;

```
With Worksheets("Hoja1").Range("A1:A10").Font
    .Name = "Arial Rounded MT Bold"
    .Size = 10
    .Bold = True
    .Italic = True
    .Color = RGB(39, 179, 146)
End With
```

Vemos que no tenemos que volver a escribir *"Worksheets("Hoja1").Range("A1:A10").Font"* para cada propiedad del objeto Font.

Función RGB()

En el último ejemplo, hemos visto que el color lo hemos definido con la función RGB. Esta función devuelve un valor (tipo Long) que especifica un color de 24 bits resultado de la mezcla de partes de Rojo (Red), Verde (Green) y Azul (Blue). Su sintaxis es;

Valor = **RGB**(*Red, Green, Blue*)

- ✓ ***Valor;*** es el valor (tipo Long) retornado por la función, especifica un color de 24 bits
- ✓ ***Red, Green, Blue;*** valores para las partes de Rojo, Verde, y Azul. Estos valores irán de 0 a 255 (8 bits)

➢ ***Objetos de Objetos.***

Es muy habitual que una propiedad de un objeto sea otro objeto. Siguiendo con el ejemplo del coche, una de sus propiedades es el motor, y el motor es un objeto con propiedades como cubicaje, caballos, número de válvulas, etc. y métodos, como aumentar_revoluciones, coger_combustible, mover_pistones, etc.

En Excel, el objeto ***WorkSheets*** tiene la propiedad ***Range*** que es un objeto, ***Range*** tiene la propiedad ***Font*** que es también un objeto y ***Font*** tiene la propiedad *Bold*(negrita). Ten esto muy presente ya que utilizaremos frecuentemente Propiedades de un objeto que serán también Objetos. Dicho de otra forma, hay propiedades que devuelven objetos, por ejemplo, la propiedad **Range** de un objeto **WorkSheet** devuelve un objeto de tipo **Range**.

Ejercicio 20.

Hacer una macro que cambie el formato del texto de las celdas A1-A10. Cambiar la fuente a Arial, negrita y cursiva. El tamaño de la fuente nos lo preguntará con una InputBox, y para el color (tanto del texto como de fondo) nos preguntará los valores R, G, y B del color deseado mediante InputBox (recordad qué los valores que introduzcamos tienen que estar entre 0 y 255).Para los valores RGB de este ejemplo podemos utilizar;

	Rojo	*Verde*	*Azul*
Texto (Azul Oscuro)	*33*	*89*	*103*
Fondo (Verde claro)	*216*	*228*	*188*

5. MODULOS Y FORMULARIOS

5.1. Concepto de Módulo

A estas alturas del curso ya tenemos que estar familiarizados con lo que es un módulo. Al crear una macro en MS Excel, vemos que en el editor de VBA se crea automáticamente un "Módulo1" donde se ha creado nuestra Macro. Un Módulo es el lugar físico donde escribimos nuestro código, donde definimos nuestras variables, donde creamos nuestras funciones (públicas o privadas), etc. Hay varios tipos de módulos, por ahora nosotros solo hemos visto los módulos estándar (donde estamos escribiendo nuestras macros).

5.2. Tipos de Módulos

- ***Módulos estándar***

Este tipo de módulo solo contiene declaraciones de funciones y procedimientos, así como variables y variables tipo. Es el tipo de módulo con el que hemos estado trabajando desde el principio del curso. Las declaraciones de variables, procedimientos y funciones en este tipo de módulo pueden ser públicas (por defecto, cualquier procedimiento de cualquier otro módulo puede acceder a ellas) o privadas (especificando **Private** en vez de **Dim**, solo se puede acceder a ellas desde dentro del mismo módulo).

- ***Formularios***

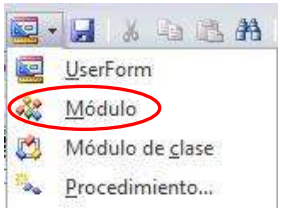
Los formularios son un tipo especial de módulos que proporcionan al usuario una interfaz de entrada-salida de datos.

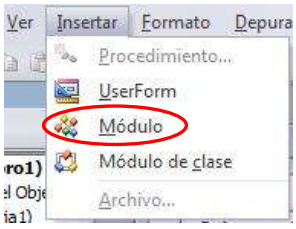
- ***Módulos de clase***

Son un tipo de módulo especial que se usan para definir clases de objetos. En este curso no veremos este tipo de módulos.

5.3. Creación de módulos en VBA

Para crear un módulo en VBA, utilizaremos el botón de insertar (2º botón de la barra de herramientas) o el Menú – Insertar.





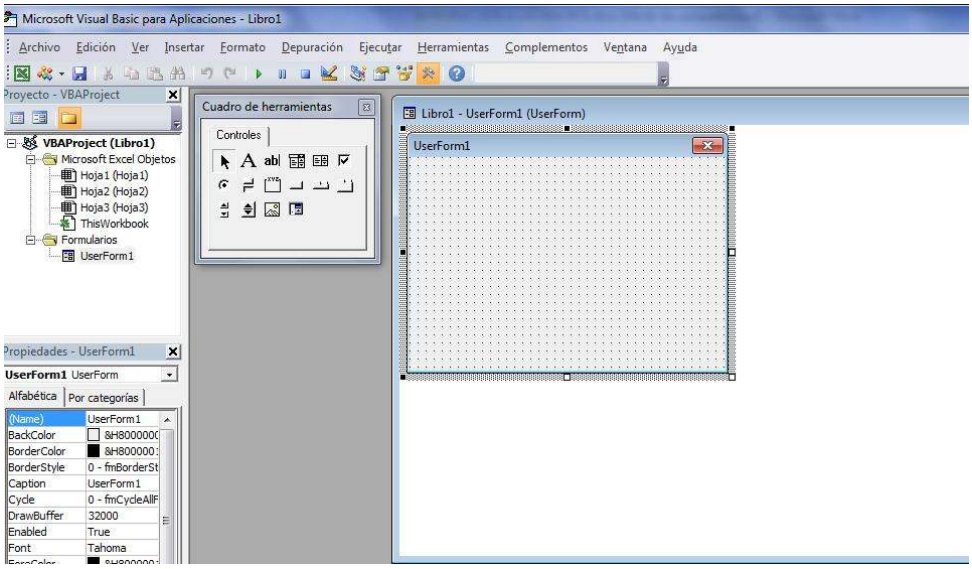
Como ejemplo vamos a crear un Módulo estándar llamado MiModulo. Abrimos un libro de Excel nuevo, pulsamos Alt-F11 y entramos en el editor de VBA. Vemos que como no hemos creado ninguna macro, no se ha creado tampoco ningún módulo estándar. Le damos a insertar Módulo, y vemos que automáticamente se crea un módulo con el nombre "Módulo1". Para cambiarle el nombre a este módulo, en la ventana de propiedades, cambiamos la propiedad de ***Name*** a MiModulo.

5.4. Trabajando con formularios

Los formularios nos permiten tener una interfaz de entrada-salida de datos. En realidad los formularios y sus controles son objetos que tienen una serie de propiedades, métodos y sobre los que pueden suceder una serie de eventos. Para entender mejor todo esto, vamos a crear un formulario llamado *MiPrimerFormulario* en el editor de VBA.

Vamos a insertar formulario, y vemos que automáticamente se crea un formulario con el nombre "*UserForm1*". Vamos a ver cada uno de los elementos que tenemos en pantalla.

Vemos que en la ventana de proyectos se ha creado una nueva carpeta llamada formularios, que contiene un elemento llamado UserForm1 (formulario de usuario). Vemos que la ventana de propiedades (inmediatamente debajo) tiene ahora las propiedades de nuestro nuevo formulario (al ser un objeto tiene propiedades, métodos y eventos). En la ventana central, vemos que se ha creado un formulario y también vemos que al lado se nos abre un cuadro con herramientas. Estas herramientas son los controles que podemos incluir en nuestro formulario; etiquetas, cajas de texto, cajas desplegables, etc.



Para cambiarle el nombre a MiPrimerFormulario, en la ventana de propiedades, cambiamos la propiedad de ***Name*** al nombre que hemos dicho (acabamos de modificar una de las propiedades del objeto formulario). ¿Por qué si le hemos cambiado el nombre, sigue poniendo en su barra de título UserForm1? Esto es debido a que la propiedad ***Caption***(etiqueta) sigue siendo "UserForm1", podemos cambiarla y ponerle lo que queramos (al contrario que Name, que no puede contener espacios ni caracteres especiales, el ***Caption*** puede incluir lo que queramos). Las propiedades principales de un formulario de usuario son;

Propiedades principales de un **UserForm** (Formulario de usuario)	
Name	Nombre del formulario. No se pueden incluir ni espacios ni caracteres especiales
Caption	Título que veremos en el formulario, como hemos dicho antes, se puede poner cualquier texto, incluyendo espacios
BackColor	Color de fondo del formulario. Por defecto es el color gris que suelen tener todos los cuadros de diálogo en Windows
ForeColor	Color de primer plano, es decir, del texto que aparezca en el formulario.
Font	Tipo de letra por defecto para el formulario
StartupPosition	Posición en la que aparecerá el formulario. Si le damos el valor "CenterScreen" el formulario saldrá en el centro de la pantalla

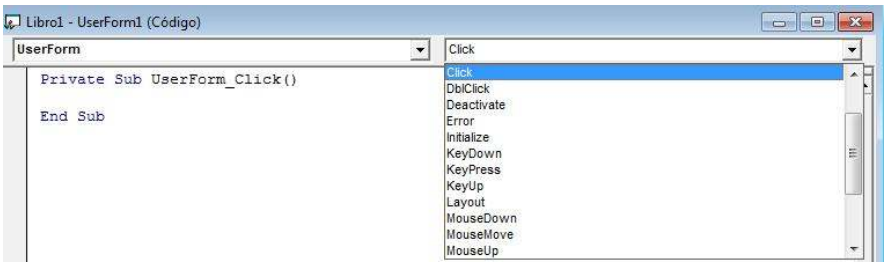
SpecialEffect	Controla el tipo de bordes del formulario
Picture, PictureAlignment, PictureSizeMode	Propiedades utilizadas por si queremos poner un gráfico de fondo en el formulario.
Height y Width	Controlan el tamaño del formulario. Podemos ver que si cambiamos el tamaño manualmente con los cuadros delimitadores de los bordes del formulario, estos valores cambian.

- ***Los formularios como objetos***

Como hemos visto, los formularios tienen una serie de propiedades que se pueden modificar directamente desde la ventana de propiedades o en tiempo de ejecución (veremos ejemplos de ello más adelante). También, sobre los formularios pueden suceder una serie de eventos (hacer clic, pulsar una tecla del teclado, mover el ratón, etc.). Estos eventos se pueden programar, es decir, se puede especificar un código para cuando suceda alguno de ellos. Vamos a ver algunos de los eventos de un formulario; para ello hacemos doble clic dentro del formulario, con esto entramos a la ventana de código del mismo (similar a las ventanas de código de los módulos estándar).

Vemos que automáticamente se ha creado un Sub que pone; **PrivateSub** UserForm_Click(). Ahi podremos escribir el código que se ejecutará cada vez que hagamos clic en el formulario.

Para ver todos los eventos podemos mirar en los cuadros desplegables arriba de la ventana de código; la primera caja nos enumera todos los objetos y controles dentro de nuestro formulario (ahora mismo solo hay un UserForm), la segunda sus eventos (vemos eventos como Click, DbClick, KeyDown, KeyPress, etc.)



Para utilizar un formulario en Excel, tenemos que cargarlo y visualizarlo. Esto lo podemos hacer creando una macro que llame al formulario (o sea, que lo cargue y visualice), o creando un botón que lo cargue cuando pulsemos.

Para practicar, vamos a ver las dos opciones;

1. Abrimos un libro nuevo de Excel, y creamos una macro que se llame CargandoFormulario

2. Una vez en el editor, añadimos un formulario (le dejamos el nombre por defecto).

3. En el código de nuestra macro escribimos lo siguiente;

```
Sub CargandoFormulario()
    UserForm1.Caption = "Este es mi primer formulario"
    UserForm1.BackColor = &HC0FFC0
    Load UserForm1
    UserForm1.Show
End Sub
```

4. Vamos a Excel y ejecutamos la macro.

En este ejemplo hemos visto una cosa importante; que las propiedades de un formulario se pueden modificar en tiempo de ejecución, es decir, mientras se está ejecutando el programa. En nuestro ejemplo hemos cambiado las propiedades de ***Caption*** (etiqueta) y ***BackColor*** (color de fondo). Vemos que para modificar estas propiedades se utiliza la sintaxis ya vista;

NombreObjeto.**propiedad** = *valor*

Vamos ahora a crear un botón que cargue nuestro formulario

1. Vamos a la ventana de hoja, ficha de programador, insertar, botón.
2. Creamos un botón en la celda B3, al crearlo se abrirá la ventana de macros con el nombre de una nueva macro "Botón1_Haga_clic_en", le damos a Nuevo, y entramos de nuevo en el editor. Esta es la macro que se ejecutará cuando hagamos clic en nuestro botón. Como queremos que se ejecute la macro anterior, llamamos a la macro "CargandoFormulario" desde esta nueva macro con la instrucción Call.
3. El código quedará de la siguiente forma;

```
Sub Botón1_Haga_clic_en()
    Call CargandoFormulario
End Sub
```

- ***Controles más comunes para formularios***

Los controles del "Cuadro de herramientas" son los típicos de cualquier cuadro de diálogo que podemos ver en cualquier aplicación de Windows:

- ✓***Etiqueta:*** Muestra un texto en un cuadro. El usuario no puede modificarlo
- ✓***Cuadro de texto***: para que el usuario introduzca un texto.
- ✓***Lista desplegable***: el clásico control de persiana con varias opciones.
- ✓***Cuadro de lista***: cuadro con distintas opciones de una lista
- ✓***Casilla de verificación***: para marcar verdadero o falso
- ✓***Botón de opción***: permiten al usuario seleccionar una opción entre varias.
- ✓***Botón de comando***: botón, como los típicos de "Aceptar" y "Cancelar"
- ✓***Marco:*** para agrupar varios controles. Uso básicamente estético

- ***Cajas de Texto (TextBox), Etiquetas (Label), y Botones (CommandButton)***

Vamos a practicar un poco con algunos de los controles más comunes para formularios, en concreto con cuadros de texto, etiquetas y botones. Abrimos un libro nuevo de Excel, vamos al editor de VBA, e introducimos un formulario nuevo. Le pondremos de nombre "Formulario1", y en el ***Caption***, cambiaremos a "Mi Formulario". Vamos a reducir un poco el tamaño de nuestro formulario, para ello tan solo haciendo clic en los cuadrados del borde.

Seguidamente vamos a crear un cuadro de texto que nos permitirá introducir valores con el teclado. Para ello pulsamos en el UserForm para que aparezca el "*Cuadro de Herramientas*" y en éste pulsamos sobre el control "*Cuadro de Texto*". Una vez seleccionado el "*Cuadro de Texto*" pulsamos en cualquier parte del UserForm y veremos que aparece un cuadro de edición. Podemos utilizar el ratón para moverlo y cambiarle el tamaño.

Pulsando con el ratón, seleccionaremos el cuadro de edición, y también podemos ver que en la ventana de "Propiedades" aparecen todas las propiedades del objeto "*Cuadro de Texto*" (*TextBox*en inglés), entre las que vamos a destacar;

Propiedades principales de un **TextBox** (Caja de texto)	
Name	Nombre del objeto. No se pueden incluir ni espacios ni caracteres especiales
Text	Texto que va dentro del cuadro de texto, el escrito por el usuario
TextAling	Alineación del texto dentro de la caja.
MultiLine	Propiedad que indica si el cuadro es multi-línea, sus valores posibles son verdadero y falso.

A continuación crearemos una etiqueta que ofrezca un texto descriptivo para nuestro formulario. Para ello seleccionamos el control "Etiqueta" del cuadro de herramientas y lo colocamos en el formulario. Utilizando el ratón, colocaremos nuestra etiqueta justo encima del cuadro de texto creado anteriormente. En la ventana de propiedades cambiaremos el ***Caption*** a "Introduce un valor".

Propiedades principales de un **Label** (Etiqueta)	
Name	Nombre del objeto. No se pueden incluir ni espacios ni caracteres especiales
Caption	Texto que mostrará la etiqueta
Font	Tipo de letra por defecto para el formulario
ForeColor	Color del texto que se mostrará en la etiqueta
SpecialEffect	Controla el tipo de bordes de la etiqueta

Por último crearemos un botón de Aceptar. Al pulsar el botón de aceptar se introducirá el valor que hemos escrito en el cuadro de texto, en la celda activa de la hoja activa. Cambiamos el ***Caption*** a "Aceptar". Por ahora no cambiaremos el nombre de los objetos; es decir, nuestro objeto caja de texto se llama "*TextBox1*", nuestra etiqueta "*Label1*" y nuestro botón "*CommandButton1*" (aceptar).

Propiedades principales de un **CommandButton** (Botón de comando)	
Name	Nombre del objeto. No se pueden incluir ni espacios ni caracteres especiales
Caption	Texto que mostrará el botón
Font	Tipo de letra por defecto para el botón
ControlTipText	Texto de ayuda que se mostrará cuando dejemos el ratón encima del botón
Locked	Controla si el botón se puede pulsar (false) o no (true)

El formulario debe de tener el mismo aspecto que la siguiente imagen;

A continuación escribiremos el código necesario para que al pulsar aceptar, el valor introducido en la caja de texto, se introduzca en la celda activa del libro activo.

Hacemos doble clic sobre el botón de aceptar, y automáticamente vamos a la parte del código del formulario, entre *Private Sub CommandButton1_Click()* y *End Sub*. Este es el código que se ejecutará cuando suceda el evento "hacer clic" sobre el botón. Para referirnos al valor que haya en el cuadro de texto, utilizaremos su propiedad ***Text.***

```
Dim Valor As Double
Valor = TextBox1.Text
ActiveCell.Value = Valor
```

Ya hemos creado nuestro primer formulario. Para probarlo, hacemos clic en el botón de ejecutar (play), o pulsamos **F5**. Vemos que el valor que introducimos en el cuadro de texto pasa a ser el de la celda activa.

Modifiquemos un poco el programa, introduzcamos un par de líneas más para que al hacer clic, la celda activa se desplace una fila hacia abajo, y el texto de la caja de texto se borre.

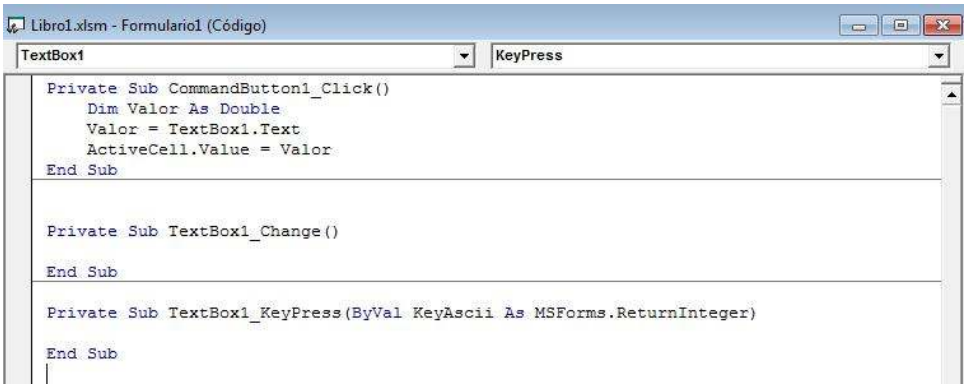
```
ActiveCell.Offset(1,0).Select
TextBox1.Text = ""
```

Por último, vamos a añadir algo más de código. Vamos a impedir que se puedan escribir caracteres que no sean numéricos en la caja de texto. Para ello utilizaremos el evento *KeyPress* sobre el cuadro de texto.

En la vista de código, en los dos cuadros desplegables de arriba, seleccionamos en el primero de ellos *TextBox1* (nuestro objeto caja de texto) y en el segundo *KeyPress*. Vemos que al seleccionar TextBox1 en la primera lista desplegable, VBA nos crea un sub con el evento *Change* del TextBox1, no pasa nada, puesto que no vamos a escribir nada en ese evento (podemos seleccionarlo y borrarlo). Al seleccionar el evento *KeyPress* de la segunda caja desplegable, VBA nos crea automáticamente un procedimiento con el código para este evento;

```
Private Sub TextBox1_KeyPress (ByVal KeyAscii As MSForms.ReturnInteger)  End Sub.
```

No os preocupéis si no entendéis esto ahora, cuando se vean los procedimientos y funciones, todo quedará mucho más claro.



Vemos que este procedimiento, a diferencia de los otros, tiene un argumento; KeyAscii. Este argumento hace referencia al código ASCII de la tecla pulsada. En este ejemplo utilizaremos este argumento para ver si se ha pulsado un número o no. Si el código ASCII coincide con un número, coma, o punto, se escribe en el cuadro de texto; sino es ninguno de esos caracteres, se cancela la pulsación. Para esto escribiremos el siguiente código dentro del procedimiento;

```
Private Sub TextBox1_KeyPress(By ValKeyAscii As MSForms.ReturnInteger)
    Dim CharacterP As String        'Definimos una variable que guardará el carácter de la tecla pulsada
CharacterP = Chr(KeyAscii)          'Con la función Chr() convertimos el código ASCII en un carácter
If InStr("1234567890,.", CharacterP) = 0 Then    'Si el carácter no es un número, punto, o una coma …
        KeyAscii = 0     'Se cancela la pulsación
    End If
End Sub
```

Para este ejemplo hemos utilizado dos funciones más; ***Chr()*** e ***InStr().*** Las describiremos brevemente;

Función Chr

Devuelve un valor String que contiene el carácter asociado con el código de carácter especificado. Su sintaxis es la siguiente;

carácter = **Chr** (*codigocaracter*)

- ✓ ***carácter;*** carácter Ascii retornado por la función (variable string de un solo carácter)
- ✓ ***codigocaracter***; código Ascii del carácter devuelto por la función Chr

Función InStr

Esta función devuelve un tipo de dato (Long) que indica la primera aparición de la una cadena en otra. Su sintaxis es la siguiente;

valor = **InStr***([start,] cadena1, cadena2 [, compare])*

- ✓ ***valor;*** valor retornado por la función InStr indicando la primera aparición de la cadena2 en la cadena1
- ✓ ***start***; (Opcional). Posición inicial para cada búsqueda. Si start contiene un valor Null se produce un error.
- ✓ ***cadena1***; Cadena de texto en la que se busca
- ✓ ***cadena2;*** Cadena buscada
- ✓ ***compare;*** (Opcional). Especifica el tipo de comparación de cadena.

Los posibles valores que puede devolver esta función son;

Si	La función InStr devuelve
cadena1 es de longitud cero	0
cadena1es Null	Null
cadena2 es de longitud cero	start
cadena2 es Null	Null
cadena2 no se encontró	0
cadena2 se encontró dentro de *cadena1*	Posición en la que se halla la coincidencia

VB tiene algunas funciones muy interesantes para manejar cadenas de texto como Left, Right, InStr, LCase, UCase, etc. Si alguno va a trabajar con cadenas de texto, recomiendo que les eche un vistazo con tranquilidad.

Con esto hemos terminado nuestro primer ejemplo de formulario. Para terminar, incluiremos un botón en la hoja de Excel que llame y cargue el formulario. Guardaremos este ejemplo como "Mi Primer Formulario.xlsm"

➢ *Cuadros combinados (ComboBox)*

Un cuadro combinado o lista desplegable es un cuadro que nos permite elegir el texto que va a contener mediante una lista que se despliega a modo de persiana. Sus propiedades más importantes son;

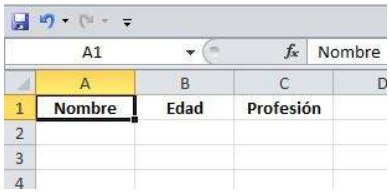
Propiedades principales de un **ComboBox** (Cuadro combinado)	
Name	Nombre del objeto. No se pueden incluir ni espacios ni caracteres especiales
Text	Texto de la combo. Se puede escribir o elegir de una ComboBox.
Value	Valor del elemento de la ComboBox
Sorted	Propiedad que determina si la lista esta ordenada. Puede ser True o False
AddItem	Método para añadir valores a la ComboBox.
RemoveItem	Método para eliminar un elemento de la lista desplegable
Clear	Método para limpiar los valores de la ComboBox

Una ***ComboBox*** se puede utilizar para seleccionar un texto de una lista. Con el método *AddItem* podemos añadir elementos a la Combo. Imaginemos que tenemos una ***ComboBox*** llamada Combobox1, con el siguiente código podemos añadirle como elementos los días de la semana;

Combobox1.AddItem "Lunes"
Combobox1.AddItem "Martes"
Combobox1.AddItem "Miercoles"
Combobox1.AddItem "Jueves"
Combobox1.AddItem "Viernes"

Como ejemplo vamos a crear un formulario de entrada de datos en una tabla Excel.

1. Abrimos un nuevo libro de Excel, vamos a poner cabeceras en la primera fila; Nombre, Edad, Profesión.



2. Vamos al editor del VBA (Alt-F11) e insertamos un nuevo formulario. En este formulario vamos a incluir los siguientes elementos;

Elemento	Nombre	Caption
Label	*label1*	*Nombre*
Label	*label2*	*Edad*
Label	*label3*	*Profesión*
TextBox	*tbNombre*	
TexBox	*tbEdad*	
ComboBox	*cbProfesión*	
CommandButton	*c_EntrarDatos*	*Guardar Dato*

CONSEJO

La fuente que nos pone VBA por defecto para etiquetas (propiedad **Font**), puede parecernos poco elegante (Tahoma, Normal, 8 puntos). Podemos cambiar a otra fuente a las etiquetas, y cajas de texto, cambiando sus propiedades **Font**.

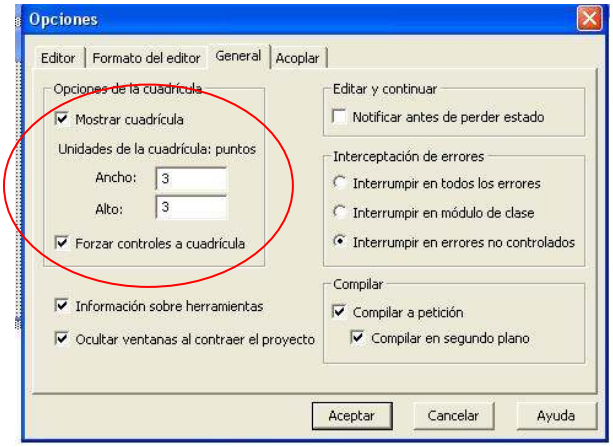
¿Tenemos que hacer este cambio para cada control? Pues la verdad es que no, porque VBA nos permite copiar controles. Al copiar y pegar un control, se copian todas sus propiedades excepto el nombre (no puede haber dos controles con el mismo nombre). Prueba a insertar el label1, cambiar su fuente, y después selecciónalo y utiliza copiar y pegar (con el botón derecho o con los botones copiar y pegar). Vemos que se crea otro control igual en el formulario, que aunque tiene la misma fuente y la misma etiqueta, tiene un nombre diferente.

CONSEJO

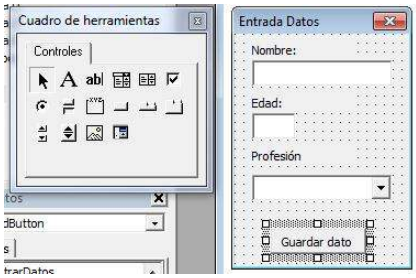
También nos habremos dado cuenta de que la rejilla del formulario es más bien gruesa (es decir, no podemos mover los controles, ni redimensionar su tamaño de una forma fina). Tenemos dos formas de variar esto;

Una manera es especificar achos y altos específicos para el control con las propiedades **Width** y **Height,** y controlar su posición dentro de un formulario con las propiedades **Top** (distancia desde el margen superior del formulario), y **Left** (distancia desde el margen izquierdo del formulario).

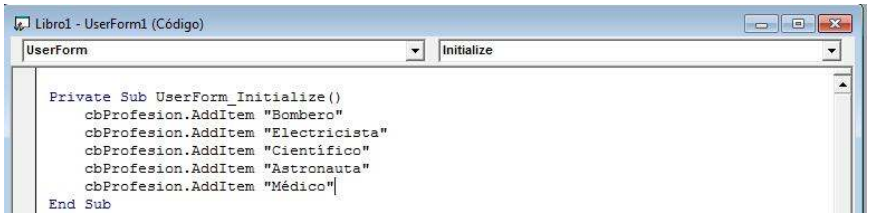
Otra forma consistirá en variar esta rejilla. Esto lo podemos hacer dentro del editor de VBA en Menu – Herrameintas – Opciones – Ficha general – y variamos las unidades de la rejilla (por defecto 6).



El aspecto de nuestro formulario deberá de quedar así;



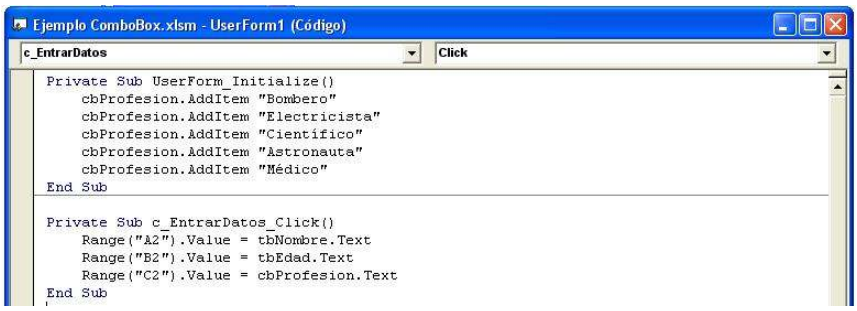
3. Ahora pasemos a escribir el código. Lo primero que haremos es llenar la comboBox con los elementos cuando se cargue el formulario. Para ello utilizaremos el evento del Formulario Initialize. En la ventana de código del formulario, en la parte superior, seleccionamos UserForm y en la siguiente lista Initalize. Entre el Sub, y en End Sub escribimos el código para llenar la combo; utilizamos el método AddItem y escribimos 4 o 5 profesiones como en el ejemplo siguiente;



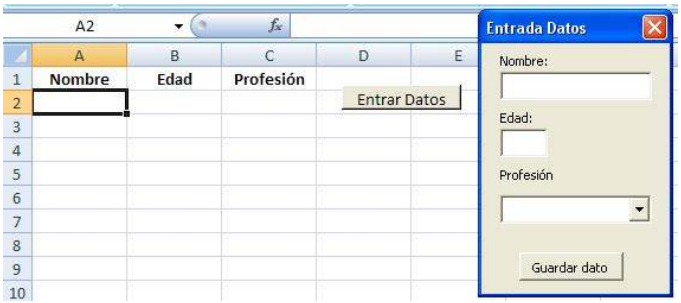
4. Una vez llena la combo, escribiremos el código para el botón de comando. Al hacer clic en él, los datos del formulario se escribirán en las columnas 1, 2 y 3 de la hoja activa. Seleccionamos en la primera lista desplegable de la ventana de código del formulario "c_EntrarDatos" (nombre de nuestro botón), y en la segunda "Click". Entre el Sub y End Sub, introducimos el siguiente código;

Range("A2").Value = tbNombre.Text
Range("B2").Value = tbEdad.Text
Range("C2").Value = cbProfesión.Text

El aspecto que tendrá la ventana de código de nuestro formulario será el siguiente;



5. Ahora en Excel, en la ficha de programador, crearemos un botón al que llamaremos "Entrar Datos". Este botón cargará y mostrará el formulario como ya hemos visto anteriormente. El aspecto final será;



6. Guarda este libro como EjemploCombo1.xlsm. Más adelante modificaremos el código para que al hacer clic, los datos se introduzcan en la siguiente fila vacía.

Ejercicio 21.

Abre el libro Excel CM_E21.xlsx. Este libro tiene dos hojas, vemos que en la Hoja2 hay nombres de especies. Crear un formulario con una **ComboBox** que se llene con los nombres de las especies de esta Hoja2. Añadir un botón al formulario que permita introducir el valor seleccionado en la combo, en la celda A1 de la Hoja1.

6. PROCEDIMIENTOS Y FUNCIONES

6.1. Concepto de Procedimiento

Un procedimiento es una porción de código que tiene un principio y un final. Puede tener un ámbito público o privado. Un procedimiento se encierra entre las palabras clave **Sub** y **End Sub**, es decir, al ejecutarse un procedimiento se ejecuta el código encerrado en él.

Una macro es en realidad un procedimiento. Se puede crear directamente un procedimiento escribiendo en un módulo **Sub** más el nombre del procedimiento, veremos que automáticamente VBA nos añade unos paréntesis detrás del nombre y nos pone un **End Sub**.

Podemos especificar el ámbito del procedimiento al crearlo usando las palabras clave **Public** o **Private**, si queremos que sea público o privado respectivamente.

Un procedimiento puede tener argumentos (son opcionales), que se integraran dentro del paréntesis que sigue al nombre del procedimiento.

Un ejemplo de procedimiento sería;

```
Public Sub MiMacro()
    MsgBox "Esto es un ejemplo de un procedimiento", vbInformation, "Mi Primer Procedimiento"
End Sub
```

Al crear un control en un formulario, los eventos que se producen en el son en realidad procedimientos.

Como hemos dicho los procedimientos pueden ser públicos o privados. Un procedimiento público se podrá llamar desde cualquier lugar (cualquier módulo o formulario), un procedimiento privado solo se podrá llamar desde el mismo módulo al que pertenece. Vamos a ilustrar este último concepto.

1. Abrimos una hoja Excel y entramos directamente en el editor de VBA. Insertamos un módulo, Módulo1 y un formulario de usuario UserForm1. En el módulo vamos a crear un procedimiento privado que nos mostrará una caja de diálogo.

```
Private Sub MostrarCaja()
    MsgBox "Esto es un ejemplo de un procedimiento", vbInformation, "Mi Primer Procedimiento"
End Sub
```

2. Ahora abrimos el formulario, insertamos un botón (CommandButton1) y pulsamos sobre él para entrar en la ventana de código. Vemos que VB nos ha creado un procedimiento privado que corresponde al evento de hacer clic sobre el botón.

3. Vamos a intentar llamar al procedimiento MostrarCaja cuando se pulse sobre el botón. Para ello escribiremos simplemente su nombre.

```
Private Sub CommandButton1_Click()
    MostrarCaja
End Sub
```

Vemos que al hacer clic sobre el botón nos dice que no se ha definido el Sub o Function. Esto sucede porque es privado. Cambia el Private por Public (**Public Sub** MostrarCaja()) ¿Ves la diferencia ahora?

4. Volvamos a definir MostrarCaja como privado (Private). Ahora crearemos otro procedimiento en el Módulo1. Lo llamaremos ProcedimientoLlamar y lo definiremos como público. Este procedimiento llamará al procedimiento MostrarCaja.

```
Public Sub ProcedimientoLlamar()
    MostrarCaja
End Sub
```

5. Solo falta modificar, que el botón tiene que llamar a este procedimiento en vez de al de MostrarCaja. Al probar el formulario, vemos que ahora no da fallo. ¿Por qué? Porque hemos llamado al procedimiento MostrarCaja desde el mismo módulo (lo ha llamado ProcedimientoLlamar, no el clic del botón).

Una instrucción importante que podemos utilizar dentro de un procedimiento es la de **Exit Sub**.

1. Vamos a crear un formulario que nos muestre una caja de diálogo 10 veces.

```
Sub MostrarCaja()
    DimiAsInteger, Respuesta As Long
    For i = 1 To 10
      MsgBox "Soy la Caja de diálogo  " & i, vbInformation
    Next i
End Sub
```

2. Al ejecutarlo vemos que debemos de pulsar sobre 10 cajas y no hay manera de salir. Modifiquémoslo;

```
Sub MostrarCaja()
    Dim i As Integer, Respuesta As Long
    For i = 1 To 10
      MsgBox "Soy la Caja de diálogo  " & i, vbInformation
      Respuesta = MsgBox("¿Deseas salir?", vbYesNo)
      If Respuesta = vbYes Then
                Exit Sub
      End If
    Next i
End Sub
```

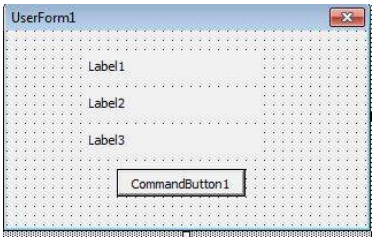
Ahora vemos que si pulsamos Si sobre el segundo cuadro de diálogo, nos salimos del procedimiento aunque este no esté acabado (aunque no hayamos llegado a la caja numero 10).

- ***Argumentos de un procedimiento***

Otro concepto importante de los procedimientos son los argumentos. Los argumentos son variables que usará el procedimiento en su código interno, se especifican entre los paréntesis.

Vamos a ilustrar esto con un sencillo ejemplo;

1. Abre un nuevo libro de Excel, inserta un formulario con 3 labels y un botón de comando;

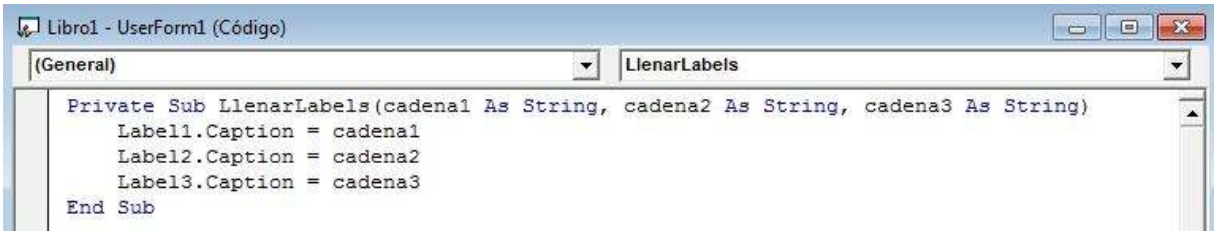


2. Ahora crearemos un procedimiento llamado LlenarLabels que le cambie el Caption a esas etiquetas. Para ello, el procedimiento tendrá 3 argumentos de tipo cadena de texto (string). Esto se especifica dentro del paréntesis detrás del nombre del procedimiento. Deberá quedar así.

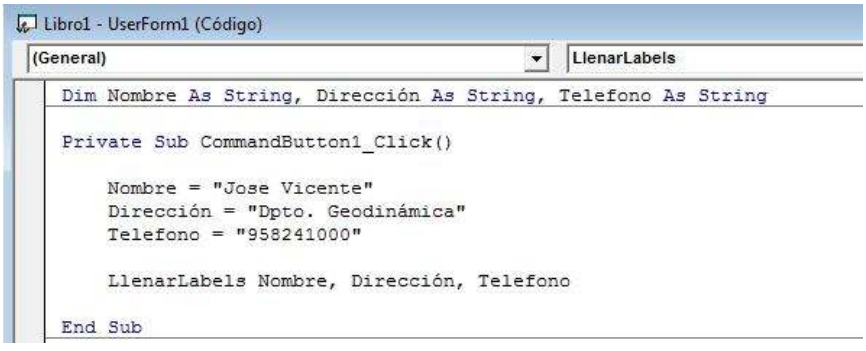
   Private Sub *LlenarLabels(cadena1* ***As String****, cadena2* ***As String****, cadena3* ***As String****)*

   End Sub

3. Ahora dentro del procedimiento podemos utilizar estas variables para cambiar los **caption** de las etiquetas;



4. Es hora de llamar a nuestro procedimiento utilizando argumentos. Nos vamos al procedimiento de clic de nuestro botón, y ahí escribimos lo siguiente;



Nos hemos dado cuenta de que al llamar al procedimiento esta vez hay que especificar cada uno de los argumentos separados por comas. Las variables Nombre, Dirección y Teléfono se han pasado como los argumentos cadena1, cadena2, y cadena3 de nuestro procedimiento. Si no se especifican los argumentos, se produce un error, puesto que son obligatorios (por defecto). Si alguna de las variables no fuera del mismo tipo que el argumento se produce un error. Probemos a cambiar el tipo de la variable Teléfono a Long ¿Qué pasa entonces?

Las variables se pueden pasar como argumentos por valor (**ByVal**) o por referencia (**ByRef**) (esta última forma es por defecto si no se especifica nada). Esto quiere decir que si se pasan las variables por referencia, si dentro del procedimiento se modifica el argumento, también se modifica la variable. Veamos un ejemplo.

5. Modifiquemos nuestro procedimiento, para que una vez utilizados los argumentos los borre. Añadimos 3 líneas;

   cadena1 = “”
   cadena2 = “”
   cadena3 = “”

```
Private Sub LlenarLabels(ByVal cadena1 As String, ByVal cadena2 As String, ByRef cadena3 As String)
    Label1.Caption = cadena1
    Label2.Caption = cadena2
    Label3.Caption = cadena3
    cadena1 = ""
    cadena2 = ""
    cadena3 = ""
End Sub
```

6. En el procedimiento de clic del commandbuton1, añadamos una línea al final para visualizar el contenido de las variables con una caja de texto;

   MsgBox Nombre
   MsgBox Dirección
   MsgBox Teléfono

7. Ahora ejecutemos el formulario, ¿Qué ocurre? Vemos que al borrar las variables cadena1, cadena2, y cadena3 (argumentos de nuestro procedimiento), también se borran las variables Nombre, Dirección, y Teléfono que hemos definido. ¿Cómo podemos evitar esto?, simplemente pasando los argumentos por valores y no por referencia. Escribamos **ByVal** delante de cadena1, y cadena2, **ByRef** delante de cadena3. Ahora ejecutemos la aplicación y demos clic al botón, ¿vemos cuál es la diferencia?

Ejercicio 22

Abrimos el libro de Excel CM_E22.xls Vamos a crear un procedimiento que coloque la selección en la primera celda vacía de una columna concreta. El procedimiento tendrá dos argumentos, el número de la columna y el nombre de la hoja. Para buscar la celda vacía podemos utilizar un bucle **Do** y la función **IsEmpty**

Do Until IsEmpty(Cells(i, Columna).Value)

Loop

Ejercicio 23

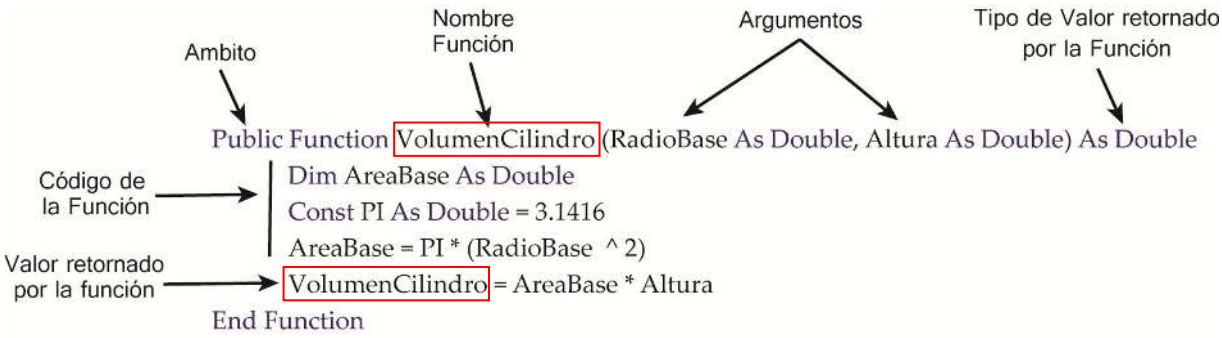
Abrimos la macro realizada en el ejercicio 21. Vamos a modificar esta última aplicación. Vamos a definir un procedimiento privados (OrdenarRango) dentro del formulario. Este procedimiento llenará la ComboBox.

6.2. Concepto de Función

Una función es como un procedimiento o subrutina, es decir una porción de código con un principio y un final. La principal diferencia entre un procedimiento y una función es que las funciones retornan un valor y los procedimientos NO. Para definir una función se utilizan las palabras claves **Function** y **End Function**.

Las funciones también tienen un ámbito que puede ser público o privado; para definirlo utilizaremos las palabras clave **Public** o **Private**. Y también al igual que los procedimientos, las funciones pueden utilizar argumentos (por valor, **ByVal**; o por referencia **ByRef**).

La definición de una función tiene las siguientes partes;



Como vemos en el gráfico de arriba, una función debe de tener dentro de su código una línea donde se le asigne el valor a la función. Esto se hace de la siguiente manera; **NombreFunción =** *Valor*.

Para llamar una función desde cualquier parte del código;

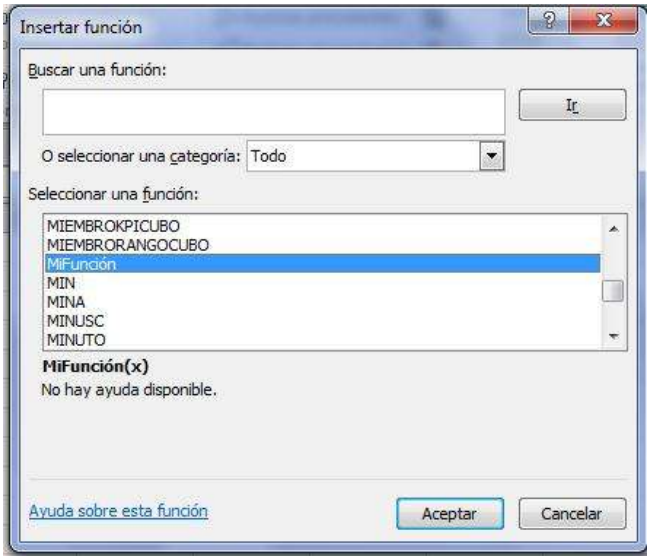
Variable = NombreFunción (arg1, arg2)

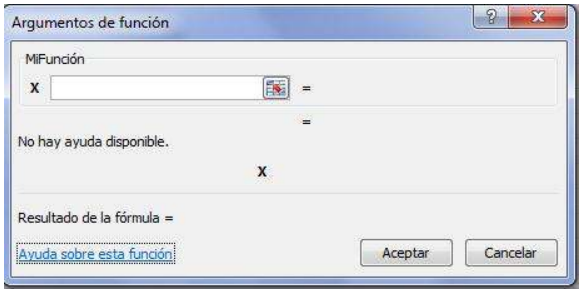
Lo mismo que los procedimientos, las funciones públicas definidas en un módulo estándar se podrán utilizar desde cualquier procedimiento o formulario. Una función pública también se puede llamar como una función de Excel.

Ejemplo, definir una función que calcule la siguiente ecuación; $f(x) = 2x^3 + \ln(x) - \frac{\cos(x)}{e^x} + sen(x)$

```
Public Function MiFunción(x As Double) As Double
    MiFunción = 2 * x ^ 3 + Log(x) - Cos(x) / Exp(x) + Sin(x)
End Function
```

Ahora volvamos a Excel, y le damos a introducir función. Vemos que se agregado una nueva función que se llama MiFunción, y que tiene un argumento que se llama X.





Ejercicio 24. Crear una función propia que calcule la covarianza de una distribución bi-dimensional

Abrimos el libro de CM_E24.xls. Vemos que en el libro hay dos columnas de datos (X e Y). Vamos a crear una función personalizada que se llame "*MiCovarianza*". Esta función calculará la covarianza de una distribución bidimensional y tendrá como argumentos un rango de valores de X, y un rango de valores de Y. Por si no nos acordamos, las fórmulas para calcular la covarianza son;

$$\sigma_{xy} = \frac{n\sum_i (x_i - \bar{x})(y_i - \bar{y})}{N} = \frac{n\sum_i x_i . y_i}{N} - \bar{x} \cdot \bar{y}$$

Una vez calculada la covarianza, comprobaremos que el resultado de nuestra función "*MiCovarianza"* es el mismo que el de la función de Excel "*Covar*".

Ejercicio 25. Crear una función que calcule el índice Moran I de auto-correlación espacial

Este ejercicio requiere una macro más compleja, pero es un ejemplo de toda la funcionalidad que podemos tener programando nuestras propias funciones. Para hacer este ejercicio, descargarse el pdf "Cálculo del índice MoranI", donde se explica paso a paso la formula y la forma de crear la Macro.

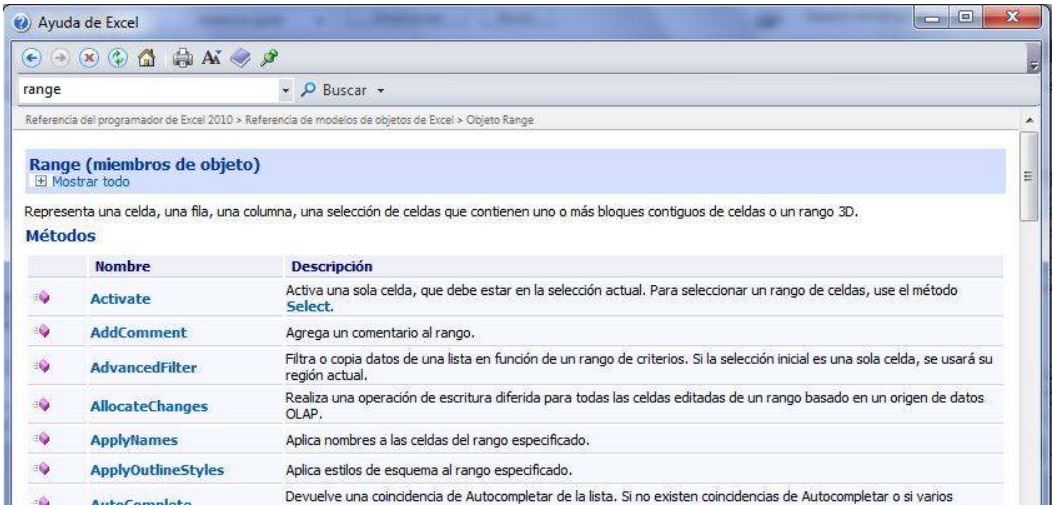
7. TRABAJANDO CON OBJETOS PROPIOS DE EXCEL

Ya hemos visto anteriormente varios objetos propios de Excel tales como ***Range***, ***WorkSheet***, ***Workbook***, etc. En esta parte vamos a ver estos objetos con más detalle para poder exprimir al máximo sus funcionalidades. Debido al carácter de este curso, las propiedades y métodos de los objetos que veamos se mostrarán de la manera más simple, es decir, no se verán todos los argumentos opcionales de muchos métodos y/o propiedades. Para una consulta más detallada podemos acudir a la ayuda del editor de VB para Excel.

➢ ***Trabajando con la ayuda de Excel***

Para muchos de los objetos de Excel, podemos acudir a su ayuda. En el editor de VBA vamos a Menu—ayuda y accederemos a la ayuda de VBA. En la ayuda referente a los objetos de Excel se nos mostrarán los miembros (propiedades y métodos) de cada objeto. La ayuda muchas veces será más exhaustiva que los códigos que se verán en la siguiente sección.

Como ejemplo, entremos en el editor de VBA. En la ayuda escribamos Range; escogemos la primera entrada, Range (miembros del objeto, puede que no sea la primera). Ahí podemos ver una descripción de los métodos y propiedades del objeto Range.



7.1. Objeto Workbook

Este objeto referencia un libro de Excel. Un objeto ***Workbook*** es un miembro de la colección ***Workbooks( )*** que hace referencia a todos los libros abiertos en la sesión actual de Excel.

ActiveWorkbook: Hace referencia al libro activo en la sesión actual

ThisWorkbook: Hace referencia al libro desde donde se ejecuta el código de la macro.

Workbooks*(Index):* Hace referencia a una colección de objetos de tipo Workbook que representa todos los libros abiertos en la sesión actual de Excel. *Index* representa una variable entera que representa el elemento de la colección (comenzando en 1).

Workbooks(1) – representa el primer libro de la colección
Workbooks("Libro1") – representa al libro "Libro1" dentro de la colección

- ***Workbooks.Open***

Método que nos permite abrir un libro. Su sintaxis es;

Workbooks.**Open(***FileName, UpdateLinks, ReadOnly, Format, Password, WriteResPassword, IgnoreReadOnlyRecommended, Origin, Delimiter, Editable, Notify, Converter, AddToMru, Local, CorruptLoad***)**

Ejemplo; abrir un libro desde una macro. Antes de nada, copiemos el libro EjemploAbrir.xlsx al directorio C:\Temp\. A continuación abriremos un libro nuevo y escribiremos la siguiente macro.

```
Sub Abrir()
    DimNombreLibro As String
    NombreLibro = "C:\Temp\EjemploAbrir.xls"
    Workbooks.OpenNombreLibro, Password:="alimoche"
End Sub
```

Con este ejemplo hemos visto una cosa muy importante; aunque hay muchos argumentos opcionales en el método open, podemos referenciar cualquiera de ellos con la sintaxis ***NombreArgumento:=*** *Valor*.

En el ejemplo vemos que el argumento *FileName* es el primero, y *Password* es el quinto, ¿qué ha pasado con los otros 3? Como son argumentos opcionales, podemos simplemente no especificarlos. Si solo queremos especificar dos argumentos (en nuestro ejemplo FileName y Password, podemos hacerlo de dos maneras;

a) Especificar el nombre de cada argumento y su valor; ***NombreArgumento:=*** *Valor*

b) Poner comas que separarán los valores de los distintos argumentos, dejando "en blanco" los argumentos que no queramos especificar.

Estas 3 líneas de código, hacen exactamente lo mismo. Compruébalo por ti mismo.

```
Workbooks.Open"C:\Temp\EjemploAbrir.xls", Password:= "alimoche"
Workbooks.OpenFileName:="C:\Temp\EjemploAbrir.xls", Password:= "alimoche"
Workbooks.Open"C:\Temp\EjemploAbrir.xls", , , ,"alimoche"
```

Al utilizar la grabadora de macros, MS Excel utiliza esta nomenclatura con mucha frecuencia.

Si quisiéramos abrir todos los libros de un directorio, podemos utilizar una matriz de tipo string con todos los nombres de todos los archivos de Excel de un mismo directorio. Para obtener todos los nombres, podemos utilizar la función Dir. Veremos un ejemplo de esto más adelante.

➢ ***Workbook.SaveAs***

Esta propiedad nos permite guardar un libro en una ruta especificada. Su sintaxis es;

*Workbook**.SaveAs(**FileName, FileFormat, Password, WriteResPassword, ReadOnlyRecommended, CreateBackup, AccessMode, ConflictResolution, AddToMru, TextCodepage, TextVisualLayout, Local**)***

- ✓ ***FileName,*** es una variable de tipo String que especifica la ruta completa y el nombre del archivo a guardar.
- ✓ ***FileFormat;*** es una constante de tipo Long (**XlFileFormat**) que especifica el formato.
- ✓ ***Password;*** es una variable tipo Variant que especifica la contraseña del libro

El siguiente ejemplo guarda el libro activo en la carpeta C:\Temp como "LibroGuardado.xlsm" (libro con macros habilitados) y con un password (alimoche)

```
Sub Guardar()
    Dim NombreLibro As String
    Dim Contraseña As String
    NombreLibro = "C:\Temp\EjemploGuardar"
    Contraseña = "alimoche"
    ActiveWorkbook.SaveAs NombreLibro, 52, Contraseña
End Sub
```

El valor de XlFileFormat para un archivo de Excel con macros habilitados (*.xlsm) es 52 (*xlOpenXMLWorkbookMacroEnabled)*

➢ ***Workbook.Close***

El método Close cierra el libro especificado (preguntará si deseamos guardar cambios).

Para evitar que la aplicación (MS Excel) muestre alertas (como preguntar si deseamos o no guardar los cambios, podemos usar la propiedad **DisplayAlerts** del objeto ***Application***.

Application.*DisplayAlerts = False*

Este ejemplo cierra el libro activo, sin preguntarnos si deseamos guardar los cambios;

```
Sub CierraLibro()
    Application.DisplayAlerts = False
    ActiveWorkbook.Close
End Sub
```

Si queremos cerrar todos los libros abiertos;

```
Workbooks.Close
```

➢ ***Workbook.Save***

Este método guarda los cambios realizados en el libro activo. En la siguiente macro, primero guardamos los cambios del libro activo, luego cerramos el libro, y por ultimo salimos de Excel.

```
Sub GuardaCierra()
    ActiveWorkbook.Save
    ActiveWorkbook.Close
    Application.Quit
End Sub
```

- ***Workbook.Add***

Este método crea un nuevo libro (nombre por defecto), lo añade a la colección Workbooks( ) y lo convierte en el libro activo. El ejemplo crea un nuevo libro

```
Sub CreaNuevo()
    Workbooks.Add
End Sub
```

Este segundo ejemplo referencia una variable-objeto de tipo Workbook que apuntará a un nuevo libro. Cambiaremos el nombre a este libro, (como su propiedad Name es de solo lectura, para cambiarle el nombre, deberemos de guardarlo con otro nombre) y lo guardaremos en "C:/Temp". Con la primera línea de código especificamos que este nuevo libro tendrá solo 2 Hojas por defecto en vez de 3 (nota, cambiar esto luego, puesto que para cada nuevo libro tendremos ya solo 2 hojas por defecto).

Application.*SheetsInNewWorkbook* = 2

```
Sub CreaNuevo2()
   Application.SheetsInNewWorkbook = 2
   Dim MiLibro As Workbook
   Dim Ruta As String, Name As String
   Set MiLibro = Workbooks.Add
   Ruta = "C:\Temp\"
   Name = "Mi Libro"
   MiLibro.SaveAs Ruta & Name
   MiLibro.Close
End Sub
```

- ***Propiedades para extraer información de un libro (propiedades de solo lectura)***

*Workbook.**Name***; nombre del libro.

*Workbook.**Path*** ; ruta del libro (solo la ruta, no el nombre).

*Workbook.**FullName***; ruta y nombre completo del libro.

- ***Workbook.Worksheets***

Esta propiedad del objeto Workbook devuelve una colección con todas sus hojas (objetos Worksheet).

- ***Worksheets(Item)***

Hace referencia a una hoja concreta (objeto Worksheet) dentro de la colección Worksheets. Item puede ser un índice, o el nombre de la hoja. Una vez que se referencia una hoja concreta, podemos acceder a sus propiedades. Las dos siguientes líneas nos muestran una MsgBox con el nombre de la segunda Hoja del libro activo.

```
MsgBox ActiveWorkbook.Worksheets(2).Name, vbinformation, "Curso Macros para Excel 2011"
MsgBox ActiveWorkbook.Worksheets("Hoja2").Name, vbinformation, "Curso Macros para Excel 2011"
```

- ***Worksheets.Add***

Este método agrega una hoja nueva a la colección Worksheets de un libro especificado. Su sintaxis es;

Worksheets.**Add(***Before*, *After*, *Count*, *Type***)**

- ✓ ***Before;*** Hoja antes de la cual se insertará la nueva hoja
- ✓ ***After;*** Hoja después de la cual se insertará la nueva hoja
- ✓ ***Count;*** Numero de hojas que se insertarán (el valor por defecto es una)
- ✓ ***Type;*** Tipo de hoja que se agregará

Si no se especifica *Before* ni *After*, la nueva hoja se agrega justo después de la hoja activa.

Esta línea inserta 3 hojas delante de la Hoja2 del libro activo

```
ActiveWorkbook.Worksheets.Add Before:=Worksheets("Hoja2"), Count:=3
```

➢ ***Worksheets.Count***

Devuelve una variable (tipo Long) con el número de objetos de la colección (número de hojas de la colección Worksheets). Su sintaxis es;

variable = Worksheets.**Count**

Con este código podemos mostrar una caja de dialogo que nos informe del número de hojas del libro activo;

```
Dim NumeroHojas As Long
NumeroHojas = ActiveWorkbook.Worksheets.Count
MsgBox "El libro active tiene " & NumeroHojas & " hojas."
```

7.2. Objeto Worksheet

Este objeto referencia una hoja de cálculo. Es un miembro de la colección Worksheets( ) que referencia la colección de todos los objetos Worksheet del libro especificado o activo.

Podemos definir una variable objeto del tipo Worksheet. En el siguiente ejemplo, vamos a referir una variable-objeto tipo Worksheet a una nueva hoja insertada en la colección Worksheets.

```
Dim NuevaHoja As Worksheet
Set NuevaHoja = Worksheets.Add(Before:=Worksheets(Worksheets.Count))
```

Con el objeto **ActiveSheet**, podemos hacer referencia a la hoja del libro activa.

➢ ***Worksheet.Name***

Devuelve una variable de tipo variant (String) con el nombre de la hoja.

variable = Worksheet.**Name**

Estas líneas nos muestra el nombre de la hoja activa.

```
Dim HojaActiva As Worksheet, Nombre As String
Set HojaActiva = ActiveSheet
Nombre = HojaActiva.Name
MsgBox "La hoja activa se llama " & Nombre, vbInformation, "Curso Macros Excel 2011"
```

➢ ***Worksheet.Visible***

Propiedad que controla si una hoja es visible o no. Sus valores pueden ser verdadero falso. Ejemplo;

```
Dim MiHoja As Worksheet
Set MiHoja = Worksheets("Hoja3")
MiHoja.Visible = False          'Ocultamos la Hoja3 del libro activo
```

- ***Worksheet.Sort***

Propiedad que devuelve los valores ordenados de la hoja actual. Se verá más adelante.

- ***Worksheet.Activate***

Método que activa la hoja especificada, convirtiéndola en la hoja activa (**ActiveWorksheet**). Con la siguiente línea de código activamos la Hoja2 del libro activo.

```
ActiveWorkbook.Worksheets("Hoja2").Activate
```

- ***Worksheet.Select***

Método que selecciona la hoja especificada. El siguiente código selecciona la tercera hoja del libro activo;

```
ActiveWorkbook.Worksheets(3).Select
```

Diferencias entre seleccionar y activar;

Cuando seleccionamos varias hojas, tan solo una de ellas puede ser la hoja activa. Podemos probar a seleccionar dos hojas en un libro de Excel haciendo clic sobre ellas mientras mantenemos pulsada la tecla Crtl. Las dos hojas pueden estar seleccionadas pero tan solo una de ellas (la que tenga su nombre en negrita) es la hoja activa. Lógicamente, cuando operamos solo sobre una hoja y la misma está seleccionada, **Select** o **Activate** es lo mismo (es decir seleccionar o activar da igual).

Seleccionando varias hojas a la vez

Con el siguiente código podemos seleccionar las hojas 1 y 2 del libro activo (siempre que no hayamos cambiado sus nombres, y sigan siendo Hoja1 y Hoja2)

```
ActiveWorkbook.Worksheets(Array("Hoja1","Hoja2")).Select
```

La función Array, devuelve una variante que contiene una matriz. Su sintaxis es;

Array(*lista_argumentos***)**

El argumento lista_argumentos requerido es una lista de valores delimitados por comas que se asignan a los elementos de la matriz contenida en el tipo Variant. El límite inferior de una matriz generada con la función Array, viene determinado por la instrucción Option Base (si no se ha especificado, será 0). Un ejemplo de su uso sería;

```
Dim Nombres As String
Nombres = Array("Juan","Gustavo","Enrique")
Msgbox nombres(0)   'Muestra una caja MsgBox con el primer nombre de la Matriz (Juan)
```

Para el siguiente ejemplo, abre un nuevo libro y añade 5 hojas. Con el código siguiente seleccionaremos la primera, la tercera, y la quinta (da igual el nombre qué tengan), y además determinaremos que la tercera hoja será la hoja activa.

```
Sub Selecciona()
    ActiveWorkbook.Worksheets(Array(1, 3, 5)).Select
    ActiveWorkbook.Worksheets(3).Activate
End Sub
```

- ***Worksheet.Delete***

Método que borra la hoja especificada. También podemos utilizar Worksheets con una array;

```
ActiveWorkbook.Worksheets(Array(3, 5)).Delete
```

- ***Worksheet.Copy*** **y** ***Worksheet.Move***

Estos dos métodos sirven para copiar y mover una hoja de cálculo. Sus sintaxis son muy parecidas;

Worksheet.**Copy(***Before, After***)**

Worksheet.**Move(***Before, After***)**

- ✓ ***Before;*** Hoja antes de la cual se copiará/moverá la hoja especificada.
- ✓ ***After;*** Hoja después de la cual se copiará/moverá la hoja especificada.

Con este ejemplo, se copia la Hoja1 al final del libro

```
Dim MiHoja As Worksheet
Set MiHoja = Worksheets(1)
MiHoja.Copy after:=Worksheets(Worksheets.Count)
```

Si utilizamos el código, vemos que la hoja se copia con el mismo nombre, pero añadiéndole un (2) detrás (para indicar que es una copia). La hoja copiada/movida se convierte en la hoja activa. Con el siguiente código, podemos cambiarle el nombre a la hoja copiada;

```
Dim MiHoja As Worksheet
Dim MiCopia As Worksheet
Set MiHoja = Worksheets(1)
MiHoja.Copy after:=Worksheets(Worksheets.Count)
Set MiCopia = ActiveSheet
MiCopia.Name = "Copia de la Hoja1"
```

- ***Worksheet.Protect*** **y** ***Worksheet.Unprotect***

Estos dos métodos sirven para proteger/desproteger la hoja especificada. Su sintaxis es;

Worksheet.**Protect(***Password, UserInterfaceOnly,***)**

Worksheet.**Unprotect(***Password***)**

No vamos a definir todos los argumentos de protect, solo dos importantes (si queréis más información acerca de éste método consultad la ayuda);

- ✓ ***Password;*** variable que identifica la contraseña para proteger/desproteger la hoja.
- ✓ ***UserInterfaceOnly;*** variable que especifica si sólo se protege la hoja de las acciones de usuario normales, no de las macros. Si su valor es true, se protegerá la hoja de las acciones del usuario, pero no de las macros.

Es muy importante que especifiquemos los nombres de los argumentos como hemos visto (argumento:= valor), puesto que mientras que *Password* es el primer argumento, *UserInterfaceOnly* es el quinto argumento.

Con este ejemplo, protegemos la Hoja1 de la edición por parte de un usuario, pero no desde una macro.

```
Dim MiHoja As Worksheet
Set MiHoja = Worksheets("Hoja1")
MiHoja.Protect Password:="alimoche", UserInterfaceOnly:=True
```

Con este otro ejemplo, desprotegemos la hoja protegida anteriormente.

```
Dim MiHoja As Worksheet
Set MiHoja = Worksheets("Hoja1")
MiHoja.UnProtect Password:="alimoche"
```

- ***Worksheet.Range (¡¡Ponte cómodo que comienza lo bueno!!)***

Devuelve un objeto de tipo *Range*, que hace referencia a un rango de celdas de la hoja. Su sintaxis es;

Worksheet.**Range(***Cell1, Cell2***)**

- ✓ ***Cell1*** (Obligatorio); Primera celda del rango. Objeto de tipo Range
- ✓ ***Cell2*** (Opcional); Última celda del rango. Objeto de tipo Range

Para referirnos a un rango en concreto de una hoja determinada podemos utilizar la nomenclatura; "CSI:CID" Donde CSI es la Celda Superior Izquierda del rango, y CID es la Celda Inferior Derecha del rango. No olvidar que se deben poner entrecomilladas y separadas por los dos puntos.

```
ActiveSheet.Range("A1:B5").Select
```

También podemos expresarlo de esta otra manera;

```
ActiveSheet.Range(Range("A1"), Range("B5")).Select
```

En el ejemplo de arriba; ***Cell1*** = Range("A1"), y ***Cell2*** = Range("B5")

O incluso de esta tercera manera (recordamos que el objeto Cells(i, j) era también un objeto de tipo Range.

```
ActiveSheet.Range(Cells(1,1), Range("B5")).Select
```

En este último ejemplo; ***Cell1*** = Cells(1,1), y ***Cell2*** = Range("B5")

Como devuelve un objeto de tipo *Range*, podemos utilizarlo bien para referenciar una variable-objeto de tipo *Range*, o bien para utilizar algunas de las propiedades del propio objeto *Range*.

Este primer ejemplo utiliza la propiedad *Range* para referenciar una variable-objeto de tipo *Range*.

```
Dim MiRango As Range
Set MiRango = Worksheets("Hoja2").Range("A1:C20")
```

Este segundo ejemplo utiliza directamente una propiedad del objeto

```
Worksheets("Hoja2").Range("A1:C20").Select
```

7.3 Objeto Range

El objeto ***Range*** hace referencia a un rango de celdas concreto de una hoja. Es el objeto más importante a la hora de trabajar con los datos de las Hojas. Como ya recordaremos, hemos trabajado con algunas de sus propiedades en varios momentos del curso. Es hora en entrar más en detalle y estudiar sus funcionalidades. Un objeto *Range* puede ser desde un rango de celdas a una sola celda.

- ***Range.Value***

Propiedad que hace referencia al valor de las celdas. Hemos utilizado varias veces esta propiedad.

Si no especificamos nada, Range actúa directamente sobre la hoja activa. Para darle un mismo valor a todas las celdas de un rango;

```
Range("A1:C20").Value = "Hola Mundo"
```

Un objeto de tipo Range, está compuesto como hemos dicho anteriormente por una celda o un conjunto de celdas. Si utilizamos un bucle For Each … Next, sobre un objeto de tipo range utilizando como contador otro objeto de tipo Range, este rango "contador" tomará el valor de cada celda del Rango en cada repetición del bucle. Para ver esto más claramente, fíjate en el ejemplo a continuación. Vamos a rellenar el rango A1:C20 con valores aleatorios entre 100 y 200;

```
Sub RellenaRango()
    Dim Valor As Double
    Dim MiRango As Range
    Dim Rg As Range
    Randomize
    Set MiRango = Range("A1:C20")
    For Each Rg In MiRango       'Por cada rango unidad (celda) en el rango "MiRango"
        Valor = (200 - 100) * Rnd + 100
        Rg.Value = Valor
    Next Rg
End Sub
```

- ***Range.Select***

Selecciona un rango concreto. Al seleccionar un rango, el objeto **Selection** (objeto tipo *Range*) pasa a contener la selección.

Ejemplo; vamos a seleccionar un rango de la Hoja2, lo vamos a seleccionar, y después vamos a referenciar una variable de tipo *Range* a la selección.

```
Dim MiRango As Range
Dim MiHoja As Worksheet
Worksheets("Hoja2").Select
Worksheets("Hoja2").Range("A1:B5").Select
Set MiRango = Selection
```

Para realizar una selección, la hoja tiene que ser la hoja activa. Prueba a eliminar la 3 línea de código ¿ves que pasa?

Es posible realizar una selección de varias celdas, para ello utilizaremos la siguiente sintaxis;

*Range***(**"celda1, celda2, celda3, …, celdan"**).Select**

celda1, celda2, celdan; son las referencias a las celdas. Se pueden obtener mediante la propiedad Range.Adress. Con la siguiente línea realizamos una selección de varias celdas.

```
Range("A2,B5:B8,C10:G20").Select
```

- ***Range.Adress***

Esta propiedad devuelve un valor de tipo String que representa la referencia del rango en el lenguaje de la macro. Su sintaxis es;

*Range***.Address(***RowAbsolute*, *ColumnAbsolute*, *ReferenceStyle*, *External*, *RelativeTo***)**

- ✓ ***RowAbsolute;*** Variable (true/false) que especifica si la fila se referencia de forma absoluta ($)
- ✓ ***ColumnAbsolute***; Variable (true/false) que especifica si la columna se referencia de forma absoluta ($)
- ✓ ***ReferenceStyle;*** Constante (**XlReferenceStyle**) que define el estilo de la referencia (valor predeterminado - xlA1)
- ✓ ***External;*** Variable que especifica si es una referencia externa (true) o local (false, predeterminado).
- ✓ ***RelativeTo***; Si RowAbsolute y ColumnAbsolute tienen el valor False y ReferenceStyle es xlR1C1, se debe incluir un punto inicial para la referencia relativa. Este argumento es un objeto Range que define el punto inicial de la referencia.

Con este ejemplo, obtenemos la referencia de la celda activa (con valores absolutos)

```
MsgBox ActiveCell.Adress
```

➢ ***Range.Offset***

Devuelve un objeto de tipo *Range* que representa un rango desplazado con respecto al rango especificado. Su sintaxis es;

Range**.Offset(**RowOffset, ColumnOffset**)**

- ✓ ***RowOffset***; variable de tipo long que especifica el número de filas desplazadas
- ✓ ***ColumnOffset***; variable de tipo long que especifica el número de columnas desplazadas

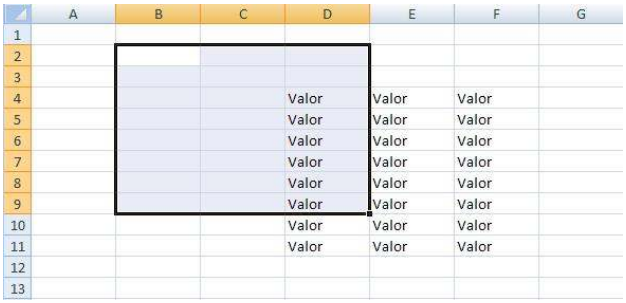
Para ver un ejemplo y entender mejor esta propiedad (muy importante), vamos a ver un ejemplo;

1. Abre un libro nuevo y selecciona un grupo de celdas en la Hoja1. Podemos colorear el interior de las celdas para visualizar mejor el ejemplo.

2. Crea una nueva macro y pega el siguiente código (¡Con la selección aún activa!)

```
Sub Ejemplo_OffSet()
    Dim MiRango As Range
    Set MiRango = Selection.Offset(2, 2)
    MiRango.Value = "Valor"
End Sub
```

En la siguiente imagen podemos ver lo que ocurre (el rango seleccionado inicialmente es B2:D9)



➢ ***Range.Item***

Representa un rango desplazado con respecto al rango especificado. Sin embargo nosotros vamos a utilizar esta propiedad para "movernos" por las celdas del rango*. La sintaxis que utilizaremos será;

*Range.**Item(**Index**)***

- ✓ ***Index***; representa la celda a la que se desea obtener acceso, por orden de izquierda a derecha y después hacia abajo.

En el siguiente ejemplo, utilizamos la propiedad Item para acceder a cada una de las celdas del rango.

```
Sub Ejemplo_Item()
    Dim MiRango As Range
    Dim i As Integer
    Set MiRango = Range("A1:C2")
    For i = 1 To 6
        MiRango.Item(i).Value = "Celda " & i
    Next i
End Sub
```

**Para una descripción más exacta de esta propiedad consultad la ayuda de VBA desde MS Excel.*

- ***Range.Copy***

Copia el rango especificado en otro rango que se especifique o en el portapapeles. Su sintaxis es;

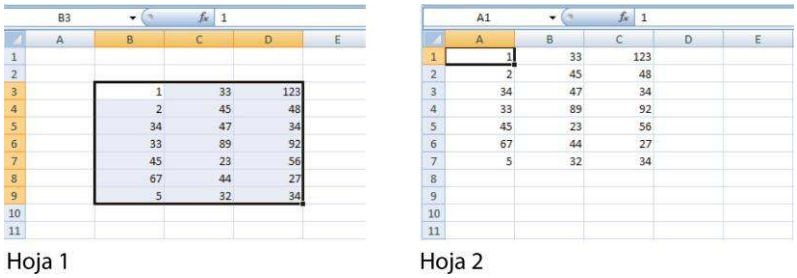
*Range.**Copy(**Destination**)***

- ✓ ***Destination*;** es el rango donde se copia el rango origen. Si no se especifica el rango se copia en el portapapeles.

En el siguiente ejemplo vamos a copiar un rango de la Hoja1 en otro rango de la Hoja2.

```
Sub CopiaRango()
    Dim RangoOrigen As Range
    Dim RangoDestino As Range
    Set RangoOrigen = Worksheets("Hoja1").Range("B3:D9")
    Set RangoDestino = Worksheets("Hoja2").Range("A1")
    RangoOrigen.Copy RangoDestino
End Sub
```

El resultado lo podemos ver en la siguiente imagen. Nos damos cuenta de que aunque el rango origen y el destino no sean iguales, el rango se copia desde la primera celda (superior izquierda) del rango destino.



Hoja 1 Hoja 2

Ejercicio 26

Abre el libro EjemploCombo1.xlsm que creamos anteriormente. Vimos que cuando introducíamos datos, siempre lo hacía en la primera fila. Vamos a modificar el código para que cada vez que introduzcamos un dato, lo hagamos en las celdas seleccionadas y la selección se mueva una fila hacia abajo.

1. Al cargar el formulario (evento Initalize del UserForm) seleccionar las celdas A2:C2

2. Al pulsar el botón para introducir datos;
   - ✓ Referenciar la selección con un objeto de tipo Range
   - ✓ Introducir los valores de las cajas de texto y combos en las celdas utilizando la propiedad Item(Index)
   - ✓ Hacer que la selección avance una fila hacia abajo (con OffSet)

Ejercicio 27.

Abre el libro Terremotos1990.xlsx. Vamos a crear una macro que seleccione todos los terremotos que se hayan producido un día 10. Copiaremos estos terremotos en una hoja nueva.

1. Para buscar en las diferentes celdas podemos utilizar;

```
Set Rg = Range("C2")          'Primera celda con valor del día del terremoto
Do until IsEmpty(Rg.value)
    [código ...]
    Set Rg = Rg.Offset(1,0)   'Nos movemos una celda hacía abajo
Loop
```

2. Para ir guardando los rangos podemos utilizar dos variables de tipo String, y haremos lo siguiente

```
NombreRango = Rg.Address(RowAbsolute:=False, ColumnAbsolute:=False)
RangoSeleccion = RangoSeleccion & NombreRango & ","
```

3. Al final tendremos nuestra variable RangoSeleccion, ¡¡pero con una coma al final!!. Para quitarle la coma podemos utilizar la función Left. (Ver Pdf adjunto, "Trabajo con cadenas de texto en VBA")

```
RangoSeleccion = Left(RangoSeleccion, Len(RangoSeleccion) - 1)
```

4. Para seleccionar las filas elegidas haremos

```
Range(RangoSeleccion).Select
Selection.EntireRow.Select
Set RangoDatos = Selection
```

5. Para copiar el rango en una hoja nueva; crearemos la hoja y utilizaremos el método Range.Copy(Destino)

- ***Range.Count***

Esta propiedad devuelve un entero largo (Long) con el número de celdas del rango especificado. Si lo combinamos con las propiedades Range.Columns y Range.Rows podemos saber el número de columnas y el de filas de un rango en concreto.

Range.**Count** ———— *Número de celdas del rango especificado.*

Range.**Columns.Count** — *Número de columnas del rango especificado*

Range.**Rows.Count** —— *Número de filas del rango especificado*

- ***Range.End***

Devuelve un objeto *Range* que representa la celda situada al final de la región que contiene el rango fuente. Equivale a presionar las teclas FIN+FLECHA ARRIBA, FIN+FLECHA ABAJO, FIN+FLECHA IZQUIERDA o FIN+FLECHA DERECHA. Su sintaxis es;

Range.**End(***Direction***)**

- ✓ ***Direction*;** es una constante de tipo *XlDirection* que especifica la dirección de búsqueda.

Constante	Valor	Dirección
xlDown	-4121	Hacia abajo.
xlToLeft	-4159	Hacia la izquierda.
xlToRight	-4161	Hacia la derecha.
xlUp	-4162	Hacia arriba.

Esta propiedad es muy útil para situarnos al principio o final de rangos. En el siguiente ejemplo vamos a mover la celda activa a la última celda hacia la derecha de la región actual (equivale a hacer FIN + Flecha Derecha).

```
Range("A1").Select
Selection.End(XlToRight).Select
```

Con el siguiente ejemplo vamos a seleccionar todas las celdas hacia la derecha. Para ello utilizaremos la propiedad Range(Cell1, Cell2).

```
Range("A1").Select
Range(Selection, Selection.End(XlToRight)).Select
```

En el ejemplo anterior ***Cell1*** = Selection (o sea la celda A1), y ***Cell2*** = Selection.End(XlToRight) (ultima celda a la derecha de la región actual.

Con este último ejemplo podemos seleccionar hacia la derecha y luego hacia abajo. El resultado es el mismo que utilizar la propiedad CurrentRegión si la selección se encuentra al comenzar en la celda superior izquierda de la región.

```
Range("A1").Select
Range(Selection, Selection.End(XlToRight)).Select
Range(Selection, Selection.End(XlDown)).Select
```

- ***Range.CurrentRegion***

Esta propiedad devuelve un objeto de tipo Range que representa la región actual del rango especificado. El siguiente ejemplo selecciona la región actual a la que pertenezca la celda A1.

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

Ejemplo; abre el libro Terremotos1990.xlsx. Vamos a ver las diferencias entre varios de los códigos mostrados anteriormente. Vamos a crear dos macros; Macro1 y Macro2. Les daremos el código a continuación para cada una de ellas.

```
Sub Macro2()
   Dim Micelda As Range
   Set Micelda = ActiveCell
   Range(Micelda, Micelda.End(xlToRight)).Select
   Range(Selection, Selection.End(xlDown)).Select
End Sub
```

```
Sub Macro1()
   Dim Micelda As Range
   Set Micelda = ActiveCell
   ActiveCell.CurrentRegion.Select
End Sub
```

Ahora prueba a seleccionar la celda A1, y ejecutar la primera Macro. Repite pero ahora ejecuta la segunda. ¿Ves alguna diferencia?

Ahora en vez de seleccionar la celda A1, seleccionemos la D6. Ejecutemos la primera macro, y luego la segunda ¿Observamos diferencias?

Ejercicio 28.

Abre de nuevo el libro del ejercicio 26. Vamos a modificar de nuevo el programa, para que al iniciarse, la selección se coloque en las siguientes 3 columnas vacías. Para ello utilizaremos las propiedades vistas anteriormente.

Ejercicio 29

Vamos a practicar un poco todo lo visto con un ejercicio.

Un compañero nuestro nos ha pasado una carpeta con libros de Excel correspondientes a datos de terremotos en los alrededores de Granada (carpeta Datos Terremotos Granada). Vamos a hacer una macro que combine todos los datos de todos los libros en uno solo.

1. Antes de nada crea un libro en blanco llamado "Terremotos Granada.xlsm" y guárdalo en la misma carpeta. Lo volvemos a abrir y creamos la macro (haciendo esto cogeremos la ruta de este libro).

   Para obtener la ruta podemos utilizar la propiedad de nuestro libro ("Terremotos Granada.xlsm")

   ```
   Dim MiLibro As Workbook
   Set MiLibro = ThisWorkbook
   Ruta = MiLibro.Path & "\"
   ```

2. Para obtener un listado de los archivos podemos utilizar la función Dir
   NombreArchivo = Dir(Ruta & "*.xlsx") ——— Devuelve el nombre del primer archivo
3. La rutina que tendrá que seguir la macro será;
   a. Abrir un libro y referenciar un rango con los datos
   b. Activar de nuevo nuestro libro y copiar los datos del rango referenciado en la siguiente fila vacía
   c. Cerrar el libro abierto, y abrir el siguiente
4. Para evitarnos copiar la cabecera más de una vez, podemos referenciar dos Rangos (RangoCabecera y RangoDatos) y con una variable booleana (true/false) podemos indicar si la cabecera ya se copió o no.
5. No olvidemos que Dir solo nos devolverá el nombre del archivo, y que para utilizar el método Workbooks.Open, es necesario el nombre completo (ruta + nombre)

Función Dir

Esta función devuelve una variable String con el nombre de un archivo (si existe), o grupo de archivos que coincida con un patrón. Su sintaxis es

Dir (*nombre_Ruta, atributos***)**

- ✓ **Nombre_Ruta**; representa el nombre de un archivo con su ruta completa
- ✓ **Atributos**; Constante o expresión numérica, cuya suma especifica los atributos de archivo. (opcional)

Si el archivo no existe, devuelve una cadena vacía. Ejemplo;

Dir("C:\Windows\win.ini") *– Devuelve "win.ini", si este archivo existe en la ruta especificada*

Dir("C:\Windows*.ini") *– Si hay más de un archivo *.ini en el directorio, devuelve el primer archivo encontrado. Al llamar nuevamente a la función Dir (esta vez sin argumentos) nos devolverá el siguiente archivo *.ini, y así sucesivamente.*

Ejemplo; generar una matriz de tipo texto con todos los archivos *.ini del directorio C:\Windows

```
Sub CreaListado()
    Dim Archivos() As String
    Dim Ruta As String, Nombre As String
    Dim i As Integer
    Ruta = "C:\Windows\*.ini"
    Nombre = Dir(Ruta)
    Do Until Nombre = ""
        i = i + 1
        ReDim Preserve Archivos(1 To i)
        Archivos(i) = Nombre
        Nombre = Dir
    Loop
End Sub
```

- ➢ ***Range.ColumnDifferences*** *y* ***Range.RowDifferences***

Estos dos métodos devuelven un objeto *Range* que representa todas las celdas cuyo contenido es diferente del de la celda de comparación de cada columna o fila respectivamente. Sus sintaxis son;

*Range***.ColumnDifferences(***Comparison***)**

*Range***.RowDifferences(***Comparison***)**

- ✓ ***Comparison;*** Una sola celda que se compara con el rango especificado.

Si especificamos un rango de varias columnas/filas, el valor de comparison cambiará para cada fila/columna. Es decir, si especificamos un rango con varias columnas (Columnas A, B, C, D y F), y el valor de comparison es (A1), para la columna A será A1, para la B será B1, para la C será C1 …

Para ilustrar mejor estos dos métodos, abre el libro EjemploDifferences.xlsm. Este libro contiene dos macros; DiferenciasColumnas y DiferenciasFilas. Ejecuta cada una de ellas y observa las diferencias.

```
Sub DiferenciasColumnas()
   Dim MiRango As Range
   Dim Rg As Range
   Dim valor As Variant
   Set MiRango = Range("A1").CurrentRegion
   MiRango.ColumnDifferences(Range("A1")).Select
End Sub
```

	A	B	C	D	E	F	G	H	I	J	K
1		10	20	30	40	50	60	70	80	90	100
2	10	40	50	20	10	30	10	60	20	100	70
3	20	20	20	60	90	10	30	10	20	40	20
4	30	10	30	30	70	50	40	20	80	100	60
5	40	50	30	40	10	40	20	100	10	50	100
6	50	30	50	10	40	80	30	30	50	30	100
7	60	10	60	60	40	60	10	70	50	100	20
8	70	50	30	70	70	70	40	60	100	60	50
9	80	80	40	80	50	80	20	50	60	90	60
10	90	50	70	90	40	40	30	20	60	90	90
11	100	70	80	100	100	20	90	30	100	10	10

```
Sub DiferenciasFilas()
   Dim MiRango As Range
   Dim Rg As Range
   Dim valor As Variant
   Set MiRango = Range("A1").CurrentRegion
   MiRango.RowDifferences(Range("A1")).Select
End Sub
```

	A	B	C	D	E	F	G	H	I	J	K
1		10	20	30	40	50	60	70	80	90	100
2	10	40	50	20	10	30	10	60	20	100	70
3	20	20	20	60	90	10	30	10	20	40	20
4	30	10	30	30	70	50	40	20	80	100	60
5	40	50	30	40	10	40	20	100	10	50	100
6	50	30	50	10	40	80	30	30	50	30	100
7	60	10	60	60	40	60	10	70	50	100	20
8	70	50	30	70	70	70	40	60	100	60	50
9	80	80	40	80	50	80	20	50	60	90	60
10	90	50	70	90	40	40	30	20	60	90	90
11	100	70	80	100	100	20	90	30	100	10	10

Esté método también puede ser muy útil para seleccionar datos de una columna dada. Para ello el rango deberá de ser solo la columna donde queremos buscar las diferencias. El siguiente ejercicio nos muestra un ejemplo de ello.

Ejercicio 30.

Abre el libro "Terremotos Granada.xlsm" creado en el ejercicio anterior. Vemos que la columna G contiene el tipo de terremoto. Los designados con LP corresponden a explosiones, y queremos eliminarlos de nuestra tabla. Para ello haremos una macro que nos seleccione todos los terremotos que sean "L" y los copie en una segunda hoja ("Terremotos Buenos"). Utilizar ColumnDifferences.

➢ ***Range.AutoFill***

Rellena automáticamente las celdas del rango especificado. Su sintaxis es;

Range.**AutoFill(**Destination, Type**)**

- ✓ **Destination;** Rango de destino donde se rellenarán las celdas
- ✓ **Type;** tipo de relleno. Constante de tipo **XlAutoFillType.** (***XlFillCopy*** –copia valores; ***XlFillFormat*** – copia formatos; ***XlFillDefault*** – copia formulas [valor por defecto])

Constante	Valor	Dirección
xlDown	-4121	Hacia abajo.
xlToLeft	-4159	Hacia la izquierda.
xlToRight	-4161	Hacia la derecha.
xlUp	-4162	Hacia arriba.

- ✓

El siguiente ejemplo rellena las celdas de la columna B con la fórmula de la celda B1

```
Dim Destino As Range
Set Destino = Range("B1:B100")
Range("B1").AutoFill Destino, XlFillDefault
```

Ejercicio 31.

Abre el libro de Excel "EjemploFormatos.xlsx". Hacer una macro que copie los formatos de la primera fila con datos a las demás.

➢ ***Range.Sort***

Ordena un rango de valores. Su sintaxis es;

Range.**Sort(***Key1, Order1, Key2, Type, Order2, Key3, Order3, Header, OrderCustom, MatchCase, Orientation, SortMethod, DataOption1, DataOption2, DataOption3***)**

- ✓ **Key1;** Especifica el primer campo de ordenación, ya sea como nombre de rango (cadena) u objeto Range; determina los valores que se deben ordenar.
- ✓ **Order1**; Determina el criterio de ordenación para los valores especificados en Key1. Constante *XlSortOrder.*
- ✓ **Key2;** Segundo campo de ordenación; no se puede utilizar al ordenar tablas dinámicas.
- ✓ **Type**; Especifica qué elementos se deben ordenar.
- ✓ **Order2;** Determina el criterio de ordenación para los valores especificados en Key2. Constante *XlSortOrder.*
- ✓ **Key3;** Tercer campo de ordenación; no se puede utilizar al ordenar tablas dinámicas.
- ✓ **Order3**; Determina el criterio de ordenación para los valores especificados en Key3. Constante *XlSortOrder.*
- ✓ **Header;** Especifica si la primera fila contiene información de encabezado. xlNo es el valor predeterminado.
- ✓ **Ordercustom;** Especifica un entero en base uno que constituye la posición en la lista de criterios de ordenación personalizados.
- ✓ **MatchCase;** Se debe establecer en True para realizar una ordenación que distinga entre mayúsculas y minúsculas, o en False para no tener en cuenta las mayúsculas y minúsculas al llevar a cabo la ordenación; no se puede utilizar con tablas dinámicas.
- ✓ **Orientation;** Especifica si la ordenación debe realizarse por filas o por columnas. Constante *XlSortOrientation.*

- ✓ **SortMethod;** Especifica el método de ordenación.
- ✓ **DataOption1**; Especifica cómo se debe ordenar el texto del rango especificado en el parámetro Key1; no se aplica a la ordenación de tablas dinámicas.
- ✓ **DataOption1;** Especifica cómo se debe ordenar el texto del rango especificado en el parámetro Key2; no se aplica a la ordenación de tablas dinámicas.
- ✓ **DataOption1;** Especifica cómo se debe ordenar el texto del rango especificado en el parámetro Key3; no se aplica a la ordenación de tablas dinámicas.

Constante ***XlSortOrder*** (determina si la orientación se hace en orden ascendente o descendente)

Constante	Valor	Descripción
xlAscending	1	Ordena el campo especificado en sentido ascendente. Éste es el valor predeterminado.
xlDescending	2	Ordena el campo especificado en sentido descendente.

Constante ***XlSortOrientation*** (determina si ordenación se realiza por filas o por columnas)

Constante	Valor	Descripción
xlSortColumns	1	Ordena por columnas.
xlSortRows	2	Ordena por filas. Éste es el valor predeterminado.

Vamos a ver algunos ejemplos fáciles;

Ordenar un rango de valores en orden ascendente según el valor de la primera celda;

Mirango.Sort Key1:=Range("A1"), Order1:=xlAscending

Ordenar un rango de valores en orden descendente y que tenga cabecera;

Mirango.Sort Key1:=Range("A1"), Order1:=xlDescending, Header:=xlYes

Ordenar un rango de valores con dos claves (*XlAscending* es el valor de Order por defecto);

Mirango.Sort Key1:=Range("A1"), Key2:=Range("B1"), Header:= True

Ejercicio 32

Abre el libro de Excel "ListaEspecies para ordenar.xlsx". Este libro representa un inventariado de árboles de un jardín botánico en la que a cada especie se l ha asignano un número identificador y se le ha medido la altura y el diámetro. Haz una macro que ordene el inventario según dos claves; clave1 será el nombre de la especie, y la clave2 será o bien la altura o bien el diámetro (la macro nos preguntará que clave queremos aplicar como clave2)

➢ ***Dando formato a las celdas***

Para dar formato a las celdas de un rango utilizaremos sus propiedades;

Range.**Interior** –Controla el interior de una celda (color de relleno, motivo, etc.)

Range.**Borders** – Controla el borde de las celdas (bordes, grosor, color, etc.)

Range.**Font** – Controla la fuente de las celdas (tipo de fuente, negrita, cursiva, tamaño, color, etc.)

Estas propiedades devuelven a su vez un objeto; objeto tipo Interior, colección de objetos Borders, y objeto tipo Font. Para una explicación detallada de estos objetos consultad la ayuda de Excel. A continuación se expondrán unos ejemplos tipos para controlar el formato.

1. Cambiar el color de fondo a un rango

```
Dim MiRango As Range
Set MiRango = Range("A1:A10")
With MiRango.Interior
    .Color = RGB(204,192,218)
End With
```

2. Cambiar el tipo de fuente

```
Dim MiRango As Range
Set MiRango = Range("A1:A10")
With MiRango.Font
   .Bold = True
   .Italic = True
   .Size = 14
   .Name = "Times New Roman"
   .Color = vbBlue
End With
```

3. Cambiar los bordes

```
Dim MiRango As Range
Set MiRango = Range("A1:A10")
With MiRango.Borders
   .Color = RGB(108, 82, 136)
   .Weight = xlThick
End With
```

Cambia todos los bordes

```
Dim MiRango As Range, Rg As Range
Set MiRango = Range("A1:A10")
For Each Rg In MiRango
      With Rg.Borders.Item (xlEdgeBottom)
         .Color = RGB(108, 82, 136)
         .Weight = xlThick
      End With
Next Rg
```

Cambia el borde inferior de cada celda