

MACROS EN MS EXCELL

¿Que es una macro ?.

Una macro son un conjunto de instrucciones que sirven para automatizar procesos. Refiriéndonos a excel, supongamos que realizamos frecuentemente la acción de seleccionar un rango para aplicarle negrita, cambio de fuente y centrado. En lugar de hacer estas acciones manualmente, se puede elaborar una macro e invocarla para que ejecute los tres procesos automáticamente.

Objetos, propiedades y métodos.

A la hora de trabajar con macros en excel, deben tenerse claros ciertos conceptos de lo que se llama programación orientada a objetos (OOP). No nos extenderemos demasiado sobre la OOP, pero si definiremos a continuación los conceptos de **Objeto**, **Propiedades** y **Métodos**.

Objeto.

Cuando en el mundo real nos referimos a objeto significa que hablamos de algo más o menos abstracto que puede ser cualquier cosa. Si decidimos concretar un poco más podemos referirnos a objetos coche, objetos silla, objetos casa, etc. En OOP, la generalización (o definición) de un objeto se llama **Clase**, así la clase coche sería como la representante de todos los coches del mundo, mientras que un objeto coche sería un coche en concreto. De momento, no definiremos ni estudiaremos las clases sino que nos concentraremos en los objetos, tenga en cuenta pero que **cualquier objeto está definido por una clase**.

Cuando decimos que la clase coche representa a todos los coches del mundo significa que define como es un coche, cualquier coche. Dicho de otra forma y para aproximarnos a la definición informática, la clase coche define algo que tiene cuatro ruedas, un motor, un chasis,... entonces, cualquier objeto real de cuatro ruedas, un motor, un chasis,... es un objeto de la clase coche.

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.

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.

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 casilla o rango de casillas).

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,...

Conjuntos.

Una conjunto es una colección de objetos del mismo tipo, para los que conozcan algún lenguaje de programación es un array de objetos. Por ejemplo, dentro de un libro de trabajo puede existir más de una hoja (**WorkSheet**), todas las hojas de un libro de trabajo forman un conjunto, **el conjunto Worksheets**. Cada elemento individual de un conjunto se referencia por un índice, de esta forma, la primera, segunda y tercera hoja de un libro de trabajo, se referenciarán por **Worksheets(1)**, **Worksheets(2)** y **Worksheets(3)**.

Objetos de Objetos.

Es muy habitual que una propiedad de un objeto sea otro objeto. Siguiendo con el coche, una de las propiedades del coche es el motor, y el motor es un objeto con propiedades como cubricaje, 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). Tenga 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**.

Programación Orientada a Objetos o Programación Basada en Objetos.

Hay una sutil diferencia entre las definiciones del título. Programación orientada a Objetos, significa que el programador trabaja con objetos fabricados por él mismo, es decir, el programador es quien implementa las clases para luego crear objetos a partir de ellas. Lo que haremos nosotros, por el momento, será utilizar objetos ya definidos por la aplicación Excel (**Worksheets**, **Range**,...) sin implementar ninguno de nuevo, por lo que en nuestro caso es más correcto hablar de programación basada en objetos. Observe que esta es una de las grandes ventajas de la OOP, utilizar objetos definidos por alguien sin tener que conocer nada sobre su implementación, sólo debemos conocer sus propiedades y métodos y utilizarlos de forma correcta.

Bueno, después de esta extensa pero necesaria introducción pasemos ya a hacer alguna cosa en Excel. No es necesario que se aprenda lo anterior al pie de la letra y tampoco es necesario que lo comprenda al cien por cien, sólo téngalo presente para las definiciones que vienen a continuación y verá como va asimilando los conceptos de Objeto, propiedades, métodos, etc.

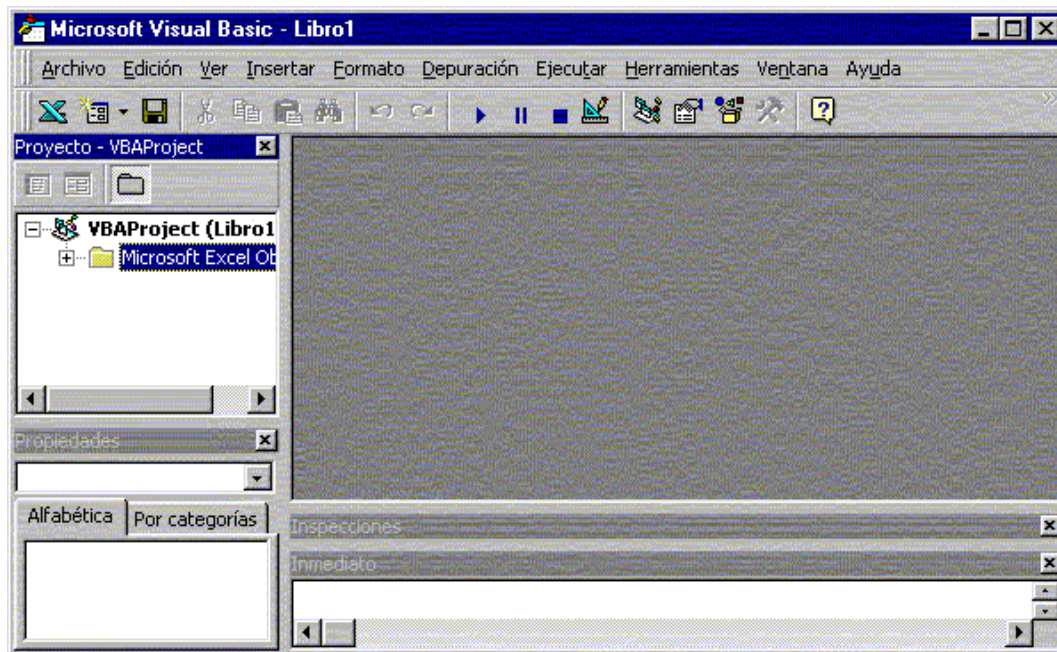
Editor de Visual Basic.

El editor de visual básico es la aplicación que utilizaremos para construir las macros que interactuarán junto con los libros de trabajo. A continuación prepararemos un archivo en el que escribiremos las primeras instrucciones en Visual basic.

Preparar un archivo nuevo.

Para entrar en el editor de Visual Basic, ejecute los pasos siguientes.

1. Active opción **Herramientas/ Macro/ Editor de Visual Básic** Se abrirá la ventana siguiente.



Maximize la ventana para trabajar más cómodamente y procure tener activadas la ventana **Explorador de proyectos** y la ventana **Propiedades** (**Ver/ Explorador de proyectos** y **Ver/ Ventana propiedades**).

Insertar un nuevo módulo.

Un módulo sirve para agrupar procedimientos y funciones. El procedimiento y la función son entidades de programación que sirven para agrupar instrucciones de código que realizan una acción concreta.

Para insertar un módulo active opción del menú **Insertar/ Módulo** Se activará una nueva ventana, si aparece demasiado pequeña, maximícela.

Insertar un procedimiento.

Ya hemos dicho que un procedimiento es un bloque de instrucciones de código que sirven para llevar a cabo alguna tarea específica. Un procedimiento empieza siempre con la instrucción

Sub *Nombre_Procedimiento*

Y termina con la instrucción

End Sub

A continuación crearemos un procedimiento para poner el texto "Hola" en la casilla A1.

Ejemplo 1

Sub *Primero*

Range("A1").**Value** = "Hola"

End Sub

Observe el código.

Range("A1").**Value**="Hola"

En esta línea estamos indicando que trabajamos con un objeto **Range**. Para indicarle que nos referimos a la casilla A1, encerramos entre paréntesis esta referencia (más adelante verá otra forma de referirnos a las casillas). De este objeto, indicamos que queremos establecer un nuevo valor para la propiedad **Value**, observe que para separar el objeto de su propiedad utilizamos la notación punto.

Recuerde que el conjunto **Range** es un objeto que pende del objeto **Worksheets**, así por ejemplo el siguiente código haría lo mismo que el anterior.

Worksheets(1).**Range**("A1").**Value** = "Hola"

Bueno, de hecho no hace lo mismo, en la primera opción, el texto "Hola" se pone dentro de la casilla A1 de la hoja activa, mientras que en el segundo es en la casilla A1 de primera hoja (del conjunto de hojas). La segunda notación es más larga, pero también más recomendable ya que se especifican todos los objetos. En muchas ocasiones se pueden omitir algunos objetos precedentes, no le aconsejamos hacerlo, sus programas perderán claridad y concisión.

Si desea hacer referencia a la hoja activa puede utilizar **ActiveSheet**, así, el primer ejemplo lo dejaremos de la manera siguiente.

Sub *Primero*

ActiveSheet.**Range**("A1").**Value** = "Hola"

End Sub

Si desea poner "Hola" (o cualquier valor) en la casilla activa, puede utilizar la propiedad (objeto) **ActiveCell** de **Worksheets**. Así para poner "Hola" en la casilla activa de la hoja activa sería,

Sub *Primero*

```
ActiveSheet.ActiveCell.Value = "Hola"
```

End Sub

Para terminar con este primer ejemplo. **Worksheets** están dentro del Objeto **WorkBooks** (libros de trabajo) y **WorkBooks** están dentro de **Application**. **Application** es el objeto superior, es el que representa la aplicación Excel. Así, el primer ejemplo, siguiendo toda la jerarquía de objetos quedaría de la forma siguiente.

Sub *Primero*


```
Application.WorkBooks(1).Worksheets(1).Range("A1").Value = "Hola"
```

End Sub

Insistiendo con la nomenclatura, **Application** casi nunca es necesario especificarlo, piense que todos los objetos penden de este, **WorkBooks** será necesario implementarlo si en las macros se trabaja con diferentes libros de trabajo (diferentes archivos), a partir de **Worksheets**, es aconsejable incluirlo en el código, sobre todo si se quiere trabajar con diferentes hojas, verá, sin embargo, que en muchas ocasiones no se aplica.

Ejecutar un procedimiento o función.

Pruebe ejecutar el primer procedimiento de ejemplo.

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

Para ejecutar el procedimiento desde la hoja de cálculo.

Debe estar en una hoja, no en el editor de Visual Basic

1. Active opción de la barra de menús **Herramientas/ Macro/ Macros**. Se despliega una ventana que muestra una lista donde estás todas las macros incluidas en el libro de trabajo.
2. Seleccione la macro de la lista y pulse sobre el botón **Ejecutar**.

Ejemplo 2

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

Sub Segundo

```
ActiveSheet.Range("A1").Value = "Hola"  
ActiveSheet.Range("A1").Font.Bold = True  
ActiveSheet.Range("A1").Font.Color = RGB(255,0,0)
```

End Sub

True.

True, que traducido es verdadero, simplemente indica que la propiedad **Bold** está activada. Si se deseara desactivar, bastaría con igualarla al valor **False**.

La función RGB.

Observe que para establecer el color de la propiedad se utiliza la función **RGB**(Red, Green, Blue), los tres argumentos para esta función son valores del 0 a 255 que corresponden a la intensidad de los colores Rojo, Verde y Azul respectivamente.

Referenciar un rango de celdas.

Sólo tiene que cambiar a la forma **Casilla_Inicial:Casilla_Final**. Por ejemplo aplicar el último ejemplo al rango de casillas que va de la A1 a la A8, ponga.

Sub Segundo

```
ActiveSheet.Range("A1:A8").Value = "Hola"  
ActiveSheet.Range("A1:A8").Font.Bold = True  
ActiveSheet.Range("A1:A8").Font.Color = RGB(255,0,0)
```

End Sub

Variables.

A continuación vamos a repetir el programa Ejemplo1, pero en lugar de poner "Hola" en la casilla A1 de la hoja activa, dejaremos que el usuario entre un texto desde teclado y a continuación guardaremos ese valor en esa casilla. Observe que el valor que entre del usuario debe guardarse en algún lugar para poder ponerlo después en la casilla A1; pues bien, ese valor se guardará en una variable. Una variable es simplemente un trozo de memoria que la función o procedimiento se reserva para guardar datos, la forma general de declarar una variable es

DIM variable **AS** *tipo* .

Siendo variable el nombre que se asigna a la misma y **Tipo** el tipo de datos que se guardarán (números, texto, fecha, booleanos,...). En nuestro ejemplo, declaramos la variable de tipo **String** (tipo texto), y lo haremos de la forma siguiente.

Dim Texto **As** *String*

Con esto estamos indicando que se reserve un trozo de memoria (el que sea) , que se llama Texto y que el tipo de datos que se guardarán ahí serán caracteres.

La Función *InputBox*.

Esta función muestra una ventana para que el usuario pueda teclear datos. Cuando se pulsa sobre **Aceptar**, los datos entrados pasan a la variable a la que se ha igualado la función. Vea la línea siguiente.

Texto = **InputBox**("Introduzca el texto", "Entrada de datos").

Si en la ventana que muestra **InputBox** pulsa sobre el botón **Aceptar**, los datos tecleados se guardarán en la variable **Texto**.

Sintaxis de *InputBox*.

InputBox(*Mensaje, Título, Valor por defecto, Posición horizontal, Posición Vertical, Archivo ayuda, Número de contexto para la ayuda*).

Mensaje : Es el mensaje que se muestra en la ventana. Si desea poner más de una línea ponga Chr(13) para cada nueva línea, vea el ejemplo siguiente.

Título : Es el título para la ventana **InputBox**. Es un parámetro opcional.

Valor por defecto: Es el valor que mostrará por defecto el cuadro donde el usuario entra el valor. Parámetro opcional.

Posición Horizontal: La posición X de la pantalla donde se mostrará el cuadro, concretamente es la posición para la parte izquierda. Si se omite el cuadro se presenta horizontalmente centrado a la pantalla.

Posición Vertical: La posición Y de la pantalla donde se mostrará el cuadro, concretamente es la posición para la parte superior. Si se omite el cuadro se presenta verticalmente centrado a la pantalla.

Archivo Ayuda: Es el archivo que contiene la ayuda para el cuadro. Parámetro opcional.

Número de contexto para la ayuda: Número asignado que corresponde al identificador del archivo de ayuda, sirve para localizar el texto que se debe mostrar. Si se especifica este parámetro, debe especificarse obligatoriamente el parámetro **Archivo Ayuda**.

Ejemplo 3

Sub *Entrar_Valor*

Dim Texto **As String**

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

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

End Sub

Este ejemplo también se puede hacer sin variables.

Sub *Entrar_Valor*

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

End Sub

Ejemplo 4

Repetiremos el ejemplo 3, pero en lugar de entrar los valores sobre la casilla A1, haremos que el usuario pueda elegir en que casilla quiere entrar los datos, es decir, se le preguntará al usuario mediante un segundo Inputbox sobre que casilla quiere entrar el valor del primer Inputbox. Serán necesaria dos variables, una para guardar la casilla que escoja el usuario y otra para guardar el valor.

Option Explicit

Sub *Entrar_Valor*

Dim Casilla **As String**

Dim Texto **As String**

Casilla = InputBox("En que casilla quiere entrar el valor", "Entrar Casilla")
Texto = InputBox("Introducir un texto " & Chr(13) & "Para la casilla " & Casilla , "Entrada de
datos")

ActiveSheet.Range(Casilla).Value = Texto

End Sub

La sentencia *Option Explicit*.

En visual basic no es necesario declarar las variables, por ejemplo, en el programa anterior se hubiera podido prescindir de las líneas

```
Dim Casilla As String  
Dim Texto As String
```

A pesar de ello, le recomendamos que siempre declare las variables que va a utilizar, de esta forma sabrá cuales utiliza el procedimiento y que tipo de datos guarda cada una, piense que a medida que vaya aprendiendo, creará procedimientos cada vez más complicados y que requerirán el uso de más variables, si no declara las variables al principio del procedimiento ocurrirán dos cosas. Primero, las variables no declaradas son asumidas como tipo **Variant** (este es un tipo de datos que puede almacenar cualquier valor, número, fechas, texto, etc. pero tenga en cuenta que ocupa 20 Bytes y para guardar una referencia a una casilla, la edad de alguien, etc. no son necesarios tantos bytes); segundo, reducirá considerablemente la legibilidad de sus procedimientos ya que las variables las irá colocando a medida que las necesite, esto, a la larga complicará la corrección o modificación del procedimiento.

Bueno, pues toda la explicación anterior es para que declare todas las variables que va a utilizar. La sentencia **Option Explicit** al principio del módulo fuerza a que se declaren todas las variables. Si al ejecutar el programa, se encuentra alguna variable sin declarar se producirá un error y no se podrá ejecutar el programa hasta que se declare.

Si todavía no se ha convencido sobre la conveniencia de declarar las variables y utilizar **Option Explicit**, pruebe el procedimiento siguiente, cópielo tal cual (Texto y Testo están puestos adrede simulando que nos hemos equivocado al teclear).

Sub Entrar_Valor

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

End Sub

Observe que el programa no hace lo que se pretendía que hiciera. Efectivamente, Texto y Testo son dos variables diferentes, como no se ha declarado ninguna ni se ha utilizado **Option Explicit** Visual Basic no da ningún tipo de error y ejecuta el programa. Pruebe el siguiente módulo e intente ejecutarlo.

Option Explicit

Sub Entrar_Valor

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

End Sub

Observe que el programa no se ejecuta, al poner **Option Explicit**, forzamos a que se declaren todas las variables. Visual Basic detecta que la variable *Testo* no ha sido declarada y así lo indica mostrando Error, entonces es cuando es más fácil darnos cuenta del error que hemos cometido al teclear y cambiamos Testo por Texto. Ahora imagine que el error se produce en un programa de cientos de líneas que necesita otras tantas variables.

Ejecute el procedimiento y ponga respectivamente los valores 25 y 25. Observe que todo ha ido correctamente y en la casilla A1 de la hoja activa aparece un 50.

Ahora, vuelva a ejecutar el programa y cuando se le pide el primer valor teclee "Hola". Observe que el programa se detiene indicando un error en el tipo de datos. Efectivamente, observe que la función `InputBox` devuelve siempre datos tipo *String*, en el primer ejemplo no ha habido ningún problema, al entrar caracteres numéricos¹, estos pueden asignarse a variables tipo *Integer* porque Visual Basic hace automáticamente la conversión, pero al entrar texto e intentarlo asignar a una variable *Integer* Visual Basic muestra un error indicando que la variable no es adecuada para los datos que se desean guardar.

Para solucionar estos problemas se deben utilizar funciones de conversión de tipo. Estas funciones, como su nombre indica, convierten datos de un tipo a otro, de *String* a *Integer*, de *Integer* a *String*, de *Date* a *String*, ... Así el procedimiento anterior quedaría.

Option Explicit

Sub *Sumar*()

Dim Numero1 **As Integer**

Dim Numero2 **As Integer**

Numero1 = Val(InputBox("Entrar el primer valor", "Entrada de datos"))

Numero2 = Val(InputBox("Entrar el primer valor", "Entrada de datos"))

ActiveSheet.Range("A1").Value = Numero1 + Numero2

End Sub

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

¹ Tenga en cuenta que para los ordenadores no es lo mismo el número 1 que el carácter "1". En los lenguajes de programación actuales la conversión de carácter "1" a número 1 se hace automáticamente en muchos casos, esto es bueno y es malo. Es bueno porque nos ahorra tener que hacer las conversiones y es malo porque es más difícil controlar ciertos casos. Siga con los ejemplos y entenderá de lo que estamos hablando. Sólo para su información, el ordenador guarda el número 1 de la siguiente manera 00000001, mientras que el carácter "1" se guarda como 00110000 (el número 48 del código ASCII).

Variables de Objetos.

Una variable objeto sirve para hacer referencia a un objeto, esto significa que podremos acceder a las propiedades de un objeto e invocar a sus métodos a través de la variable en lugar de hacerlo directamente a través del objeto. Posiblemente no se utilice demasiado esta clase de variables (esta claro que esto dependerá de las preferencias del programador), pero hay casos en los que no hay más remedio que utilizarlas, por ejemplo en estructuras **For Each ... Next** como veremos, o cuando sea necesario construir funciones que devuelvan rangos, referencias a hojas, etc.

Para declarar una variable objeto se utiliza también la palabra **Dim** de la forma siguiente,

```
Dim Var_Objeto As Objeto
```

Por Ejemplo

```
Dim R As Range  
Dim Hoja As WorkSheet
```

Para asignar un objeto a una variable debe utilizar la instrucción **Set**.

```
Set Variable_Objeto = Objeto
```

Por Ejemplo

```
Set R= ActiveSheet.Range("A1:B10")  
Set Hoja = ActiveSheet  
Set Hoja = WorkSheets(1)
```

Veamos a continuación un ejemplo de cómo utilizar este tipo de variables,

Ejemplo 6.

Algo muy simple, llenar el rango de A1 a B10 con la palabra "Hola" y después poner negrita, observe como se asigna una variable objeto al objeto y luego como se trabaja con esa variable de la misma forma que trabajaría directamente sobre el objeto.

Sub obj()

```
Dim R As Range
```

```
Set R = ActiveSheet.Range("A10:B15")  
R.Value = "Hola"  
R.Font.Bold = True
```

End Sub

El valor Nothing.

Algunas veces puede que sea necesario desasignar una variable del objeto al cual hace referencia, en este caso debe igualar la variable al valor **Nothing** de la forma siguiente.

```
Set Variable_Objeto = Nothing
```

Habitualmente se utiliza **Nothing** en una estructura condicional para comprobar si la variable objeto está asignada. Observe que si se utiliza una variable objeto a la cual todavía no se le ha hecho ninguna asignación el programa dará error y detendrá su ejecución. Es buena práctica hacer este tipo de comprobaciones antes de trabajar con variables objeto. Veremos un ejemplo de esto en el tema siguiente.

Estructuras condicionales.

Ahora que ya ha experimentado con unos cuantos objetos y propiedades, nos detendremos a estudiar las estructuras condicionales. Las estructuras condicionales son instrucciones de programación que permiten controlar la ejecución de un fragmento de código en función de si se cumple o no una condición. Estudiaremos en primer lugar la instrucción **if** Condición **then..End if** (**Si** Condición **Entonces..Fin Si**)

La estructura condicional que se construye con la instrucción **Si** Condición **Entonces... Fin Si** tiene la forma siguiente.

```
Si Condición Entonces
    Sentencia1
    Sentencia2
    .
    .
    SentenciaN
Fin Si
```

Cuando el programa llega a la instrucción **Si** Condición **Entonces**, se evalúa la condición, si esta se cumple (es cierta), se ejecutan todas las sentencias que están encerradas en el bloque, si no se cumple la condición, se saltan estas sentencias. Esta estructura en Visual Basic tiene la sintaxis siguiente,

```
If Condición Then
    Sentencia1
    Sentencia2
    .
    .
    SentenciaN
End If
```

Ejemplo 6.

Entrar una cantidad que representa el precio de algo por el teclado con la instrucción **InputBox** y guardarlo en la celda A1 de la hoja activa. Si el valor entrado desde el teclado (y guardado en A1) es superior a 1000, pedir descuento con otro **InputBox** y guardarlo en la casilla A2 de la hoja activa. Calcular en A3, el precio de A1 menos el descuento de A2.

Sub Condicional()

```
ActiveSheet.Range("A1").Value = 0           ' Poner las casillas donde se guardan los valores 0
ActiveSheet.Range("A2").Value = 0
ActiveSheet.Range("A3").Value = 0

ActiveSheet.Range("A1").Value = Val(InputBox("Entrar el precio", "Entrar"))

' Si el valor de la casilla A1 es mayor que 1000, entonces, pedir descuento
If ActiveSheet.Range("A1").Value > 1000 Then
    ActiveSheet.Range("A2").Value = Val(InputBox("Entrar Descuento", "Entrar"))
End If

ActiveSheet.Range("A3").Value = ActiveSheet.Range("A1").Value - _
ActiveSheet.Range("A2").Value
```

End Sub

Ejemplo 7.

El mismo que el anterior pero utilizando variables.

Option Explicit

Sub Condicional()

```
Dim Precio As Integer  
Dim Descuento As Integer
```

```
Precio = 0  
Descuento = 0
```

```
Precio = Val(InputBox("Entrar el precio", "Entrar"))
```

```
' Si el valor de la variable precio es mayor que 1000, entonces, pedir descuento
```

```
If Precio > 1000 Then  
    Descuento = Val(InputBox("Entrar Descuento", "Entrar"))
```

```
End If
```

```
ActiveSheet.Range("A1").Value = Precio  
ActiveSheet.Range("A2").Value = Descuento  
ActiveSheet.Range("A3").Value = Precio - Descuento
```

End Sub

Viendo los dos programas anteriores puede que le surja la duda de si emplear variables o directamente valores almacenados en las celdas. La solución es fácil, lo que le parezca más conveniente en cada caso concreto que desee solucionar. Las variables, aunque muchas veces "innecesarias", quizás dejan los programas más legibles y claros. Y la legibilidad de un programa es lo más valioso del mundo para un programador (profesionalmente hablando), sobre todo si se da el caso (inevitable el 99,999...% de las ocasiones) que se tenga que modificar un programa para dotarle de más funcionalidades, facilitar su manejo, etc. En la mayoría de ejemplos que encontrará en este manual verá que se utilizan variables preferentemente. Aunque muchas veces su función sea simplemente recoger datos de las celdas para operarlas y dejarlas en otras celdas y, consecuentemente, aumente el número de operaciones, creemos que con ello se gana en legibilidad y flexibilidad.

Ejemplo 8.

Macro que compara los valores de las casillas A1 y A2 de la hoja activa. Si son iguales pone el color de la fuente de ambas en azul.

Sub Condicional2()

```
If ActiveSheet.Range("A1").Value = ActiveSheet.Range("A2").Value Then
```

```
    ActiveSheet.Range("A1").Font.Color = RGB(0, 0, 255)  
    ActiveSheet.Range("A2").Font.Color = RGB(0, 0, 255)
```

```
End If
```

End Sub

- **Estructura If..Else**

Esta estructura se utiliza cuando se requiere una respuesta alternativa a una condición. Su estructura es la siguiente.

Si Condición **Entonces**

Sentencia1

Sentencia2

.

.

SentenciaN

Sino

Sentencia1

Sentencia2

.

.

SentenciaN

Fin Si

Observe que, si se cumple la condición, se ejecuta el bloque de sentencias delimitado por **Si** Condición **Entonces** y Si no se cumple la condición se ejecuta el bloque delimitado por **Sino** y **Fin Si**. En Visual Basic la instrucción **Si** Condición **Entonces** ... **Sino** ... **Fin Si** se expresa con las instrucciones siguientes.

If Condición **Then**

Sentencia1

Sentencia2

.

.

SentenciaN

Else

Sentencia1

Sentencia2

.

.

SentenciaN

End If

Ejemplo 9.

Entrar una cantidad que representa el precio de algo por el teclado con la instrucción `InputBox` y guardarlo en la celda A1 de la hoja activa. Si el valor entrado desde el teclado (y guardado en A1) es superior a 1000, se aplica un descuento del 10% sino se aplica un descuento del 5%, el descuento se guarda en la casilla A2 de la hoja activa. Colocar en A3, el total descuento y en A4 el total menos el descuento.

Sub Condicional_Else()

Dim Precio **As** Single

Dim Descuento **As** Single

Precio = 0

Precio = Val(InputBox("Entrar el precio", "Entrar"))

' Si el valor de la variable precio es mayor que 1000, entonces, aplicar descuento del 10%

If Precio > 1000 **Then**

Descuento = Precio * (10 / 100)

ActiveSheet.Range("A2").Value = 0,1

Else *' Sino → Aplicar descuento del 5%*

Descuento = Precio * (5 / 100)

ActiveSheet.Range("A2").Value = 0,05

End If

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

ActiveSheet.Range("A3").Value = Descuento

ActiveSheet.Range("A4").Value = Precio - Descuento

End Sub

Ejemplo 10.

Restar los valores de las casilla A1 y A2. Guardar el resultado en A3. Si el resultado es positivo o 0, poner la fuente de A3 en azul, sino ponerla en rojo.

Sub Condicional_Else2()

ActiveSheet.Range("A3").Value = ActiveSheet.Range("A1").Value - _
ActiveSheet.Range("A2").Value

If ActiveSheet("A3").Value < 0 **Then**

ActiveSheet.Range("A3").Font.Color = RGB(255,0,0)

Else

ActiveSheet.Range("A3").Font.Color = RGB(0,0,255)

End If

End Sub

- **Estructuras If anidadas.**

No tiene que sorprenderle, dentro de una estructura if puede ir otra, y dentro de esta otra, y otra... Vea el ejemplo siguiente.

Ejemplo 11.

Comparar los valores de las casilla A1 y A2 de la hoja activa. Si son iguales, escribir en A3 "Los valores de A1 y A2 son iguales", si el valor de A1 es mayor que A2, escribir "A1 mayor que A2", sino, escribir "A2 mayor que A1" .

Sub Condicional()

```
If ActiveSheet.Range("A1").Value = ActiveSheet.Range("A2").Value Then  
    ActiveSheet.Range("A3").Value = "Los Valores de A1 y A2 son iguales"  
Else  
    If ActiveSheet.Range("A1").Value > ActiveSheet.Range("A2").Value Then  
        ActiveSheet.Range("A3").Value = "A1 mayor que A2"  
    Else  
        ActiveSheet.Range("A3").Value = "A2 mayor que A1"  
    End If  
End If
```

End Sub

Observe que la segunda estructura **If.Else.End If** queda dentro del **Else** de la primera estructura. Esta es una regla general, cuando pone un **End If**, este cierra siempre el último **If** (o **Else**) abierto.

• Operadores lógicos.

Estos operadores se utilizan cuando se necesitan evaluar dos o más condiciones para decidir si se ejecutan o no determinadas acciones.

Operador Lógico And (Y).

Utilizaremos este operador cuando sea preciso que para ejecutar un bloque de instrucciones se cumpla más de una condición. Observe que deberán cumplirse todas las condiciones . Vea el ejemplo siguiente.

Ejemplo 12.

Entrar el Nombre, la cantidad y el precio de un producto desde el teclado y guardarlos respectivamente en A1, A2 y A3. Calcular el total y guardarlo en A4. Si el total es superior a 10.000 y el nombre del producto es "Patatas", pedir un descuento, calcularlo el total descuento y guardarlo en A5, luego restar el descuento del total y guardarlo en A6.

Sub Ejemplo_12()

```
Dim Producto As String  
Dim Cantidad As Integer  
Dim Precio As Single  
Dim Total As Single  
Dim Descuento As Single  
Dim Total_Descuento As Single
```

```
Precio = 0
```

```
Producto = InputBox("Entrar Nombre del Producto", "Entrar")  
Precio = Val(InputBox("Entrar el precio", "Entrar"))  
Cantidad = Val(InputBox("Entrar la cantidad", "Entrar"))
```

```
Total = Precio * Cantidad
```

```
ActiveSheet.Range("A1").Value = Producto  
ActiveSheet.Range("A2").Value = Precio  
ActiveSheet.Range("A3").Value = Cantidad  
ActiveSheet.Range("A4").Value = Total
```

'Si total mayor que 10.000 y el producto es Patatas, aplicar descuento.

```
If Total > 10000 And Producto = "Patatas" Then  
    Descuento = Val(InputBox("Entrar Descuento", "Entrar"))  
    Total_Descuento = Total * (Descuento / 100)  
    Total = Total - Total_Descuento  
    ActiveSheet.Range("A5").Value = Total_Descuento  
    ActiveSheet.Range("A6").Value = Total  
End If
```

End Sub

Observe que para que se ejecute el bloque de instrucciones entre If.. End If deben cumplirse las dos condiciones que se evalúan, si falla cualquiera de las dos (o las dos a la vez), no se ejecuta dicho bloque.

Operador Lógico Or (O).

Utilizaremos este operador cuando sea preciso que para ejecutar un bloque de instrucciones se cumpla alguna de una serie de condiciones. Observe que sólo es necesario que se cumpla alguna de las condiciones que se evalúan. Vea el ejemplo siguiente.

Ejemplo 13.

Entrar el Nombre, la cantidad y el precio de un producto desde el teclado y guardarlos respectivamente en A1, A2 y A3. Calcular el total y guardarlo en A4. Si el total es superior a 10.000 o el nombre del producto el "Patatas", pedir un descuento, calcularlo el total descuento y guardarlo en A5, luego restar el descuento del total y guardarlo en A6.

Sub Ejemplo_13()

```
Dim Producto As String  
Dim Cantidad As Integer  
Dim Precio As Single  
Dim Total As Single  
Dim Descuento As Single  
Dim Total_Descuento As Single
```

```
Precio = 0
```

```
Producto = InputBox("Entrar Nombre del Producto", "Entrar")  
Precio = Val(InputBox("Entrar el precio", "Entrar"))  
Cantidad = Val(InputBox("Entrar la cantidad", "Entrar"))
```

```
Total = Precio * Cantidad
```

```
ActiveSheet.Range("A1").Value = Producto  
ActiveSheet.Range("A2").Value = Precio  
ActiveSheet.Range("A3").Value = Cantidad  
ActiveSheet.Range("A4").Value = Total
```

'Si total mayor que 10.000 o el producto es Patatas, aplicar descuento.

```
If Total > 10000 Or Producto = "Patatas" Then  
    Descuento = Val(InputBox("Entrar Descuento", "Entrar"))  
    Total_Descuento = Total * (Descuento / 100)  
    Total = Total - Total_Descuento  
    ActiveSheet.Range("A5").Value = Total_Descuento  
    ActiveSheet.Range("A6").Value = Total
```

```
End If
```

End Sub

Observe que para que se ejecute el bloque de instrucciones entre If.. End If sólo es necesario que se cumpla alguna de las dos condiciones que se evalúan (o las dos a la vez). Sólo cuando no se cumple ninguna de las dos no se ejecutan las instrucciones del bloque.

Operador Lógico Not (no).

Este operador se utiliza para ver si NO se cumple una condición. El siguiente ejemplo hace lo mismo que el ejemplo 7 pero utilizando el operador Not.

Ejemplo 14.

Entrar una cantidad que representa el precio de algo por el teclado con la instrucción InputBox y guardarlo en la celda A1 de la hoja activa. Si el valor entrado desde el teclado (y guardado en A1) es superior a 1000, pedir descuento con otro InputBox y guardarlo en la casilla A2 de la hoja activa. Calcular en A3, el precio de A1 menos el descuento de A2.

Sub Ejemplo_14()

Dim Precio **As Integer**

Dim Descuento **As Integer**

Precio = 0

Descuento = 0

Precio = Val(InputBox("Entrar el precio", "Entrar"))

' Si el valor de la variable precio NO es menor igual 1000, entonces, pedir descuento

If Not (Precio <= 1000) **Then**

Descuento = Val(InputBox("Entrar Descuento", "Entrar"))

End If

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

ActiveSheet.Range("A2").Value = Descuento

ActiveSheet.Range("A3").Value = Precio - Descuento

End Sub

- **Tablas de la verdad.**

Vea las tablas siguientes para ver los resultados de evaluar dos condiciones con el operador And y con el operador Or.

And.

Condición1	Condición2	Resultado
Falsa	Falsa	Falso
Falsa	Cierta	Falso
Cierta	Falsa	Falso
Cierta	Cierta	Cierto

Or.

Condición1	Condición2	Resultado
Falsa	Falsa	Falso
Falsa	Cierta	Cierto
Cierta	Falsa	Cierto
Cierta	Cierta	Cierto

Observe que con el operador AND deben de cumplirse todas las condiciones (dos o veinticinco) para que el resultado sea cierto. Con el operador OR sólo es necesario que se cumpla una (de las dos o de las veinticinco) para que el resultado sea cierto.

- **Estructura Select Case.**

En ocasiones se dará el caso que en función del valor o rango de valores que pueda tener **una** variable, **una** casilla, **una** expresión, etc. deberán llevarse a cabo diferentes acciones o grupos de acciones. Vea el ejemplo siguiente.

Ejemplo 15.

Macro que suma, resta, multiplica o divide los valores de las casillas A1 y A2 dependiendo de si B1 contiene el signo +, -, x, :. El resultado lo deja en A3. Si en B1 no hay ninguno de los signos anteriores en A3 debe dejarse un 0.

Sub Ejemplo_15()

Dim Signo **As String**

Dim Valor1 **As Integer**, Valor2 **As Integer**, Total **As Integer**

Valor1 = ActiveSheet.Range("A1").Value

Valor2 = ActiveSheet.Range("A2").Value

Signo = ActiveSheet.Range("B1").Value

Total=0

If Signo = "+" **Then**

Total = Valor1 + Valor2

End if

If Signo = "-" **Then**

Total = Valor1 - Valor2

End if

If Signo = "x" **Then**

Total = Valor1 * Valor2

End if

If Signo = ":" **Then**

Total = Valor1 / Valor2

End if

ActiveCell.Range("A3").Value = Total

End Sub

Observe que en el ejemplo anterior todas las instrucciones if evalúan la misma variable. El programa funciona correctamente pero para estos casos es mejor utilizar la instrucción Select Case, el motivo principal es por legibilidad y elegancia. Select Case tiene la sintaxis siguiente,

Select Case Expresión

Case valores :

Instrucciones.

Case valores :

Instrucciones.

.

.

Case valores:

Instrucciones.

Case Else

Instrucciones en caso que no sean ninguno de los valores anteriores.

End Select

Vea el ejemplo anterior solucionado con esta estructura.

Ejemplo 16.

Sub Ejemplo_16()

Dim Signo **As String**

Dim Valor1 **As Integer**, Valor2 **As Integer**, Total **As Integer**

Valor1 = ActiveSheet.Range("A1").Value

Valor2 = ActiveSheet.Range("A2").Value

Signo = ActiveSheet.Range("A3").Value

Select Case signo

Case "+"

Total = Valor1 + Valor2

Case "-"

Total = Valor1 - Valor2

Case "x"

Total = Valor1 * Valor2

Case ":"

Total = Valor1 / Valor2

Case Else

Total = 0

End Select

ActiveCell.Range("A3").Value = Total

End Sub

Vea el ejemplo siguiente donde cada sentencia Case evalúa un rango de valores.

Ejemplo 17.

Programa que pide tres notas de un alumno mediante la función InputBox. Las notas van a parar respectivamente a las casillas A1, A2 y A3 de la hoja activa. El programa calcula la media y la deja en A4. Si la media está entre 0 y 2 deja en A5 el mensaje "Muy deficiente", si la nota es 3 deja en A5 el mensaje "Deficiente", si la nota es 4 deja "Insuficiente", si es 5 "Suficiente", si es 6 "Bien", si está entre 7 y 8 deja "Notable", si es mayor que 8 deja "Sobresaliente".

Sub Ejemplo_17()

Dim Nota1 **As Integer**, Nota2 **As Integer**, Nota3 **As Integer**

Dim Media **As Single**

Nota1 = Val(InputBox("Entrar Nota primera evaluación", "Nota"))

Nota2 = Val(InputBox("Entrar Nota Segunda evaluación", "Nota"))

Nota3 = Val(InputBox("Entrar Nota Tercera evaluación", "Nota"))

Media = (Nota1 + Nota2 + Nota3) / 3

ActiveSheet.Range("A1").Value = Nota1

ActiveSheet.Range("A2").Value = Nota2

ActiveSheet.Range("A3").Value = Nota3

ActiveSheet.Range("A4").Value = Media

Select Case Media

Case 0 To 2

ActiveSheet.Range("A5").Value = "Muy deficiente"

Case 3

ActiveSheet.Range("A5").Value = "Deficiente"

Case 4

ActiveSheet.Range("A5").Value = "Insuficiente"

Case 5

ActiveSheet.Range("A5").Value = "Suficiente"

Case 6

ActiveSheet.Range("A5").Value = "Bien"

Case 7 To 8

ActiveSheet.Range("A5").Value = "Notable"

Case >8

ActiveSheet.Range("A5").Value = "Sobresaliente"

End Select

End Sub

- **Funciones de comprobación.**

Antes de terminar con el tema de condicionales veremos unas funciones que nos serán útiles a la hora de comprobar o validar el tipo de los datos ent rados desde teclado o simplemente los datos contenidos en una casilla.

Volvamos al ejemplo que codificamos de la manera siguiente.

Sub Ejemplo_16()

Dim Signo **As String**

Dim Valor1 **As Integer**, Valor2 **As Integer**, Total **As Integer**

Valor1 = ActiveSheet.Range("A1").Value

Valor2 = ActiveSheet.Range("A2").Value

Signo = ActiveSheet.Range("A3").Value

Select Case signo

Case "+"

Total = Valor1 + Valor2

Case "-"

Total = Valor1 - Valor2

Case "x"

Total = Valor1 * Valor2

Case ":"

Total = Valor1 / Valor2

Case Else

Total = 0

End Select

ActiveCell.Range("A3").Value = Total

End Sub

Imagine que en alguna de las casillas que se deben operar no hubiera ningún valor o bien datos alfanuméricos. Al ejecutar la macro se producirá un error. Aunque con Visual Basic se puede controlar el flujo del programa cuando se produce un error imprevisto, para solucionar este problema utilizaremos una función de comprobación que nos diga si en las casillas A1 y A2 hay valores adecuados (numéricos) para proseguir con la ejecución de la macro, en caso contrario se mostrará un error y no se ejecutará ninguna de las operaciones.

La función que utilizaremos es **IsNumeric**(expresión), esta función devuelve un valor **True** si la expresión que se evalúa es un valor numérico, en caso contrario devuelve **False**. Vea como quedaría el programa. También se utiliza la función **IsEmpty** para comprobar si en B1 hay algo, **IsEmpty**(Expresión) evalúa si expresión está vacía, devuelve **True** si es así y **False** en caso contrario.

Ejemplo 18.

Sub Ejemplo_18()

Dim Signo **As** String

Dim Valor1 **As** Integer, Valor2 **As** Integer, Total **As** Integer

Dim Continuar **As** Boolean

Valor1 = ActiveSheet.Range("A1").Value

Valor2 = ActiveSheet.Range("A2").Value

Signo = ActiveSheet.Range("A3").Value

Continuar = True

' Si en la casilla A1 no hay un valor numérico

If Not IsNumeric(ActiveSheet.Range("A1")) **Then**

MsgBox Prompt:="En la casilla A1 no hay ningún valor numérico", Title:="ERROR"

Continuar= False

End If

' Si en la casilla A2 no hay un valor numérico

If not IsNumeric(ActiveSheet.Range("A2")) **Then**

MsgBox Prompt:="En la casilla A2 no hay ningún valor numérico", Title:="ERROR"

Continuar= False

End If

If IsEmpty(ActiveSheet.Range("B1")) **Then**

MsgBox Prompt:="la casilla B1 está vacía", Title:="ERROR"

Continuar= False

End If

If Continuar **Then**

Select Case signo

Case "+"

Total = Valor1 + Valor2

Case "-"

Total = Valor1 - Valor2

Case "x"

Total = Valor1 * Valor2

Case "/"

Total = Valor1 / Valor2

Case Else

Total = 0

End Select


```
ActiveCell.Range("A3").Value = Total
```

```
End if
```

```
End Sub
```

En lugar de los tres If de comprobación se hubiera podido utilizar el operador OR de la manera siguientes,

```
If not IsNumeric(ActiveSheet.Range("A1")) Or not IsNumeric(ActiveSheet.Range("A2")) _  
Or IsEmpty(ActiveSheet.Range("B1")) Then
```

```
MsgBox Prompt:="Debe entrar números en A1 y A2 y un signo (+,-,x, :) en B1,  
Title:"ERROR"
```

```
Else
```

```
' Instrucciones de las operaciones
```

```
.....
```

```
End if
```

Lista de Funciones de Comprobación.

IsNumérico(Expresión). Comprueba si expresión tiene un valor que se puede interpretar como numérico.

IsDate(Expresión). Comprueba si expresión tiene un valor que se puede interpretar como fecha.

IsEmpty(Expresión). Comprueba que expresión tenga algún valor, que se haya inicializado.

IsError(Expresión). Comprueba si expresión devuelve algún valor de error.

IsArray(Expresión). Comprueba si expresión (una variable) es un array o no.

IsObject(Expresión). Comprueba si expresión (una variable) representa una variable tipo objeto.

IsNull(Expresión). Comprueba si expresión contiene un valor nulo debido a datos no válidos.

Nothing. No es propiamente una función, sirve para comprobar si una variable objeto esta asociada a un objeto antes de hacer cualquier operación con ella. Recuerde que para trabajar con una variable objeto antes debe asignarse a uno (mediante la instrucción **Set**), en caso contrario se producirá un error en el programa cuando utilice el objeto y se detendrá su ejecución.

```
Sub Obj()
```

```
Dim R As Range
```

```
.
```

```
.
```

```
' Si la variable R es Nothing es que no ha sido asignada, no se puede trabajar con ella
```

```
If R Is Nothing Then
```

```
MsgBox Prompt := "La variable Objeto no ha sido asignada", Buttons:=vbOk, _  
Title := "Error"
```

```
Else
```

```
R.Value = "Hola"
```

```
End If
```

```
.
```

```
.
```

```
End Sub
```

- **La función MsgBox.**

Esta función muestra un mensaje en un cuadro de diálogo hasta que el usuario pulse un botón. La función devuelve un dato tipo Integer en función del botón pulsado por el usuario. A la hora de invocar esta función, se permiten diferentes tipos de botones.

Sintaxis de MsgBox.

MsgBox(Mensaje, Botones, Título, Archivo de ayuda, contexto)

Mensaje : Obligatorio, es el mensaje que se muestra dentro del cuadro de diálogo.

Botones : Opcional. Es un número o una suma de números o constantes (vea tabla Valores para botones e Iconos), que sirve para mostrar determinados botones e iconos dentro del cuadro de diálogo. Si se omite este argumento asume valor 0 que corresponde a un único Botón OK.

Título : Opcional. Es el texto que se mostrará en la barra del título del cuadro de diálogo.

Archivo de Ayuda : Opcional. Si ha asignado un texto de ayuda al cuadro de diálogo, aquí debe especificar el nombre del archivo de ayuda donde está el texto.

Context: Opcional. Es el número que sirve para identificar el texto al tema de ayuda correspondiente que estará contenido en el archivo especificado en el parámetro Archivo de Ayuda.

Tabla para botones e iconos del cuadro MsgBox. (Tabla copiada del archivo de ayuda de Microsoft Excel).

Constante	Valor	Descripción
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.
VbDefaultButton1	0	El primer botón es el predeterminado.
VbDefaultButton2	256	El segundo botón es el predeterminado.
VbDefaultButton3	512	El tercer botón es el predeterminado.
VbDefaultButton4	768	El cuarto botón es el predeterminado.
VbApplicationModal	0	Aplicación modal; el usuario debe responder al cuadro de mensajes antes de poder seguir trabajando en la aplicación actual.
VbSystemModal	4096	Sistema modal; se suspenden todas las aplicaciones hasta que el usuario responda al cuadro de mensajes.

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, el tercer grupo (0, 256, 512) determina el botón predeterminado y el cuarto grupo (0, 4096) determina la modalidad del cuadro de mensajes. Cuando se suman números para obtener el valor final del argumento buttons, se utiliza solamente un número de cada grupo.

Nota Estas constantes las especifica Visual Basic for Applications. Por tanto, el nombre de las mismas puede utilizarse en cualquier lugar del código en vez de sus valores reales.

Los valores que puede devolver la función msgbox en función del botón que pulse el usuario se muestran en la tabla siguiente.

Tabla de valores que puede devolver MsgBox. (Tabla copiada del archivo de ayuda de Microsoft Visual Basic para aplicaciones).

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

Ejemplos de MsgBox.

Sub Tal()

```
.  
' El cuadro Muestra los botones Si y No y un icono en forma de interrogante. Cuando se pulsa  
' un botón, el valor lo recoge la variable X. En este caso los valores devueltos pueden ser 6 o 7  
' que corresponden respectivamente a las constantes VbYes y VbNo, observe la instrucción If de  
' después.
```

```
X = MsgBox("Desea Continuar", vbYesNo + vbQuestion, "Opción",,)
```

```
' Se ha pulsado sobre botón Si
```

```
If X = vbYes Then
```

```
.....
```

```
Else ' Se ha pulsado sobre botón No
```

```
.....
```

```
End If
```

```
.
```

```
.
```

End Sub

Algunas veces puede que le interese simplemente desplegar un cuadro MsgBox para mostrar un mensaje al usuario sin que se requiera recoger ningún valor. En este caso puede optar por la forma siguiente,

```
MsgBox Prompt:= "Hola usuaria, Ha acabado el proceso", Buttons:=VbOkOnLy _  
Title:= "Mensaje"
```

Lo que no puede hacer porque Visual Basic daría error es poner la primera forma sin igualarla a ninguna variable. Por ejemplo, la expresión siguiente es incorrecta,

```
MsgBox ("Hola usuario, Ha acabado el proceso", VbOkOnly, "Mensaje")
```

Seria correcto poner

```
X=MsgBox ("Hola usuario, Ha acabado el proceso", VbOkOnly, "Mensaje")
```

En este caso, aunque X reciba un valor, luego no se utiliza para nada, es decir simplemente se pone para que Visual Basic dé error.

- **La instrucción With.**

Suponemos que llegado a este punto le parecerá engorroso tener que referirse a los objetos siguiendo toda o casi toda la jerarquía. Ya hemos indicado que es mejor hacerlo de esta manera porque el programa gana en claridad y elegancia y, consecuentemente, el programador gana tiempo a la hora de hacer modificaciones o actualizaciones. La sentencia **With** le ayudará a tener que escribir menos código sin que por esto el programa pierda en claridad. Concretamente esta función sirve para ejecutar una serie de acciones sobre un mismo Objeto. Su sintaxis es la siguiente.

```
With Objeto
    Instrucciones
End With
```

Repetiremos el ejemplo 13 utilizando esta sentencia. Observe como con **With** se hace referencia al objeto **ActiveSheet**.

Ejemplo 19.

Entrar el Nombre, la cantidad y el precio de un producto desde el teclado y guardarlos respectivamente en A1, A2 y A3. Calcular el total y guardarlo en A4. Si el total es superior a 10.000 o el nombre del producto el "Patatas", pedir un descuento, calcularlo el total descuento y guardarlo en A5, luego restar el descuento del total y guardarlo en A6.

Sub Ejemplo_19()

```
Dim Producto As String
Dim Cantidad As Integer
Dim Precio As Single
Dim Total As Single
Dim Descuento As Single
Dim Total_Descuento As Single
```

```
Precio = 0
```

```
Producto = InputBox("Entrar Nombre del Producto", "Entrar")
```

```
Precio = Val(InputBox("Entrar el precio", "Entrar"))
```

```
Precio = Val(InputBox("Entrar la cantidad", "Entrar"))
```

```
Total = Precio * Cantidad
```

```
With ActiveSheet
```

```
    .Range("A1").Value = Producto
```

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

```
    .Range("A3").Value = Cantidad
```

```
    .Range("A4").Value = Total
```

```
End With
```

```
' Si total mayor que 10.000 o el producto es Patatas, aplicar descuento.
```

```
If Total > 10000 Or Producto = "Patatas" Then
```

```
    Descuento = Val(InputBox("Entrar Descuento", "Entrar"))
```

```
    Total_Descuento = Total * (Descuento / 100)
```

```
    Total = Total - Total_Descuento
```

```
With ActiveSheet
```

```
    .Range("A5").Value = Total_Descuento
```

```
    .Range("A6").Value = Total
```

```
End With
```

```
End If
```

```
End Sub
```

Estructuras Repetitivas.

Este tipo de estructuras permiten ejecutar más de una vez un mismo bloque de sentencias.

Ejemplo 20.

Supongamos que tenemos que hacer un programa para entrar las notas de una clase de 5 alumnos que se guardaran respectivamente en las celdas de A1 a A5 de la hoja activa. Después hacer la media que se guardará en A6. Con las estructuras vistas hasta ahora, podríamos hacer:

Sub Ejemplo_20 ()

DimNota **As Integer**

DimMedia **As Single**

Media = 0

Nota = Val(InputBox("Entrar la 1 Nota : ","Entrar Nota"))
ActiveSheet.Range("A1").Value = Nota
Media = Media + Nota

Nota = Val(InputBox("Entrar la 1 Nota : ","Entrar Nota"))
ActiveSheet.Range("A2").Value = Nota
Media = Media + Nota

Nota = Val(InputBox("Entrar la 1 Nota : ","Entrar Nota"))
ActiveSheet.Range("A3").Value = Nota
Media = Media + Nota

Nota = Val(InputBox("Entrar la 1 Nota : ","Entrar Nota"))
ActiveSheet.Range("A4").Value = Nota
Media = Media + Nota

Nota = Val(InputBox("Entrar la 1 Nota : ","Entrar Nota"))
ActiveSheet.Range("A5").Value = Nota
Media = Media + Nota

Media = Media / 5

ActiveSheet.Range("A6").Value = Media

End Sub

Observe que este programa repite el siguiente bloque de sentencias, 5 veces.

```
Nota = Val(InputBox("Entrar la 1 Nota : ","Entrar Nota"))
ActiveSheet.Range("A5").Value = Nota
Media = Media + Nota
```

Para evitar esta tipo de repeticiones de código, los lenguajes de programación incorporan instrucciones que permiten la repetición de bloques de código.

- **Estructura repetitiva Para (for).**

Esta estructura sirve para repetir la ejecución de una sentencia o bloque de sentencias, un número definido de veces. La estructura es la siguiente:

```
Para var =Valor_Inicial Hasta Valor_Final Paso Incremento Hacer
  Inicio
    Sentencia 1
    Sentencia 2
    .
    .
    Sentencia N
  Fin
```

Var es una variable que la primera vez que se entra en el bucle se iguala a *Valor_Inicial*, las sentencias del bucle se ejecutan hasta que **Var** llega al *Valor_Final*, cada vez que se ejecutan el bloque de instrucciones **Var** se incrementa según el valor de *Incremento*.

En Visual Basic para Excel la estructura Para se implementa con la instrucción **For ... Next**.

```
For Variable = Valor_Inicial To Valor_Final Step Incremento
  Sentencia 1
  Sentencia 2
  .
  .
  Sentencia N
Next Variable
```

** Si el incremento es 1, no hace falta poner Step 1.*

Ejemplo 21.

Entrar 10 valores utilizando la función `InputBox`, sumarlos y guardar el resultado en la casilla A1 de la hoja activa.

```
Sub Ejemplo_21()
```

```
  Dim i As Integer
  Dim Total As Integer
  Dim Valor As Integer

  For i=1 To 10
    Valor= Val(InputBox("Entrar un valor","Entrada"))
    Total = Total + Valor
  Next i

  ActiveCell.Range("A1").Value = Total
```

```
End Sub
```

Recorrer casillas de una hoja de cálculo.

Una operación bastante habitual cuando se trabaja con Excel es el recorrido de rangos de casillas para llenarlas con valores, mirar su contenido, etc. Las estructuras repetitivas son imprescindibles para recorrer grupos de celdas o rangos. Vea los siguientes ejemplos para ver ejemplos de utilización de estructuras repetitivas para recorrer rangos de casillas, observe la utilización de las propiedades **Cells** y **Offset**.

Propiedad Cells.

Ya conoce esta propiedad, sirve para referenciar una celda o un rango de celdas según coordenadas de fila y columna.

Ejemplo 22

Llenar el rango de las casillas A1..A5 con valores pares consecutivos empezando por el 2.

Sub Ejemplo_22()

```
Dim Fila As Integer
Dim i As Integer

Fila = 1
For i=2 To 10 Step 2
    ActiveSheet.Cells(Fila, 1).Value = i
    Fila = Fila+1
Next i
```

End Sub

Ejemplo 23.

Llenar un rango de filas, empezando por una celda, que se debe especificar desde teclado, con una serie de 10 valores correlativos (comenzando por el 1).

Sub Ejemplo_23()

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

Casilla_Inicial = InputBox("Introducir la casilla Inicial : ", "Casilla Inicial")

ActiveSheet.Range(Casilla_Inicial).Activate

' Coger el valor de fila de la celda activa sobre la variable Fila
Fila = ActiveCell.Row
' Coger el valor de columna de la celda activa sobre la variable Fila
Columna = ActiveCell.Column

For i = 1 To 10
    ActiveSheet.Cells(Fila, Columna).Value = i
    Fila = Fila + 1
Next i
```

End Sub

Propiedades ROW y COLUMN.

Como habrá deducido del ejemplo anterior devuelven la fila y la columna de un objeto range. En el ejemplo anterior se utilizaban concretamente para obtener la fila y la columna de la casilla activa.

Otra forma de solucionar el ejemplo 23 sería.

Sub Ejemplo_23()

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

Casilla_Inicial = InputBox("Introducir la casilla Inicial : ", "Casilla Inicial")

ActiveSheet.Range(Casilla_Inicial).Activate

Fila = 1

For i = 1 To 10
    ActiveSheet.Range(Casilla_Inicial).Cells(Fila, 1).Value = i
    Fila = Fila + 1
Next i
```

End Sub

** Recuerde que cuando utilizamos **Cells** como propiedad de un rango (Objeto Range), **Cells** empieza a contar a partir de la casilla referenciada por **Range**.

Ejemplo 24.

El mismo con el que introducíamos el tema (ejemplo 20), pero utilizando el **for** y propiedad **Cells**

Sub Ejemplo_24()

```
Dim Nota As Integer
Dim Media As Single
Dim Fila As Integer

Media = 0
For Fila = 1 To 5

    Nota = Val(InputBox("Entrar la " & Fila & " Nota : ", "Entrar Nota"))
    ActiveSheet.Cells(Fila, 1) = Nota
    Media = Media + Nota

Next Fila
Media = Media / 5
ActiveSheet.Cells(6, 1).Value = Media
```

End Sub

Propiedad Offset.

Esta propiedad es también muy útil a la hora de recorrer rango. **Offset**, que significa desplazamiento, es una propiedad del objeto **Range** y se utiliza para referenciar una casilla situada a n Filas y n Columnas de una casilla dada. Vea los ejemplos siguientes.

ActiveSheet.Range("A1").Offset(2, 2).Value = "Hola" ' Casilla C3 = Hola, 2 filas y 2 columnas desde A1.

ActiveCell.Offset(5,1).Value = "Hola" ' 5 Filas por debajo de la casilla Activa = Hola

ActiveCell.Offset(2,2).Activate ' Activar la casilla que está 2 filas y 2 columnas de la activa

Ejemplo 25.

El mismo con el que introducíamos el tema (ejemplo 20), pero utilizando el **For** y propiedad **Offset**

Sub Ejemplo_25()

DimNota **As Integer**

DimMedia **As Single**

DimFila **As Integer**

Media = 0

ActiveSheet.Range("A1").Activate

For Fila = 0 **To** 4

Nota=Val(Input Box("Entrar la " & Fila+1 & " Nota : ", "Entrar Nota"))

ActiveCell.Offset(Fila, 0).Value = Nota

Media = Media + Nota

Next Fila

Media = Media / 5

ActiveCell.Offset(6, 1).Value = Media

End Sub

Ejemplo 26.

El mismo con el que introducíamos el tema (ejemplo 20), pero utilizando el **For** y propiedad **Offset**.
Observe que ahora vamos cambiando de celda activa.

Sub Ejemplo_26()

DimNota **As Integer**

DimMedia **As Single**

Dim i **As Integer**

Media = 0

ActiveSheet.Range("A1").Activate

For i = 1 **To** 5

Nota=Val(InputBox("Entrar la " & i & " Nota : ", "Entrar Nota"))

ActiveCell.Value = Nota

Media = Media + Nota
' *Hacer activa la casilla situada una fila por debajo de la actual*
ActiveCell.Offset(1, 0).Activate

Next Fila
Media = Media / 5
ActiveCell.Value = Media

End Sub

Observe la diferencia entre los ejemplos 25 y 26, ambos utilizan la propiedad **Offset** de diferente forma, en el ejemplo 25 la casilla activa siempre es la misma A1, **Offset** se utiliza para referenciar una casilla a partir de esta. En A26 se va cambiando de casilla activa cada vez de forma que, cuando termina la ejecución del programa la casilla activa es A6.

Cuando utilizar cada método, como casi siempre depende de lo que se pretenda hacer. Aquí es bastante claro, si le interesa que no cambie la casilla utilice **Offset** como en el ejemplo 25, en caso que interese que vaya cambiando, haga como en el Ejemplo 6. Por supuesto hay muchas variantes sobre el estilo de recorrer Celdas, tanto con **Cells** como con **Offset**, solo tiene que ir probando y, como casi siempre, el que más le guste.

• Estructura repetitiva Do While..Loop (Hacer Mientras).

La estructura repetitiva **for** se adapta perfectamente a aquellas situaciones en que se sabe previamente el número de veces que se ha de repetir un proceso, entrar veinte valores, recorrer cincuenta celdas, etc. Pero hay ocasiones o casos en los que no se sabe previamente el número de veces que se debe repetir un proceso. Por ejemplo, suponga que ha de recorrer un rango de filas en los que no se sabe cuantos valores habrá (esto es, cuantas filas llenas habrá), en ocasiones puede que hayan veinte, en ocasiones treinta, en ocasiones ninguna, etc. Para estos casos la estructura **for** no es adecuada y deberemos recurrir a la sentencia **Do While..Loop** en alguna de sus formas. Esta estructura repetitiva está controlada por una o varias condiciones, la repetición del bloque de sentencias dependerá de si se va cumpliendo la condición o condiciones.

Hacer Mientras (se cumpla la condición)
Sentencia1
Sentencia2
.
.
Sentencia N
Fin Hacer Mientras

En Visual Basic

Do While (se cumpla la condición)
Sentencia1
Sentencia2
.
.
Sentencia N
Loop

** Los ejemplos que veremos a continuación sobre la instrucción **Do While..Loop** se harán sobre una base de datos. Una base de datos en Excel es simplemente un rango de celdas en que cada fila representa un registro y cada columna un campo de registro, la primera fila es la que da nombre a los campos. Para nuestra base de datos utilizaremos los campos siguientes, *Nombre*, *Ciudad*, *Edad*, *Fecha*. Ponga estos títulos en el rango A1:D1 de la Hoja1 (En A1 ponga Nombre, en B1 ponga Ciudad, en C1 ponga Edad y en D1 Fecha), observe que los datos se empezarán a entrar a partir de A2.

Ejemplo 27.

Programa para entrar registros en la base de datos. Cada campo se entra con InputBox. El programa va pidiendo datos mientras se entre un valor en el InputBox correspondiente al nombre, es decir cuando al preguntar el nombre no se entre ningún valor, terminará la ejecución del bloque encerrado entre **Do While...Loop** Observe la utilización de la propiedad **Offset** para colocar los datos en las celdas correspondientes.

Sub Ejemplo_27()

DimNombre **As String**

Dim Ciudad **As String**

DimEdad **As Integer**

Dimfecha **As Date**

' Activar hoja1

WorkSheets("Hoja1").Activate

' Activar casilla A2

ActiveSheet.Range("A2").Activate

Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")

' Mientras la variable Nombre sea diferente a cadena vacía

Do While Nombre <> ""

Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")

Edad = Val(InputBox("Entre la Edad : ", "Edad"))

Fecha=Cdate(InputBox("Entra la Fecha : ", "Fecha"))

' Copiar los datos en las casillas correspondientes

With ActiveCell

.Value = Nombre

.Offset(0,1).Value = Ciudad

.Offset(0,2).Value = Edad

.Offset(0,3).Value = fecha

End With

'Hacer activa la celda de la fila siguiente a la actual

ActiveCell.Offset(1,0).Activate

Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")

Loop

End Sub

Ejemplo 28.

Preste especial atención a este ejemplo ya que seguro que el código que viene a continuación lo utilizará en muchas ocasiones. Antes que nada observe el ejemplo anterior, fíjese en que siempre empezamos a llenar el rango de la hoja a partir de la celda A2, esto tiene una nefasta consecuencia, la segunda vez que ejecute la macro machacará los datos de A2:D2 y si continua ejecutando machacará los datos de los rangos siguientes. Una solución sería observar cual es la casilla vacía siguiente y cambiar en la instrucción **ActiveSheet.Range("A2").Activate** , la referencia **A2** por la que corresponde a la primera casilla vacía de la columna A. El código que le mostramos a continuación hará esto por nosotros, es decir recorrerá una fila de celdas a partir de A1 hasta encontrar una vacía y dejará a esta como celda activa para que la entrada de datos comience a partir de ella.

Sub Ejemplo_28()

```
.  
.  
    ' Activar hoja1  
    WorkSheets("Hoja1").Activate  
    ' Activar casilla A2  
    ActiveSheet.Range("A1").Activate  
  
    ' Mientras la celda activa no esté vacía  
    Do While Not IsEmpty(ActiveCell)  
        ' Hacer activa la celda situada una fila por debajo de la actual  
        ActiveCell.Offset(1,0).Activate  
  
    Loop  
    .  
    .
```

End Sub

Ejemplo 29.

Es la unión de los dos programas anteriores. Es decir habrá un bucle **Do While** que buscará la primera casilla vacía de la base da datos y otro para pedir los valores de los campos hasta que se pulse Enter en Nombre.

Sub Ejemplo_28()

```
    Dim Nombre As String  
    Dim Ciudad As String  
    Dim Edad As Integer  
    Dim fecha As Date  
  
    WorkSheets("Hoja1").Activate  
    ActiveSheet.Range("A1").Activate  
  
    ' Buscar la primera celda vacía de la columna A y convertirla en activa  
    Do While Not IsEmpty(ActiveCell)  
        ActiveCell.Offset(1,0).Activate  
  
    Loop  
  
    Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")  
  
    ' Mientras la variable Nombre sea diferente a cadena vacía  
    Do While Nombre <> ""  
        Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")  
        Edad = Val(InputBox("Entre la Edad : ", "Edad"))  
        Fecha=Cdate(InputBox("Entra la Fecha : ", "Fecha"))  
  
        With ActiveCell  
            .Value = Nombre  
            .Offset(0,1).Value = Ciudad  
            .Offset(0,2).Value = Edad  
            .Offset(0,3).value = fecha  
        End With  
  
        ActiveCell.Offset(1,0).Activate  
        Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")  
  
    Loop  
  
End Sub
```

Cuando se tienen que entrar desde el teclado conjuntos de valores, algunos programadores y usuarios prefieren la fórmula de que el programa pregunte si se desean entrar más datos, la típica pregunta ¿Desea Introducir más datos ?, si el usuario contesta Sí, el programa vuelve a ejecutar las instrucciones correspondientes a la entrada de datos, si contesta que no se finaliza el proceso, observe como quedaría nuestro bucle de entrada de datos con este sistema.

```
Mas_datos = vbYes
Do While Mas_Datos = vbYes

    Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")
    Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")
    Edad = Val(InputBox("Entre la Edad : ", "Edad"))
    Fecha=Cdate(InputBox("Entra la Fecha : ", "Fecha"))

    With ActiveCell
        .Value = Nombre
        .Offset(0,1).Value = Ciudad
        .Offset(0,2).Value = Edad
        .Offset(0,3).value = fecha
    End With

    ActiveCell.Offset(1,0).Activate

    ' Preguntar al usuario si desea entrar otro registro.
    Mas_datos = MsgBox("Otro registro ?", vbYesNo+vbQuestion,"Entrada de datos")
```

Loop

** Observe que es necesaria la línea anterior al bucle **Mas_datos = vbYes**, para que cuando se evalúe la condición por vez primera esta se cumpla y se ejecuten las sentencias de dentro del bucle, Mas_datos es una variable de tipo **Integer**. Vea la sección siguiente donde se estudia una variante de la estructura **Do While** que es más adecuada para este tipo de situaciones.

- **Estructura Do..Loop While.**

El funcionamiento de esta estructura repetitiva es similar a la anterior salvo que la condición se evalúa al final, la inmediata consecuencia de esto es que las instrucciones del cuerpo del bucle se ejecutaran al menos una vez. Observe que para nuestra estructura de entrada de datos vista en el último apartado de la sección anterior esta estructura es más conveniente, al menos más elegante, si vamos a entrar datos, al menos uno entraremos, por tanto las instrucciones del cuerpo del bucle se deben ejecutar al menos una vez, luego ya decidiremos si se repiten o no.

```
Do
Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")
Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")
Edad = Val(InputBox("Entre la Edad : ", "Edad"))
Fecha=Cdate(InputBox("Entra la Fecha : ", "Fecha"))

With ActiveCell
.Value = Nombre
.Offset(0,1).Value = Ciudad
.Offset(0,2).Value = Edad
.Offset(0,3).value = fecha
End With

ActiveCell.Offset(1,0).Activate

Mas_datos = MsgBox("Otro registro ?", vbYesNo+vbQuestion,"Entrada de datos")

'Mientras Mas_Datos = vbYes
Loop While Mas_Datos = vbYes
```

Observe que en este caso no es necesario la línea Mas_Datos = vbYes antes de **Do** para forzar la entrada en el bucle ya que la condición va al final.

- **Estructura Do..Loop Until (Hacer.. Hasta que se cumpla la condición).**

Es otra estructura que evalúa la condición al final observe que la interpretación es distinta ya que el bucle se va repitiendo **HASTA que se cumple la condición**, no MIENTRAS se cumple la condición. Cual de los dos utilizar, pues, no se sorprenda, la que entienda mejor o le guste más.

La entrada de datos con este bucle quedaría

```
Do
Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")
Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")
Edad = Val(InputBox("Entre la Edad : ", "Edad"))
Fecha=Cdate(InputBox("Entra la Fecha : ", "Fecha"))

With ActiveCell
.Value = Nombre
.Offset(0,1).Value = Ciudad
.Offset(0,2).Value = Edad
.Offset(0,3).value = fecha
End With

ActiveCell.Offset(1,0).Activate

Mas_datos = MsgBox("Otro registro ?", vbYesNo+vbQuestion,"Entrada de datos")

'Hasta que Mas_Datos sea igual a vbNo
Loop Until Mas_Datos= vbNo
```

- **Estructura For Each.**

Este bucle se utiliza básicamente para ejecutar un grupo de sentencias con los elementos de una colección o una matriz (pronto veremos los que es). Recuerde que una colección es un conjunto de objetos, hojas, rangos, etc. Vea el ejemplo siguiente que se utiliza para cambiar los nombres de las hojas de un libro de trabajo.

Ejemplo 29.

Programa que pregunta el nombre para cada hoja de un libro de trabajo, si no se pone nombre a la hoja, queda el que tiene.

Sub Ejemplo_29()

Dim Nuevo_Nombre **As String**

Dim Hoja **As WorkSheet**

' Para cada hoja del conjunto WorkSheets

For Each Hoja **In** WorkSheets

Nuevo_Nombre=InputBox("Nombre de la Hoja : " & Hoja.Name,"Nombrar Hojas")

If Nueva_Nombre <> "" **Then**

Hoja.Name=Nuevo_nombre

End if

Next

End Sub

** Hoja va referenciando cada una de las hojas del conjunto WorkSheets a cada paso de bucle.

Ejemplo 30.

Entrar valores para las celdas del rango A1:B10 de la hoja Activa.

Sub Ejemplo_30()

Dim R **As Range**

' Para cada celda del rango A1:B10 de la hoja activa

For Each R **in** ActiveSheet.Range("A1:B10")

R.Value = InputBox("Entrar valor para la celda " & R.Address, "Entrada de valores")

Next

End Sub

** Observe que se ha declarado una variable tipo Range, este tipo de datos, como puede imaginar y ha visto en el ejemplo sirve para guardar Rangos de una o más casillas, estas variables pueden luego utilizar todas las propiedades y métodos propios de los Objetos Range. Tenga en cuenta que la asignación de las variables que sirven para guardar o referenciar objetos (Range, WorkSheet, etc.) deben inicializarse muchas veces a través de la instrucción SET, esto se estudiará en otro capítulo.

Procedimientos y funciones.

Se define como procedimiento i/o función a un bloque de código que realiza alguna tarea. Hasta ahora, hemos construido los programas utilizando un único procedimiento, pero a medida que los programas (y los problemas) crecen se va haciendo necesaria la inclusión de más procedimientos. Podría fácilmente caer en la tentación de utilizar, como hasta ahora, un único procedimiento por programa pero se dará cuenta rápidamente de que este método no es nada práctico ya que grandes bloques de código implican mayor complicación del mismo, repetición de sentencias y lo que es más grave, mayores problemas de seguimiento a la hora de depurar errores, ampliar funcionalidades o incluir modificaciones.

La filosofía de utilizar procedimientos es la antigua fórmula del "divide y vencerás", es decir, con los procedimientos podremos tratar cada problema o tarea de forma más o menos aislada de forma que construiremos el programa paso a paso evitando tener que resolver o controlar múltiples cosas a la vez. Cada tarea la realizará un procedimiento, si esta tarea implica la ejecución de otras tareas, cada una se implementará y solucionará en su correspondiente procedimiento de manera que cada uno haga una cosa concreta. Así, los diferentes pasos que se deben ejecutar para que un programa haga algo, quedaran bien definidos cada uno en su correspondiente procedimiento, si el programa falla, fallará a partir de un procedimiento y de esta forma podremos localizar el error más rápidamente.

Los procedimientos son también un eficaz mecanismo para evitar la repetición de código en un mismo programa e incluso en diferentes programas. Suponemos que habrá intuido que hay muchas tareas que se repiten en casi todos los programas, veremos como los procedimientos que ejecutan estas tareas se pueden incluir en un módulo de forma que este sea exportable a otros programas y de esta manera ganar tiempo que, como dice el tópico, es precioso.

- **Definir un procedimiento.**

Ya lo hemos hecho unas cuantas veces, pero ahí va de nuevo.

```
Sub Nombre_Procedimiento
```

```
    Sentencias.
```

```
End Sub.
```

- **Llamar a un procedimiento.**

Para llamar un procedimiento desde otro se utiliza la instrucción **Call Nombre_Procedimiento**.

```
Sub P_Uno
```

```
    Sentencias.
```

```
    Call P_Dos
```

```
    Sentencias
```

```
End Sub
```

Las secuencias del procedimiento *P_Uno* se ejecutan hasta llegar a la línea **Call P_Dos**, entonces se salta al procedimiento *P_Dos*, se ejecutan todas las sentencias de este procedimiento y el programa continúa ejecutándose en el procedimiento *P_Uno* a partir de la sentencia que sigue a **Call P_Dos**.

Ejemplo 32.

Es el mismo programa que el visto en el ejemplo 29 pero el código que salta casilla hasta que se encuentra una vacía se implementa en un procedimiento llamado, *Saltar_Celdas_Llenas*. Observe que para entrar valores se ha sustituido Do While..Loop por Do.. Loop While.

Sub Ejemplo_32()

Dim Nombre **As String**

Dim Ciudad **As String**

Dim Edad **As Integer**

Dim fecha **As Date**

*' Llamada a la función Saltar_Celdas_Llenas, el programa salta aquí a ejecutar las
' instrucciones de este procedimiento y luego vuelve para continuar la ejecución a partir de la
' instrucción Do*

Call Saltar_Celdas_Llenas

Do

Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")

Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")

Edad = Val(InputBox("Entre la Edad : ", "Edad"))

Fecha=Cdate(InputBox("Entra la Fecha : ", "Fecha"))

With ActiveCell

.Value = Nombre

.Offset(0,1).Value = Ciudad

.Offset(0,2).Value = Edad

.Offset(0,3).value = fecha

End With

ActiveCell.Offset(1,0).Activate

Mas_datos = MsgBox("Otro registro ?", vbYesNo+vbQuestion,"Entrada de datos")

Loop While Mas_Datos = vbYes

End Sub

*' Función que salta celdas de una misma columna. Si rve para encontrar la primera celda vacía de la
' columna*

Sub Saltar_Celdad_Llenas()

WorkSheets("Hoja1").Activate

ActiveSheet.Range("A1").Activate

Do While not IsEmpty(ActiveCell)

ActiveCell.Offset(1,0).Activate

Loop

End Sub

- **Generalizar una función.**

Observe que para saltar un rango de casillas llenas sólo necesitará llamar a la función Saltar_Celdas_Llenas, pero, siempre y cuando este rango esté en una hoja llamada "Hoja1" y empiece en la casilla A1, el procedimiento es poco práctico ya que su ámbito de funcionamiento es limitado. En la siguiente sección modificaremos el procedimiento de manera que sirva para recorrer un rango que empiece en cualquier casilla de cualquier hoja.

Parámetros.

Los parámetros son el mecanismo por el cual un procedimiento puede pasarle valores a otro y de esta forma condicionar, moldear, etc. las acciones que ejecuta. El procedimiento llamado gana entonces en flexibilidad. La sintaxis de llamada de un procedimiento es la siguiente,

Call Procedimiento(Parámetro1, Parámetro2,..., ParámetroN)

Los parámetros pueden ser valores o variables.

La sintaxis para el procedimiento llamado es la siguiente,

Sub Procedimiento(Parámetro1 As Tipo, Parámetro2 As Tipo,..., Parámetro3 As Tipo)

Observe que aquí los parámetros son variables que recibirán los valores, evidentemente debe haber coincidencia de tipo. Por ejemplo, si el primer parámetro es una variable tipo Integer, el primer valor que se le debe pasar al procedimiento cuando se llama también ha de ser de tipo Integer (recuerde que puede ser un valor directamente o una variable).

Ejemplo 33.

El mismo programa que en el ejemplo 32 pero ahora la función Saltar_Celdas_Llenas tiene dos parámetros Hoja y Casilla_Inicial que reciben respectivamente la hoja donde está el rango a recorrer y la casilla inicial del rango.

Sub Ejemplo_33()

DimNombre **As String**

DimCiudad **As String**

DimEdad **As Integer**

Dimfecha **As Date**

*' Llamada a la función Saltar_Celdas_Llenas, observar que mediante dos parámetros se
' Al procedimiento en que hoja está el rango a saltar y en la casilla donde debe empezar.*

Call Saltar_Celdas_Llenas("Hoja1", "A1")

Do

Nombre = InputBox("Entre el Nombre (Return para Terminar) : ", "Nombre")

Ciudad = InputBox("Entre la Ciudad : ", "Ciudad")

Edad = Val(InputBox("Entre la Edad : ", "Edad"))

Fecha=Cdate(InputBox("Entre la Fecha : ", "Fecha"))

With ActiveCell

.Value = Nombre

.Offset(0,1).Value = Ciudad

.Offset(0,2).Value = Edad

.Offset(0,3).value = fecha

End With

ActiveCell.Offset(1,0).Activate

```
Mas_datos = MsgBox("Otro registro ?", vbYesNo+vbQuestion,"Entrada de datos")
```

```
Loop While Mas_Datos = vbYes
```

```
End Sub
```

```
,  
' Procedimiento Saltar_Celdas_Llenas.  
' Sirve para → Saltar celdas llenas de una columna hasta encontrar una vacía que se convierte en activa  
' Parámetros :  
' Hoja : Hoja donde está el rango a saltar.  
' Casilla_Inicial : Casilla Inicial de la columna  
Sub Saltar_Celdas_Llenas(Hoja As String, Casilla_Inicial As String)
```

```
WorkSheets(Hoja).Activate  
ActiveSheet.Range(Casilla_Inicial).Activate
```

```
Do While not IsEmpty(ActiveCell)  
ActiveCell.Offset(1,0).Activate
```

```
Loop
```

```
End Sub
```

Observe que ahora el procedimiento Saltar_Celdas_Llenas sirve para recorrer cualquier rango en cualquier hoja.

Observe que al procedimiento se le pasan dos valores directamente, recuerde, y esto es quizás lo más habitual, que también pueden pasarse variables, por ejemplo.

```
Sub Ejemplo_33
```

```
.  
'  
Dim Hoja As String  
Dim Casilla_Inicial As String
```

```
Hoja = InputBox("En que hoja está la base de datos : ", "Entrar Nombre de Hoja")  
Casilla_Inicial = InputBox("En que casilla comienza la base de datos", "Casilla Inicial")
```

```
' Observe que los parámetros son dos variables cuyo valor se ha entrado desde teclado en  
' las dos instrucciones InputBox anteriores.
```

```
Call Saltar_Celdas_Llenas(Hoja, Casilla_Inicial)
```

```
.
```

```
.
```

```
End Sub
```

Variables locales y variables Globales.

El ámbito de una variable declarada dentro de una función es la propia función, es decir no podrá utilizarse fuera de dicha función. Así, el siguiente programa que debería sumar las cinco filas siguientes a partir de la casilla activa y guardar el resultado en la sexta es incorrecto.

```
Sub Alguna_Cosa()  
.  
.  
    Call Sumar_Cinco_Siguientes  
    ActiveCell.Offset(6,0).Value = Suma  
.  
.  
End Sub  
  
Sub Sumar_Cinco_Siguientes()  
  
    Dim i As Integer  
    Dim Suma As Single  
  
    Suma=0  
    For i=1 To 5  
        Suma = Suma+ActiveCell.Offset(i,0).Value  
    Next i  
  
End Sub
```

Es incorrecto porque tanto la variable *i* como la variable *Suma* están declaradas dentro del procedimiento *Sumar_Cinco_Siguientes* consecuentemente, su ámbito de acción es este procedimiento. Por tanto, la instrucción *ActiveCell.Offset(6,0).Value = Suma* del procedimiento *Alguna_Cosa*, generaría un error (con *Option Explicit* activado) ya que la variable *Suma* no está declarado dentro de él. Si piensa en declarar la variable *Suma* dentro del procedimiento *Hacer_Algo*, no solucionará nada porque esta será local a dicho procedimiento, en este caso tendría dos variables llamadas *Suma* pero cada una de ellas local a su propio procedimiento o y consecuentemente con el ámbito de acción restringido a ellos.

Una solución, que a nosotros no nos gusta, sería declarar *suma* como variable global. Una variable global se declara fuera de todos los procedimientos y es reconocida por todos los procedimientos del módulo,

Option Explicit

' Suma es una variable global reconocida por todos los procedimientos del módulo.

```
Dim Suma As Single  
  
Sub Alguna_Cosa()  
.  
    Call Sumar_Cinco_Siguientes  
    ActiveCell.Offset(6,0).Value = Suma  
.  
.  
End Sub  
  
Sub Sumar_Cinco_Siguientes()  
  
    Dim i As Integer  
  
    Suma=0  
    For i=1 To 5  
        Suma = Suma+ActiveCell.Offset(i,0).Value  
    Next i  
  
End Sub
```

Las variables globales son perfectas en cierta ocasiones, para este caso sería mejor declarar *Suma* en la función *Hacer_Algo* y pasarla como parámetro al procedimiento *Sumar_Cinco_Siguientes*.

```
Sub Alguna_Cosa()  
  
    Dim Suma As Single  
    .  
    .  
    ' Llamada a la función Sumar_Cinco_Siguientes pasándole la variable Suma  
    Call Sumar_Cinco_Siguientes(Suma)  
    ActiveCell.Offset(6,0).Value = Suma  
    .  
    .  
End Sub
```

```
Sub Sumar_Cinco_Siguientes(S As Single)  
  
    Dim i As Integer  
  
    Suma=0  
    For i=1 To 5  
        S = S+ActiveCell.Offset(i,0).Value  
    Next i  
  
End Sub
```

Esto le funcionaría porque la variable parámetro *S* (y se le ha cambiado el nombre adrede) de *Sumar_Cinco_Siguientes* es la variable *Suma* declarada en *Hacer_Algo*. Funcionará porque en visual basic, a menos que se indique lo contrario, el paso de parámetros es por referencia, vea la siguiente sección.

Paso por referencia y paso por valor.

No entraremos en detalles sobre como funciona el paso de parámetros por valor y el paso de parámetros por referencia, sólo indicar que el paso por valor significa que la variable parámetro del procedimiento recibe el valor de la variable (o directamente el valor) de su parámetro correspondiente de la instrucción de llamada y en el paso por referencia, la variable parámetro del procedimiento es la misma que su parámetro correspondiente de la instrucción de llamada, es decir, la declarada en el procedimiento desde el que se hace la llamada. Por defecto, y siempre que en la instrucción de llamada se utilicen variables, las llamadas son por referencia. Si desea que el paso de parámetros sea por valor, debe anteponer a la variable parámetro la palabra reservada **ByVal**, por ejemplo,

```
Sub Saltar_Celdas_Llenas(ByVal Hoja As String, ByVal Casilla_Inicial As String)
```

Aunque lo elegante y efectivo por razones de memoria sería pasar siempre que sea posible por valor, es poco habitual que así se haga en visual basic, seguramente por comodidad. Como suponemos que hará como la mayoría, es decir, pasar por referencia, tenga cuidado con los (indeseables) efectos laterales. Copie y ejecute este programa y descubrirá que son los efectos laterales.

Ejemplo Efecto_Lateral.

Antes de copiar el programa, active una hoja en blanco y ponga valores del 1 al 15 distribuidos de la forma siguiente, en el rango A1:A5 valores del 1 al 5, en el rango B1:B5 valores del 6 al 10, en el rango C1:C5 valores del 11 al 15.

El siguiente programa debe recorrer cada una de tres las columnas de valores, sumarlos y poner el resultado en las filas 6 de cada columna. Entonces, según los valores que ha entrado en cada una de las columnas, cuando haya acabado la ejecución del programa debe haber los siguientes resultados, A6 = 15, B6=40, C6=65. Para llevar a cabo la suma de los valores de cada columna se llama a la función *Recorrer_Sumar* tres veces, una para cada columna, esta función recibe en el parámetro *F* el valor de la fila donde debe empezar a sumar, sobre el parámetro *C* el valor de la columna a sumar y sobre el parámetro *Q* la cantidad de filas que ha de recorrer.

El programa utiliza la propiedad **Cells** para referenciar las filas y columnas de los rangos. Observe atentamente los valores que irá cogiendo la variable *Fila* ya que esta será la que sufra el efecto lateral.

Sub Efecto_Lateral()

Dim Fila **As Integer**

Fila = 1

Call Recorrer_Sumar(Fila, 1,5) '*Columna A*

Call Recorrer_Sumar(Fila, 2,5) '*Columna B*

Call Recorrer_Sumar(Fila, 3,5) '*Columna C*

End Sub

Sub Recorrer_Sumar(**F As Integer**, **C As Integer**, **Q As Integer**)

Dim i **As Integer**

Dim Total **As Integer**

Total = 0

For i = 1 **To** Q

Total = Total + ActiveSheet.Cells(F, C).Value

F=F+1 '*OJO con esta asignación, recuerde que F es la variable Fila declarada en
' el procedimiento Efecto_Lateral*

Next i

ActiveSheet.Cells(F, C) = Total

End Sub

Cuando ejecute el programa se producirá la salida siguiente, en A6 habrá un 15, hasta aquí todo correcto, pero observe que en la segunda columna aparece un 0 en B12 y en la tercera columna aparece un 0 en C18, veamos que ha pasado. La primera vez que se llama la función, la variable *F* vale 1 ya que este es el valor que tiene su parámetro correspondiente (*Fila*) en la instrucción **Call**. Observe que *F* se va incrementando una unidad a cada paso de bucle **For**, RECUERDE que *F* es realmente la variable *Fila* declarada en el procedimiento *Efecto_Lateral*, por tanto cuando finaliza el procedimiento *Recorrer_Sumar* y vuelve el control al procedimiento *Efecto_Lateral* Fila vale 6, y este es el valor que se pasará a *Recorrer_Suma* la segunda vez que se llama, a partir de ahí todo irá mal ya que se empezará el recorrido de filas por la 6. Una de las soluciones a este problema para hacer que cada vez que se llame *Recorrer_Sumar* la variable *F* reciba el valor 1, es utilizar un paso por valor, es decir que F reciba el valor de Fila, no que sea la variable Fila, observe que entonces, si *F* no es la variable *Fila*, cuando incremente *F* no se incrementará *Fila*, esta siempre conservará el valor 1. Para hacer que *F* sea un parámetro por valor, simplemente ponga la palabra **ByVal** antes de *F* en la declaración del procedimiento. Vuelva a ejecutar el programa, verá como ahora funciona correctamente.

Insistimos de nuevo en que tenga cuidado con estas cosas. Al menos ahora ya está sobre aviso, cuando un programa no haga lo que ha previsto una de las cosas que deberá repasar es el paso de parámetros a los procedimientos.

** Para acabar, observe que en muchas ocasiones le hemos indicado que en el paso por referencia la variable del procedimiento llamado es la variable declarada en el procedimiento que llama. En este último ejemplo, le hemos dicho que *F* era la variable *Fila*, pues bien, esto no es cierto *Fila* es una variable y *F* es otra variable, ahora es lógico que se pregunte por qué entonces *F* actúa como si fuera *Fila*, este es un tema que no entra en el ámbito de este manual, si alguna vez programa en C y llega al tema de los punteros entenderá que es lo que sucede realmente en el paso por parámetro y en el paso por valor. Si ya conoce los punteros de C o Pascal entonces ya habrá intuido que el paso por valor en nuestro ejemplo equivaldría a,

```
Recorrer_Fila(F, C, Q);
```

```
void Recorrer_Fila(int F, int C, int Q)
```

Y un paso por referencia a

```
Recorrer_Fila(&F, C, Q);
```

```
Void Recorrer_Fila(int *F, int C, int Q)
```

- **Funciones.**

Una función es lo mismo que un procedimiento con la salvedad que este devuelve un valor al procedimiento o función que lo llama. Vea el siguiente ejemplo, es una función muy sencilla ya que simplemente suma dos números y devuelve el resultado.

Ejemplo 34.

Función que devuelve la suma de dos valores que se le pasan como parámetros. Observe las diferentes formas en como se llama la función.

Sub Ejemplo_34()

Dim x **As Integer**

Dim n1 **As Integer**, n2 **As Integer**

X = Suma(5, 5)

n1= Val (InputBox("Entrar un número : ", "Entrada"))

n2= Val (InputBox("Entrar otro número : ", "Entrada"))

X= suma(n1,n2)

ActiveCell.Value = Suma(ActiveSheet.Range("A1").Value , ActiveSheet.Range("A2").Value)

X = Suma(5, 4) + Suma (n1, n2)

End Sub

Function Suma(V1 **As Integer**, V2 **As Integer**) **As Integer**

Dim Total **As Integer**

Total = V1 + V2

Suma = Total

End Function

Observe la sintaxis de la cabecera de función,

Function Suma(V1 **As Integer**, V2 **As Integer**) **As Integer**

La estructura general seria,

Function Nombre_Funcion(par1 **As Tipo**, par2 **As Tipo**....., parN **As Tipo**) **As Tipo**.

La sintaxis es similar a la cabecera de un procedimiento, sólo que una función tiene tipo, esto tiene su lógica, ya que una función devuelve un valor, ese valor será de un tipo determinado. Así, en nuestro ejemplo de **Function Suma**, esta función es de tipo **Integer**, o dicho de otra manera, la función ejecuta sus sentencias y devuelve un valor hacia el procedimiento o la función que la llamó, el valor devuelto se establece igualando el nombre de la función a algo,

Nombre_Función =

En el ejemplo de **Function Suma**,

Suma = Total

Observe también la sintaxis de la llamada a la función, en el ejemplo hemos utilizado unas cuantas formas de llamarla, lo que debe tener siempre presente es que en cualquier expresión aritmética o de cálculo, el ordenador realiza un mínimo de dos operaciones, una de cálculo y otra de asignación. Por ejemplo,

A= B+C

El ordenador primero calcula el resultado de sumar B+C luego asigna ese resultado a la variable A. En cualquier llamada a una función, cojamos por caso,

X= suma(n1,n2)

Primero se ejecutan todas las sentencias de la función Suma, luego se asigna el cálculo de la función a la variable X. De otro vistazo a la función de ejemplo y vea lo que realiza cada sentencia en la que se llama a la función *Suma*.

Veamos a continuación unos cuantos ejemplos de funciones. Antes recordarle que todo lo referente a parámetros por valor y referencia, variables locales y globales, etc. que estudiamos en los procedimientos es lo mismo para las funciones.

Ejemplo 35.

Función que devuelve la dirección de la primera celda vacía de un rango. La función es de tipo **String** ya que devuelve la casilla en la forma "FilaColumna", por ejemplo "A10". Utilizaremos la propiedad **Address** del objeto range, esta propiedad devuelve un string que contiene la referencia "FilaColumna" de una casilla o rango de casillas. En el caso de un rango devuelve,

"FilaColumna_Inicial:FilaColumna_Final", por ejemplo "A1:C10"

Sub Ejemplo_35()

Dim Casilla **As String**

Casilla = Casilla_Vacia("A1")

.....

End Sub

```
' Función Casilla_Vacia de Tipo String
' Sirve para → Recorrer las filas de una columna hasta encontrar una vacía.
' Parámetros :
'     Casilla_Inicio : Casilla donde debe empezar a buscar.
' Devuelve → Un string que contiene la referencia de la primera casilla
Function Casilla_Vacia(Casilla_Inicio As String) As String
    ActiveSheet.Range(Casilla_Inicio).Activate
```

```
Do While Not IsEmpty(ActiveCell)
    ActiveCell.Offset(1, 0).Activate
Loop

    Casilla_Vacia = ActiveCell.Address
End Function
```

Ejemplo 36.

Similar al anterior. Es la típica búsqueda secuencial de un valor en un rango de casillas, en esta función solo se avanzará a través de una fila. La función devuelve la dirección (address) de la casilla donde está el valor buscado, en caso que el valor no esté en el rango de filas, devuelve una cadena vacía ("").

```
Sub Ejemplo_36()
    Dim Casilla As String
    Casilla = Buscar_Valor("A1", 25)
```

```
    ' Si valor no encontrado
```

```
    If Casilla = "" Then
```

```
        .....
```

```
    Else 'Valor encontrado
```

```
        ....
```

```
    End if
```

```
End Sub
```

```
' Función Buscar de Tipo String
' Sirve para → Recorrer las filas de una columna hasta encontrar el valor buscado o una de vacía.
' Parámetros :
'     Casilla_Inicial : Casilla donde debe empezar a buscar.
'     Valor_Buscado : Valor que se debe encontrar
' Devuelve → Un string que contiene la referencia de la casilla donde se ha encontrado el valor.
'     También puede devolver "" en caso que el valor buscado no esté.
```

```
Function Buscar(Casilla_Inicial As String, Valor_Buscado As Integer) As String
```

```
    ActiveSheet.Range(Casilla_Inicial).Activate
```

```
    ' Mientras casilla no vacía Y valor de casilla diferente al buscado
```

```
    Do While Not IsEmpty(ActiveCell) And ActiveCell.Value <> Valor_Buscado
        ActiveCell.Offset(1, 0).Activate
```

```
    Loop
```

```
    ' Si la casilla donde se ha detenido la búsqueda NO ESTÁ VACÍA es que se ha encontrado
    ' el valor.
```

```
    If Not IsEmpty(ActiveCell) Then
```

```
        Buscar = ActiveCell.Address ' Devolver la casilla donde se ha encontrado el valor
```

```
    Else ' La casilla está vacía, NO se ha encontrado el valor buscado
```

```
        Buscar="" ' Devolver una cadena vacía
```

```
    End if
```

```
End Function
```

Ejemplo 36.

Similar al anterior. Búsqueda secuencial de un valor en un rango de casillas, en esta función se avanzará a través de filas y columnas. La función devuelve la dirección (address) de la casilla donde está el valor buscado, en caso que el valor no esté en el rango, devuelve una cadena vacía ("").

Sub Ejemplo_36()

Dim Casilla **As** String

Casilla = Buscar_Valor("A1", 25)

If Casilla = "" **Then**

....

Else

....

End if

End Sub

Function Buscar(Casilla_Inicial **As** String, Valor_Buscado **As** Integer) **As** String

Dim Incremento_Columna **As** Integer

Dim Continuar **As** Boolean

ActiveSheet.Range(Casilla_Inicial).Activate

Continuar = True

Do While Continuar

Incremento_Columna = 0

' Buscar el valor por las columnas hasta encontrarlo o encontrar una celda vacía.

Do While Not IsEmpty(ActiveCell.Offset(0, Incremento_Columna) **And**

ActiveCell.Offset(0, Incremento_Columna.Value <> Valor_Buscado

' Siguiente columna

Incremento_Columna = Incremento_Columna + 1

Loop

' Si no está vacía la casilla entonces parar la búsqueda, se ha encontrado el valor

If Not IsEmpty(ActiveCell.Offset(0, Incremento_Columna)) **Then**

Continuar=False

Else *' La casilla está vacía, no se ha encontrado el valor*

ActiveCell.Offset(1, 0).Activate *' Saltar a una nueva fila*

If IsEmpty(ActiveCell) **Then** *' Si la casilla de la nueva fila está vacía*

Continuar=False *' Parar la búsqueda, no hay más casilla a recorrer*

End if

End if

Loop

' Si la casilla donde se ha detenido la búsqueda NO ESTÁ VACÍA es que se ha encontrado el valor.

If Not IsEmpty(ActiveCell) **Then**

Buscar = ActiveCell(0, Incremento_Columna).Address *' Devolver la casilla donde se ha encontrado el valor*

Else *' La casilla está vacía, NO se ha encontrado el valor buscado*

Buscar="" *' Devolver una cadena vacía*

End if

End Function

La cláusula Private.

Puede anteponer la cláusula private a todos los procedimientos y funciones que sean llamados sólo desde el mismo módulo, es una forma de ahorrar memoria y hacer que el programa corra un poco más rápido. Si necesita llamar un procedimiento o función desde otro módulo, nunca debe precederlo por la cláusula private, recuerde que esta cláusula restringe el ámbito de utilización de un procedimiento a su propio módulo. Supongamos el ejemplo siguiente.

' Módulo 1

```
Sub General
....
End Sub

Private Sub Privado
....
End Sub
```

' Módulo 2

```
Sub Procedimiento_de_modulo2

    ' Esto es correcto. Llama al procedimiento General definido en Módulo1
    Call General

    ' Esto no es correcto. Llama al procedimiento Privado definido en Módulo 1, este
    ' procedimiento va precedido pro la cláusula Private, por tanto sólo puede ser llamado
    ' desde procedimientos de su propio módulo
    Call Privado

End Sub
```

Vamos a ver a continuación tres ejemplos más sobre funciones. Es importante que los cree en un libro de trabajo nuevo y los ponga en un mismo módulo, al final del capítulo utilizaremos las opciones de exportar e importar módulos de procedimientos y funciones. En todos los ejemplos verá el procedimiento `Procedimiento_Llamador`, es para mostrar de que forma se debe llamar al procedimiento o función. Los procedimientos implementados son, por llamarlo de alguna manera, de tipo general, es decir, son procedimientos que podrá utilizar en muchas aplicaciones.

Ejemplo 37.

Procedimiento que abre un cuadro MsgBox y muestra el texto que se le paso como parámetro.

Sub `Procedimiento_Llamador()`

```
.  
. Call mAviso("Esto es el mensaje de aviso", "Esto es el Título")  
.  
.
```

End Sub

' Procedimiento mAviso

' Función → Mostrar el cuadro de función MsgBox, con el icono información y el botón OK (Aceptar).

' Se utiliza para mostrar avisos.

' Parámetros:

' Texto = Texto que muestra el cuadro

' Titulo = Título del cuadro

,

Sub `mAviso(Texto As String, Titulo As String)`

```
MsgBox Prompt:=Texto, Buttons:=vbOKOnly + vbInformation, Title:=Titulo
```

End Sub

Ejemplo 38.

Función tipo range que devuelve un rango. Observe como la función se iguala a una variable tipo Range, recuerde que con esta variable podrá acceder a todas las propiedades e invocar todos los métodos propios de los objetos Range. En este ejemplo en concreto se utilizan las variables para Copiar un grupo de celdas de un rango hacia otro, se utilizan los métodos `Copy` y `Paste` del objeto Range.

Sub `Procedimiento_Llamador()`

```
Dim Rango_Origen As Range
```

```
Dim Rango_Destino As Range
```

```
Set Rango_Origen=Coger_Rango(A1,5,5)
```

```
Rango_Origen.Copy
```

```
Set Rango_Destino=Coger_Rango(G1,5,5)
```

```
Rango_Destino.Paste PasteSpecial:=xlPasteAll
```

End Sub

' Función que devuelve un rango a una variable de este tipo

' Parámetros

' Casilla = casilla inicial del rango

' Filas = número de filas

' Columnas = número de columnas del rango

Function Coger_Rango(Casilla **As String**, Filas **As Integer**, Columnas **As Integer**) **As Range**

Dim Casilla_Final **As String**

ActiveSheet.Range(Casilla).Activate

ActiveCell.Cells(Filas, Columnas).Activate

Casilla_Final = ActiveCell.Address

ActiveSheet.Range(Casilla & ":" & Casilla_Final).Select

Set Coger_Rango = ActiveSheet.Range(Casilla & ":" & Casilla_Final)

End Function

Ejemplo 39.

Función para comprobar el tipo de datos. Es una función de comprobación que se puede utilizar para validar los datos que se entran desde un cuadro InputBox o desde los cuadros de texto de formularios. La función es de tipo Booleano, devuelve True (cierto) o False en función de si el dato pasado es correcto. En esta función se evalúan sólo datos numéricos y datos tipo Fecha, puede ampliarla para que se comprueben más tipos.

Sub Procedimiento_Llamador()

Dim Cantidad **As Integer**

Dim Fecha **As Date**

Dim Datos **As String**

.

.

Datos = InputBox("Entrar una Cantidad : ", "Entrar")

If Not Comprobar_Tipo(Datos,"N") **Then**

mAviso("Los datos introducido no son numéricos", "Error")

Else

Cantidad = Val(Datos)

.

.

End If

.

.

Datos=InputBox("Entrar Fecha","Entrar")

If Not Comprobar_Tipo(Datos,"F") **Then**

mAviso("Los fecha introducida no es correcta", "Error")

Else

Fecha = Val(Datos)

.

.

End If

.

.

End Sub

```
' Función que evalúa si el tipo de datos que se le pasan son correctos o no. Si son correctos devuelve  
' TRUE , en caso contrario devuelve FALSE  
' Parámetros  
' Valor =valor que se debe comprobar, de tipo String  
' Tipo = tipo a comprobar, "N"--> Numérico, "F", tipo fecha
```

Function Comprobar_Tipo(Valor **As String**, Tipo **As String**) **As Boolean**

Dim Valido **As Boolean**

Valido = **True**

Select Case Tipo

' Comprueba si es un valor numérico válido

Case "N"

If Not IsNumeric(Valor) **Then**

Valido = False

End If

' Comprueba si es un valor fecha válido

Case "F"

If Not IsDate(Valor) **Then**

Valido = False

End If

End Select

Comprobar_Tipo = Valido

End Function

• Importar y Exportar módulos.

Los últimos tres ejemplos que hemos visto, como le hemos indicado, son procedimientos que pueden ser utilizados en multitud de ocasiones o situaciones, sería interesante tenerlos disponibles en cualquiera de las hojas que confeccionemos. Podría pensar en utilizar las opciones de copiar y pegar para pasar procedimientos de una hoja a otra, es un método totalmente válido y efectivo, pero le proponemos a continuación otro método más "profesional", por llamarlo de otra manera, e igual de efectivo. Este método consiste en guardar los procedimientos de un módulo en un archivo aparte, es decir, independiente de cualquier hoja de cálculo, luego, cuando en una nueva hoja necesite estas funciones, solo deberá importar este archivo para incorporarlo.

Exportar un módulo. Guardar un módulo en un archivo.

Como ejemplo, abra la hoja donde puso los tres últimos procedimientos.

1. Pase al editor de visual basic y active el módulo a exportar.
2. Active opción de la barra de menús **Archivo/ Exportar archivo**. Aparece un cuadro de diálogo.
3. En cuadro de edición **Nombre de Archivo**, teclee el nombre para el archivo donde se guardará el módulo, por ejemplo "General.Bas", observe que .BAS es la extensión de estos archivos.
4. Pulse sobre el botón **Guardar**.

Importar un módulo.

Si está siguiendo el ejemplo, cierre todos los archivos de Excel y abra uno nuevo.

1. Active el editor Visual Basic.
2. Active opción de la barra de menús **Archivo/ Importar Archivo**. Aparece un cuadro de diálogo.
3. Seleccione en la lista **Buscar en:** la carpeta donde tiene ubicado el archivo a importar (la carpeta donde está General.Bas si está siguiendo el ejemplo).
4. Una vez localizada la carpeta, seleccione el archivo a importar (General.Bas en el ejemplo) y pulse sobre **Abrir**.

Observe como en la ventana de proyecto se ha incorporado un nuevo módulo que contiene todos los procedimientos y funciones del archivo importado.

Terminamos aquí el tema de procedimientos y funciones, déjenos insistir de nuevo en que es muy importante que construya sus programas utilizando todas las ventajas que le ofrece la programación modular. Como último consejo, agrupe todas las funciones que usted considere de utilización general en uno o dos módulos y luego utilice las opciones de importación y exportación para incorporarlos a sus programas.

La grabadora de macros.

Microsoft Excel lleva incluida una utilidad que sirve para registrar acciones que se llevan a cabo en un libro de trabajo y registrarlas en forma de macro. Podemos aprovechar esta utilidad para generar código engorroso por su sintaxis un tanto complicada de recordar, además de ahorrar tiempo. Casi siempre después deberemos modificarlo para adaptarlo a nuestros programas, sin embargo eso resultará sumamente sencillo. Vea el ejemplo siguiente que sirve para poner bordes al rango de celdas de A1 a G6, observe los comentarios para saber que bordes se ponen y dónde se ponen.

Ejemplo 1.

Poner bordes al rango que va de A1 a G6.

Sub Poner_Bordes()

```
'Seleccionar el rango A1:G6
Range("A1:G6").Select

'No hay borde diagonal hacia abajo
Selection.Borders(xlDiagonalDown).LineStyle = xlNone

'No hay borde diagonal hacia arriba
Selection.Borders(xlDiagonalUp).LineStyle = xlNone

' Borde izquierdo de la selección
With Selection.Borders(xlEdgeLeft)
    .LineStyle = xlContinuous           'Estilo de línea continuo
    .Weight = xlMedium                 'Ancho de línea Medio
    .ColorIndex = xlAutomatic          'Color de línea automático (negro)
End With

' Borde superior de la selección
With Selection.Borders(xlEdgeTop)
    .LineStyle = xlContinuous
    .Weight = xlMedium
    .ColorIndex = xlAutomatic
End With

' Borde inferior de la selección
With Selection.Borders(xlEdgeBottom)
    .LineStyle = xlContinuous
    .Weight = xlMedium
    .ColorIndex = xlAutomatic
End With

' Borde derecho de la selección
With Selection.Borders(xlEdgeRight)
    .LineStyle = xlContinuous
    .Weight = xlMedium
    .ColorIndex = xlAutomatic
End With

' Bordes verticales interiores de la selección
With Selection.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .Weight = xlThin                   ' Ancho Simple.
    .ColorIndex = xlAutomatic
End With
```

'No hay bordes horizontales interiores en la selección
Selection.Borders(xlInsideHorizontal).LineStyle = xlNone

'Seleccionar rango A1:G1
Range("A1:G1").Select

'No hay borde diagonal hacia arriba
Selection.Borders(xlDiagonalDown).LineStyle = xlNone

'No hay borde diagonal hacia arriba
Selection.Borders(xlDiagonalUp).LineStyle = xlNone

'Borde izquierdo de la selección
With Selection.Borders(xlEdgeLeft)
 .LineStyle = xlContinuous
 .Weight = xlMedium
 .ColorIndex = xlAutomatic

End With

'Borde superior de la selección
With Selection.Borders(xlEdgeTop)
 .LineStyle = xlContinuous
 .Weight = xlMedium
 .ColorIndex = xlAutomatic

End With

'Borde inferior de la selección
With Selection.Borders(xlEdgeBottom)
 .LineStyle = xlDouble
 .Weight = xlThick
 .ColorIndex = xlAutomatic

End With

'Doble línea

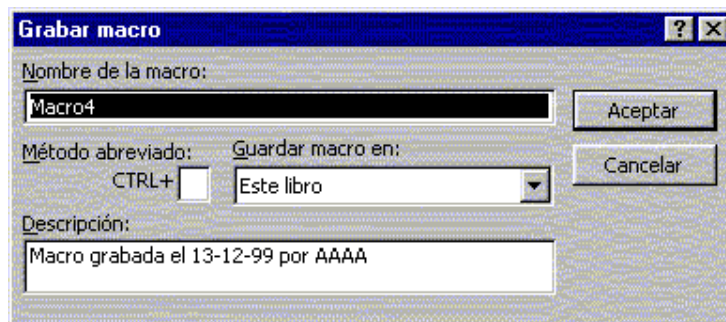
'Borde derecho de la selección
With Selection.Borders(xlEdgeRight)
 .LineStyle = xlContinuous
 .Weight = xlMedium
 .ColorIndex = xlAutomatic


End With

End Sub

Suponemos que el procedimiento anterior le parecerá abrumador, no se preocupe, le explicamos a continuación como lo hemos hecho y verá que lo único que debe hacer son los pasos sobre la hoja de cálculo, el grabador de macros se ocupa del resto.

1. Active la grabadora de macros. **Herramientas/ Macro/ Gravar nueva macro**. Aparece la ventana siguiente.



2. En **Nombre de Macro**, ponga *Poner Líneas*. En **Guardar Macro en**, deje la opción **Libro Activo**. En **Descripción**, ponga, *macro para bordes a un rango de celdas*.
3. Pulse sobre el botón **Aceptar**.
4. Ejecute los pasos siguientes. Son para poner los bordes
 - 4.1 Seleccione el rango A1:G6.
 - 4.2 Active Opción **Formato/ Celdas**. Seleccione ficha **Bordes**.
 - 4.3 Ponga línea alrededor de la selección
 - 4.3.1 En cuadro **Estilos**, seleccione la penúltima o antepenúltima línea.
 - 4.3.2 Pulse sobre el botón que representa un cuadrado.
 - 4.4 Ponga las líneas verticales.
 - 4.4.1 Seleccione del cuadro estilo la última línea de la primera columna.
 - 4.4.2 Pulse sobre el botón que representa las líneas interiores de la selección (el situado en el centro de la línea inferior de botones).
4. Pulse sobre el botón **Aceptar**.
5. Seleccione el rango A1:G1
6. Active de nuevo la opción **Formato/Celdas** y seleccione la ficha **Bordes**.
7. Ponga línea inferior doble.
 - 7.1 Seleccione del cuadro Estilo la última línea de la segunda columna.
 - 7.2 Pulse sobre el botón que representa la línea inferior de la selección.
8. Pulse sobre el botón **Aceptar**.
9. Detenga la grabadora de Macros pulsando sobre el botón  o bien **Herramientas/ Macros / Detener Grabación**.

Y Ya está. Ahora abra el Editor de Visual Basic y compruebe como la grabadora ha hecho todo el trabajo pro usted. Ahora ya sólo nos queda modificar la macro de forma que este formato de bordes sea aplicable a cualquier rango. Vea el ejemplo siguiente.

Ejemplo2.

En este ejemplo modificaremos la macro de manera que sirva para dar ese formato a cualquier rango de la hoja. Observe en este ejemplo simplemente pasamos como parámetro el rango a formatear de manera que la cabecera del procedimiento quedaria.

' Procedimiento Poner_Bordes.

' Procedimiento que pone bordes a un rango. Concretamente lo encierra en un cuadrado, le pone líneas verticales y pone una doble línea inferior en la primera fila.

' Parámetros:

' Nombre_Hoja: Nombre de la hoja donde está el rango.

' Rango_Total : Rango al que se le aplica el formato.

' Rango_Primer_Fila : Rango de la primera fila al que se debe aplicar doble línea inferior.

Sub Poner_Bordes (Nombre_Hoja **As String**, Rango_Total **As String**, Rango_Primer_Fila **As String**)

' Seleccionamos la hoja en la que se encuentra en rango
Worksheets(Nombre_Hoja).Activate

' Seleccionamos en rango
ActiveSheet.Range(Rango_Total).Select

' Hacemos cuadro y líneas verticales.

.....
.....

' Selección de la primera fila
ActiveSheet.Range(Primera_Fila).Select

' Hacemos línea inferior doble

.....

End Sub

Ejemplo 3.

Refinamiento de la macro del ejemplo anterior. En este ejemplo todavía perfeccionaremos más la macro del ejercicio anterior. Sobre todo en lo referente a los parámetros. Básicamente cambiaremos el tipo de parámetros, así en lugar del nombre de la hoja pasaremos el número y en lugar de los parámetros Rango_Total y Rango_Primer_Fila, simplemente pasaremos el rango de la casilla inicial, la macro se encargará de seleccionar todas las casillas adyacentes y de buscar la primera fila. En esta macro además se han incluido funcionalidades como borrar los formatos antes de aplicar las líneas, ajustar el ancho de las columnas, etc. Lea los comentarios para ver que hace cada sección de la macro.

Observe la propiedad **CurrentRegion** del objeto **Range**, esta propiedad devuelve el rango de las casillas llenas adyacentes a una dada. Por ejemplo imagine una hoja con el rango A1:B10 lleno de valores, la instrucción

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

Seleccionaría el rango correspondiente a A1:B10.

' Función Poner_Líneas_Selección.

' Esta función sirve para poner bordes a un rango. Pone cuadro al rango, separa las columnas con una línea y pone doble línea inferior en la ' primera fila.

'

' Parámetros:

' Num_Hoja = Número de hoja donde está el rango a formatear.

' Casilla = Casilla inicial (superior izquierda) del rango a formatear

Sub Poner_Líneas_Seleccion(Numero_Hoja **As Integer**, Casilla **As String**)

Dim x **As Integer**

Dim Casilla_Inicial **As String**

Dim Casilla_Final **As String**

Worksheets(Numero_Hoja).Activate

ActiveSheet.Range(Casilla).Activate

' Seleccionar el rango de celdas con valores adyacentes a la activa

ActiveCell.CurrentRegion.Select

' Borrar los bordes actuales de la selección actuales

With Selection

.Borders(xlDiagonalDown).LineStyle = xlNone

.Borders(xlDiagonalUp).LineStyle = xlNone

.Borders(xlEdgeLeft).LineStyle = xlNone

.Borders(xlEdgeTop).LineStyle = xlNone

.Borders(xlEdgeBottom).LineStyle = xlNone

.Borders(xlEdgeRight).LineStyle = xlNone

.Borders(xlInsideVertical).LineStyle = xlNone

.Borders(xlInsideHorizontal).LineStyle = xlNone

End With

' Líneas del recuadro

With Selection.Borders(xlEdgeLeft)

.LineStyle = xlContinuous

.Weight = xlMedium

.ColorIndex = xlAutomatic

End With

```
With Selection.Borders(xlEdgeTop)
.LineStyle = xlContinuous
.Weight = xlMedium
.ColorIndex = xlAutomatic
```

```
End With
```

```
With Selection.Borders(xlEdgeBottom)
.LineStyle = xlContinuous
.Weight = xlMedium
.ColorIndex = xlAutomatic
```

```
End With
```

```
With Selection.Borders(xlEdgeRight)
.LineStyle = xlContinuous
.Weight = xlMedium
.ColorIndex = xlAutomatic
```

```
End With
```

```
' Líneas verticales interiores, si en el
' rango hay más de una columna.
```

```
If Selection.Columns.Count > 1 Then
```

```
    With Selection.Borders(xlInsideVertical)
    .LineStyle = xlContinuous
    .Weight = xlThin
    .ColorIndex = xlAutomatic
```

```
    End With
```

```
End If
```

```
' Ajustar ancho de columnas
Selection.Columns.AutoFit
```

```
' Línea doble inferior de primera fila
' Para este proceso se selecciona la casilla inicial pasada a la macro, luego se busca
' la última casilla con datos. Se construye un rango combinando las direcciones de ambas
' casillas y se utiliza el objeto Range junto con el método Select para hacer la selección
'
```

```
    RANGE(Inicial:Final).Select
```

```
' Seleccionar primera casilla
ActiveSheet.Range(Casilla).Select
```

```
' Buscar primera casilla vacía de misma fila recorriendo las columnas
```

```
x = 0
```

```
Do While Not IsEmpty(ActiveCell.Offset(0, x))
```

```
    x = x + 1
```

```
Loop
```

```
' Recoger las direcciones de la casilla inicial Y la casilla final, es decir las referencias
```

```
' FilaColumna (A1, A2,...)
```

```
Casilla_Inicial = ActiveCell.Address
```

```
Casilla_Final = ActiveCell.Offset(0, x - 1).Address
```

```
' Seleccionar el rango de la fila. Observar la concatenación de las cadenas de  
' Casilla_Inicial y casilla_final que representan las direcciones del rango a seleccionar  
ActiveSheet.Range(Casilla_Inicial & ":" & Casilla_Final).Select
```

```
' Poner doble línea  
With Selection.Borders(xlEdgeBottom)  
    .LineStyle = xlDouble  
    .Weight = xlThick  
    .ColorIndex = xlAutomatic  
End With
```

```
ActiveSheet.Range(Casilla).Activate
```

End Sub

Bueno, este segundo ejemplo, un poco más sofisticado es sólo para que vea hasta que punto se puede refinar una macro. Primero construimos una macro con la grabadora que ponía formato a un rango de casillas que siempre era el mismo, después modificamos esta macro de manera que el rango de casillas que pudiera formatear fuera cualquiera, como tercer paso, además de conseguir lo mismo que la segunda versión pero con menos parámetros le hemos añadido nuevas funcionalidades como ajustar el ancho de las celdas, borrar los formatos previos, etc. Y esta macro todavía podría refinarse más, poner color de fondo a la primera fila, poner color a los datos numéricos, etc. Lo que intentamos decirle es que aproveche al máximo la utilidad de la grabadora de macros, tenga pero en cuenta que casi siempre tendrá que añadir código usted mismo si quiera ampliar la funcionalidad de la macro, sobre todo si quiere aplicarle cierta generalidad.

Vea el siguiente ejemplo, sirve para representar gráficamente un rango de valores. La macro se ha hecho de forma similar a la anterior, es decir, una vez activada la grabadora de macros se han seguido todos los pasos necesarios para diseñar el gráfico, luego se han cambiado las referencias a hojas y rangos por variables que se colocan como parámetros del procedimiento para así dotarle de generalidad.

' Procedimiento Grafico.

' Procedimiento que representa gráficamente los valores de un rango.

' Parámetros:

' Hoja_Datos : Hoja donde está el rango de valores.

' Rango_Datos : Rango de valores a representar gráficamente.

' Titulo : Título para el gráfico.

' Eje_X : Título para el eje X

' Eje_Y : Título para el eje Y

Sub Grafico(Hoja_Datos **As String**, Rango_Datos **As String**,Titulo **As String**, Eje_X **As String**, Eje_Y **As String**)

' Añadir el gráfico

Charts.Add

' Tipo de gráfico-> xlColumnClustered (Columnas agrupadas)

ActiveChart.ChartType = xlColumnClustered

' Definir el origen de datos y representar las series(PlotBy) por filas (xlRows)

ActiveChart.SetSourceData Source:=Sheets(Hoja_Datos).Range(Rango_Datos), PlotBy:= _
xlRows

' El gráfico debe ponerse en una hoja nueva

ActiveChart.Location Where:=xlLocationAsNewSheet

With ActiveChart

' Tiene título

.HasTitle = True

' Poner título

.ChartTitle.Characters.Text = "Ventas de Frutas"

' Tiene título para el eje X

.Axes(xlCategory, xlPrimary).HasTitle = True

' Título para el eje X

.Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "Meses"

' Tiene título para el eje Y principal

.Axes(xlValue, xlPrimary).HasTitle = True

' Título para el eje Y principal

.Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "Cantidades"

End With

' Poner líneas de división para el eje de categorías X (horizontales)

With ActiveChart.Axes(xlCategory)

' Poner Lineas de división primarias

.HasMajorGridlines = True

' No poner líneas de división secundarias

.HasMinorGridlines = False

End With


```
' Poner líneas de división para el eje Y (verticales)
With ActiveChart.Axes(xlValue)
    .HasMajorGridlines = True
    .HasMinorGridlines = False
End With

' Tiene leyenda
ActiveChart.HasLegend = True

' Seleccionar leyenda
ActiveChart.Legend.Select

' Situar la leyenda en la parte inferior
Selection.Position = xlBottom

ActiveChart.ApplyDataLabels Type:=xlDataLabelsShowNone, LegendKey:=False
End Sub
```

Para finalizar, seguro que cuando utilice la grabadora aparecerán muchas propiedades y métodos que desconoce, sólo debe hacer clic sobre estas propiedades o métodos y pulsar F1, automáticamente se activará la ayuda sobre esa propiedad o método concreto.

- **Insertar funciones de Microsoft Excel desde Visual Basic.**

Copie el siguiente procedimiento y ejecútelo. Es un procedimiento que sencillamente va pidiendo números y los va colocando en las celdas de la columna A a partir de A1, al final coloca la función =SUMA para sumar los valores introducidos y la función =PROMEDIO para hacer el promedio de los mismos valores.

```
Sub Sumar()

    Dim Valor As Integer
    Dim Casilla_Inicial As String
    Dim Casilla_Final As String

    ' Hacer activa la casilla A1 de la hoja activa
    ActiveSheet.Range("A1").Activate

    Do

        ' Entrar un valor y convertirlo a numérico
        Valor = Val(InputBox("Entrar un valor", "Entrada"))

        ' Si el valor es distinto de 0
        If Valor <> 0 Then
            ' Guardar el valor en la casilla activa
            ActiveCell.Value = Valor
            ' Hacer activa la casilla de la fila siguiente
            ActiveCell.Offset(1, 0).Activate
        End If
    Loop Until Valor = 0

End Sub
```

'Establecer la casilla inicial del rango a sumar

Casilla_Inicial = "A1"

'Establecer la casilla final del rango a sumar.

'Coger la dirección de la casilla activa, la última

Casilla_Final = ActiveCell.Address

ActiveCell.Offset(1, 0).Activate

'Poner en la casilla activa la función SUMA

ActiveCell.Formula = "=Suma(" & Casilla_Inicial & ":" & Casilla_Final & ")"

ActiveCell.Offset(1, 0).Activate

'Poner en la casilla activa la función promedio

ActiveCell.Formula = "=Promedio(" & Casilla_Inicial & ":" & Casilla_Final & ")"

End Sub

Una vez haya ejecutado la macro, observe que en las celdas donde se han colocado respectivamente las funciones =SUMA, =PROMEDIO aparece ¿NOMBRE? (es posible que aparezca ####, en es caso amplíe la columna), esto significa que Excel no reconoce el nombre de la función, que no existe. Sin embargo, estas funciones si existen y funcionan perfectamente cuando se teclean directamente sobre la hoja de cálculo, se preguntará el porqué cuando se colocan desde una macro no funcionan. Pues resulta que para que cualquier función de Excel insertada desde una macro NO de error debe ponerse con su nombre en Inglés, la traducción se hace luego de forma automática. Es decir en la macro debe ponerla en inglés y luego cuando esta se inserte en la hoja aparecerá con su nomenclatura en el idioma que corresponda.

Modifique el procedimiento del ejemplo y en lugar de poner

```
ActiveCell.Formula = "=Suma(" & Casilla_Inicial & ":" & Casilla_Final & ")"
```

Ponga

```
ActiveCell.Formula = "=Sum(" & Casilla_Inicial & ":" & Casilla_Final & ")"
```

Y ahora, en lugar de

```
ActiveCell.Formula = "=Promedio(" & Casilla_Inicial & ":" & Casilla_Final & ")"
```

Ponga

```
ActiveCell.Formula = "=Average(" & Casilla_Inicial & ":" & Casilla_Final & ")"
```

Ejecute la macro y compruebe que ahora todo funciona correctamente. Observe que en la hoja, las funciones se han insertado con su nombre correcto según el idioma, es decir SUMA y PROMEDIO.

A continuación le explicaremos como puede averiguar el nombre de cualquier función en inglés. Utilizaremos la grabadora de macros. Como ejemplo obtendremos el nombre de la función =PROMEDIO. Deberá seguir los mismos pasos para obtener el nombre de cualquier función.

1. Active la grabadora de macros.
2. Vaya a una casilla cualquiera, C1 por ejemplo y teclee la función tal como lo haría en su idioma. Por ejemplo, ponga =PROMEDIO(A1:A10) o el nombre de cualquier otra función Excel.
3. Detenga la ejecución de la macro.
4. Edite la macro y observe el nombre que se ha puesto, ya sólo debe apuntárselo y pasarlo a su procedimiento. Es posible que la función que vea tenga una nomenclatura que le suene rara y es que la grabadora de macros utiliza referencias tipo RC (row column)

Detección de Errores y Depuración de programas.

A medida que los programas van creciendo la probabilidad de cometer errores también va creciendo. Los errores se clasifican normalmente en tres categorías.

Errores en tiempo de compilación. Son los típicos errores que impiden hacer funcionar el programa debido, por ejemplo, a errores de sintaxis en las instrucciones, llamadas a funciones que no existen o llamadas con el tipo o el número de parámetros incorrectos, etc. Este tipo de errores no dan demasiados problemas, primero porque el compilador avisa de donde se han producido y luego porque simplemente revisando la sintaxis se solucionan rápidamente.

Errores en tiempo de ejecución. Estos errores se producen por una mala programación del código al no haber previsto determinados casos concretos o especiales, como por ejemplo intentar abrir un archivo que no existe, imprimir sin comprobar que la impresora está conectada, definir mal la dimensión de un array e intentar acceder a miembros que no existen, etc. Cuando se produce este tipo de errores se detiene la ejecución del programa y normalmente se informa del tipo de error que se ha producido. Muchos de estos errores se pueden solucionar mediante rutinas o funciones de tratamiento de errores, estudiaremos este tipo de rutinas un poco más adelante.

Errores de función. Son los más complicados de detectar ya que ni se detectan en la fase de ejecución, ni provocan la detención del programa, son debidos a la incorrecta programación de algún proceso y como resultado se obtienen datos erróneos. Errores de este tipo son cálculos mal hechos, bucles infinitos, devolución de valores incorrectos, etc. Como ni los detecta el compilador, ni provocan la interrupción del programa deben revisarse a mano, y claro, si el programa es extenso y trata muchos datos, su detección puede resultar dificultosa. Visual Basic, y todos los entornos de programación incorporan herramientas para facilitar la detección de este tipo de errores, son las herramientas de depuración. Antes de comenzar a describir como funcionan estas herramientas, le recomendamos, una vez más, que modularice su programa utilizando procedimientos cortos que realicen trabajos concretos y precisos, de esta forma conseguirá, además de que el programa quede más elegante y en un futuro sea más sencillo modificarlo, evitar el tener que revisar grandes bloques de código para detectar errores de este tipo.

- **Herramientas de depuración.**

Como se acaba de indicar, estas herramientas son muy útiles a la hora de testear paso a paso el funcionamiento del programa y detectar los procesos que provocan un mal funcionamiento del mismo.

Copie los datos siguientes en la primera hoja de un libro de trabajo, estos datos serán utilizados por las funciones que utilizaremos para explicar el funcionamiento de las herramientas de depuración.

	A	B	C	D	E	F	G
1		Enero	Febrero	Marzo	Abril	Mayo	Junio
2	1	3406	3336	3135	2402	4345	4891
3	2	2754	2807	3945	4780	3352	3946
4	3	3646	3704	3140	3236	2640	2052
5	4	2546	4275	4370	3193	2710	2670
6	5	3805	3533	4409	3227	3458	4917
7	6	2709	4509	3153	4894	4801	3454
8	7	2248	4293	3171	3834	3596	3258
9	8	2906	4530	3336	4770	2212	4141
10	9	3827	3538	3748	4800	3869	4896
11	10	3897	4052	4189	3132	4016	3593
12							

Copie el código siguiente, es el que utilizaremos para estudiar y ver ejemplos sobre las herramientas de depuración. La primera (*Sub Prueba*) recorre los datos de las columnas hasta encontrar una vacía, esta función va llamando a *Recorrer_Columna*, sirve para recorrer las filas de una columna hasta encontrar una vacía, va sumando los valores que encuentra en las filas y va contando cuantas hay de llenas, al final llama a la función *Cálculos*, esta función coloca en respectivas casilla, la suma de los valores de la columna, la cantidad de celdas llenas que ha encontrado, y la media. Una vez haya copiado las funciones ejecútelas para comprobar su correcto funcionamiento antes de proceder al estudio de las herramientas de depuración.

Sub Prueba()

Worksheets(1).Range("B2").Activate

' Recorrer las casillas de una fila hasta que se encuentre una vacía

Do While Not IsEmpty(ActiveCell)

 Call Recorrer_Columna

 ActiveCell.Offset(0, 1).Activate

Loop

End Sub

Private Sub Recorrer_Columna()

Dim Suma_Columna **As Long** 'Suma de los valores de la columna

Dim Mayor_Que_Cero **As Integer** ' Contar casillas con valores mayores que cero

Dim Desp_Fila **As Integer** ' Incremento de Fila

 Suma_Columna = 0

 Mayor_Que_Cero = 0

 Desp_Fila = 0

 ' Recorrer las filas de una columna hasta que se encuentre una vacía

Do While Not IsEmpty(ActiveCell.Offset(Desp_Fila, 0))

If ActiveCell.Offset(Desp_Fila, 0).Value > 0 **Then**

 Suma_Columna = Suma_Columna + ActiveCell.Offset(Desp_Fila, 0).Value

 Mayor_Que_Cero = Mayor_Que_Cero + 1

End If

 Desp_Fila = Desp_Fila + 1

Loop

Call Calcular(Suma_Columna, Mayor_Que_Cero, Desp_Fila)

End Sub

Private Sub Calcular(Suma As Long, Q As Integer, F As Integer)

 ActiveCell.Offset(F + 2, 0).Value = Suma


 ActiveCell.Offset(F + 3, 0).Value = Q

 ActiveCell.Offset(F + 4, 0).Value = Suma / Q

End Sub

Active la barra de depuración para ver los botones que se utilizarán en las secciones que se explican a continuación (**Ver/ Barras de Herramientas/ Depuración**).

Modo Ejecución paso a paso por instrucciones.

El modo paso a paso permite la ejecución del programa instrucción por instrucción, de esta forma es posible ver que el funcionamiento del programa es el correcto, es decir que la ejecución de instrucciones sigue los pasos que se habían previsto. Para ejecutar un procedimiento paso a paso, sólo debe ir pulsando sobre el botón , activar la opción de menú **Depuración /Paso a Paso por Instrucciones** o ir pulsando la tecla **F8**, que seguramente es lo más cómodo.

Ejemplo.

1. Sitúe el cursor dentro del procedimiento Prueba.
2. Vaya pulsando la tecla **F8** y verá como se ejecuta una sola instrucción por cada pulsación. Puede ir alternando con la hoja de cálculo para ver que es lo que ocurre cada vez que se ejecuta una instrucción.

Cuando esté ejecutando paso a paso puede utilizar los botones siguientes para llevar a cabo determinadas acciones.



Sirve para detener la ejecución del programa.



Sirve para ejecutar el resto del programa.




Sirve para ejecutar todo un procedimiento. Cuando en la ejecución de un procedimiento, se llega a una línea que llama a otro procedimiento o función, pulsando este botón se puede provocar la ejecución de todo el código de esta función para luego continuar con el modo paso a paso.

Ejemplo.

1. Sitúe el cursor dentro del procedimiento *Prueba*.
2. Vaya ejecutando paso a paso hasta la línea

Call Recorrer_Columna

3. En este punto pulse el botón , observe como se ejecuta toda el procedimiento *Recorrer_Columna* para luego continuar con la ejecución normal de Paso a Paso. Para activar esta opción, también puede activar la opción **Depuración/ Paso a paso por procedimientos**, o bien pulsar la combinación **MAY+F8**.




Sirve para ejecutar todas las instrucciones del procedimiento activo y volver (o terminar).

Ejemplo.

1. Sitúe el cursor dentro del procedimiento *Prueba*.
2. Vaya ejecutado paso a paso hasta la instrucción

Mayor_Que_Cero = Mayor_Que_Cero + 1


Ya dentro del procedimiento *Recorrer_Columna*.

3. Pulse sobre el botón  verá como se termina la ejecución de este procedimiento y se vuelve al procedimiento Prueba para continuar con la ejecución paso a paso. Para activar esta opción, también puede la opción Depuración/ Paso a paso para salir, o bien pulsar la combinación CONTROL+ MAY+ F8.

El modo Interrupción.

En programas largos resulta fastidioso tener que ejecutarlos paso a paso, sobretodo si sabemos que el error se produce en una parte avanzada del programa. El modo interrupción, permite la ejecución del programa hasta una instrucción determinada para, a partir de esta, ejecutar paso a paso y así poder detectar el error.

Definir puntos de interrupción.


1. Sitúe el cursor sobre la instrucción en la cual debe detenerse el programa para continuar paso a paso.
2. Pulse sobre el botón . También puede activar la opción **Depuración/ Alternar punto de interrupción**, pulsar la tecla F9 o bien hacer un clic en la parte izquierda de la ventana del módulo (la franja vertical en color gris).

Para desactivar un punto de interrupción siga los mismos pasos.

Solucionar los errores.


Todo lo dicho anteriormente no serviría de gran cosa si no fuera posible revisar los valores que las variables van cogiendo a medida que vamos avanzando o si no tuviéramos ocasión de evaluar las expresiones del programa. En las siguientes secciones veremos como llevar a cabo estas acciones.

Inspecciones rápidas de variables.

Estas opciones sirven para revisar el valor de las variables a medida que se va ejecutando el programa. Para ver los valores que van tomando las variables es conveniente tener visible la **Ventana de inspección**, para activarla **Ver/ Ventana de Inspección** o pulse sobre el botón .

Añadir una variable a la ventana de inspección.

Aunque no es necesario estar ejecutando el programa en modo paso a paso, es conveniente.


1. Seleccione la variable que desee añadir a la ventana haciendo un clic sobre ella.
2. Pulse sobre el botón , también puede activar **Depuración/ Inspección rápida** o pulsar la combinación **MaY+F9**. Aparece un cuadro de diálogo donde se muestra el valor actual de la variable. Si no está ejecutando el programa paso a paso, aparecerá el valor **Fuera de Contexto**.
3. Pulse sobre el botón **Agregar** para añadir la variable a la ventana de inspección.

Debe tener en cuenta que para revisar las variables las expresiones que les asignan valores deben de ejecutarse al menos una vez.

Ejemplo.

1. Sitúe un punto de interrupción en la línea.

$Mayor_Que_Cero = Mayor_Que_Cero + 1$

2. Ejecute el programa, cuando este se detenga en el punto de interrupción, sitúe el cursor sobre la variable *Suma_Columna* (puede ponerlo en cualquier parte).
3. Pulse sobre el botón .
4. Pulse sobre el botón **Agregar** para que la variable se inserte en la ventana Inspecciones.

Repita los pasos anteriores para las variables *Mayor_Que_Cero* y *Desp_Fila*

5. Vaya ejecutando el programa paso a paso y observe como va cambiando el valor de las variables en la ventana de inspección.

Recuerde que puede agregar una variable a la ventana de inspección aunque no esté ejecutando el programa.

*** Suponemos que ya se habrá dado cuenta que cuando ejecuta el programa paso a paso, si sitúa el puntero de ratón sobre una variable, se muestra el valor de la misma.*

Borrar una variable de la ventana de Inspección.

Sólo debe seleccionarla en la ventana de inspección y pulsar sobre la tecla **SUPR**.

Modificar el valor de una variable en tiempo de ejecución.

A veces resulta interesante cambiar el valor de alguna variable cuando se está ejecutando el programa, para ver que ocurre si coge determinados valores, para terminar un bucle, etc.

Ejemplo.

1. Sitúe un punto de interrupción en la línea.

$Mayor_Que_Cero = Mayor_Que_Cero + 1$

2. Agregue a la ventana de inspección (si no está) la variable *Suma_Columna*.
3. Ejecute el programa, al detenerse, observe en la **Ventana de Inspección** que la variable *Suma_Columna* tiene un valor que ahora cambiaremos.
4. Haga doble clic sobre el valor de *Suma_Columna* dentro de la ventana de inspección.
5. Borre el valor que tiene, cámbielo por otro y pulse **ENTER**.
6. Ahora puede continuar con la ejecución normal del programa.

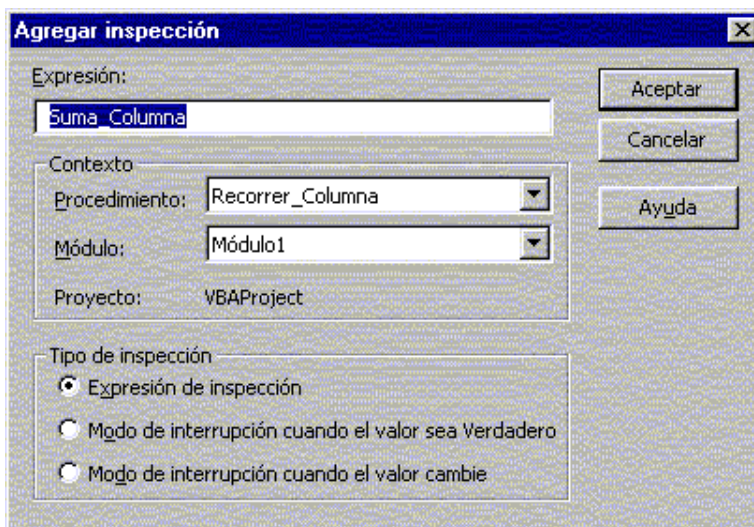
Expresiones de Revisión.

Además de permitir añadir una variable o expresión dentro de la **Ventana Inmediato**, una **Expresión de Revisión** permite interrumpir la ejecución del programa cuando una variable coge determinado valor. Piense que muchas veces un programa deja de funcionar, o funciona mal cuando una variable coge determinados valores. Con una expresión de revisión, podremos detener la ejecución del programa cuando una variable contiene determinado valor (a partie de determinado valor), luego, podremos continuar con la ejecución paso a paso para ver que ocurre a partir de este punto.

Ejemplo.

En el ejemplo que veremos a continuación, haremos que el programa se interrumpa cuando la variable *Suma_Columna* alcance un valor superior a 30000.

1. Sitúe el cursor sobre el nombre de la variable *Suma_Columna*, puede escoger cualquier posición donde aparece la variable.
2. Active la opción **Depuración/ Agregar Inspección**. Aparece el siguiente cuadro de diálogo,



Expresión : Variable o expresión que se desea evaluar.

Procedimiento: Aquí se describe el procedimiento donde debe evaluarse la variable, esto significa que el ámbito de comprobación de la variable es sólo dentro de este procedimiento. Imagine que tiene dos o más procedimientos donde utiliza una variable con el mismo nombre, o bien que tiene una variable global, aquí se indica en que procedimiento debe hacerse la evaluación.

Módulo. Lo mismo que en apartado procedimiento pero a nivel módulo.

Expresión de inspección. Activando esta opción, indicamos que lo único que deseamos hacer es incluir la variable o expresión dentro de la ventana de expresión.

Modo de interrupción cuando el valor sea verdadero. Indicamos que cuando la expresión indicada en cuadro de texto Expresión: sea cierta, debe detenerse el programa.

Modo de interrupción cuando el valor cambie. El programa se detendrá cuando la expresión cambie de valor.

Para nuestro ejemplo rellene con los datos siguientes.

Expresión : `Suma_Columna > 30000`

Procedimiento: `Recorrer_Columna`

Módulo: `Módulo1` (o como se llame su módulo).

Active **Modo de interrupción cuando el valor sea verdadero**.

3. Pulse sobre botón **Aceptar**.

Ejecute el programa. Observe que el programa se interrumpe cuando *Suma_Columna* coge un valor superior a 30000, a partir de este podríamos continuar con la ejecución paso a paso y ver como evoluciona el valor de la variable o variables.

La Ventana Inmediato.

Es otra forma de inspeccionar variables cuando el programa está en modo interrupción (ejecutándose paso a paso), pero además, ofrece la posibilidad de cambiar valores de las variables e incluso ejecutar o evaluar expresiones. Para ver el valor de una variable en la ventana inmediato debe anteponerle un ? y luego pulsar **Enter**.

Para activar la ventana Inmediato, active opción **Ver/Inmediato**, o pulse la combinación **CONTROL+G**.

Ejemplo.

En el siguiente ejemplo, utilizaremos la ventana Inmediato para ver el valor de las variables *Suma_Columna*, *Mayor_Que_Cero* y *Desp_Fila*. También cambiaremos el valor de una de ellas y comprobaremos una expresión.

1. Active la ventana de Inmediato. **Ver/ Ventana Inmediato**.
2. Sitúe un punto de interrupción en la instrucción

```
Call Calcular(Suma_Columna, Mayor_Que_Cero, Desp_Fila)
```

3. Ejecute el programa, este debe detenerse cuando llegue a la instrucción indicada en paso 2.
4. Entre en la ventana inmediato, haga las pruebas siguientes.

Escriba

```
? Suma_Columna  
? Mayor_Que_Cero  
? Desp_Fila
```

Pruebe la expresión siguiente. En este caso concreto solo sirve para ver que es una posibilidad.

```
X = Suma_Columna / Mayor_Que_Cero  
?X
```

Para terminar, cambiaremos el valor de la variable *Suma_Columna* y continuaremos la ejecución normal del programa.

```
Suma_Columna = -2350000
```

5. Quite el punto de interrupción y termine la ejecución del programa.

La instrucción Debug.Print.

Esta instrucción, que se utiliza directamente sobre el código del programa, permite ver todos los valores que ha ido cogiendo una variable o expresión durante la ejecución del programa. Los valores se mostrarán en la ventana Inmediato una vez finalizado el programa. Esta expresión resulta útil en una fase avanzada de depuración ya que permite ir viendo la evolución de una variable o expresión sin necesidad de poner puntos de interrupción. Evidentemente cuando el programa esté listo deben de sacarse.

Ejemplo.

Mediante la instrucción **Debug.Print** veremos la evolución de la variable *Suma_Columna*.

1. Sitúe el cursor después de la instrucción

```
Suma_Columna = Suma_Columna + ActiveCell.Offset(Desp_Fila, 0).Value
```

Escriba: `Debug.Print "Dentro del Bucle : " & Suma_Columna.`

2. Sitúe el cursor después de la instrucción Loop y escriba

```
Debug.Print "    Fuera del Bucle : " & Suma_Columna.
```

3. Ejecute el programa (sin puntos de interrupción).

Una vez terminada la ejecución, observe lo que hay escrito en la ventana inmediato. A veces, resulta interesante controlar en que pasos la variable ha ido cogiendo determinados valores, para hacer esto deberán declararse las correspondientes variables que hagan las funciones de contador.

Ejemplo.

Haremos lo mismo que en el ejemplo anterior pero indicando los pasos de bucle y las llamadas al procedimiento *Recorrer_Columna*.

- Declare a nivel global al variable *Contar_Llamadas* de tipo Integer.

- Declare dentro del Procedimiento *Recorrer_Columna* la variable *Pasos_De_Bucle* de tipo Integer.

- Inicialice a 1 la variable *Contar_Llamadas* dentro de la función *Prueba*, hágalo antes de la instrucción *Do While*.

- Debajo de la instrucción *ActiveCell.Offset(0, 1).Activate* ponga
Contar_Llamadas = Contar_Llamadas+1

- Inicialice a 1 la variable *Pasos_De_Bucle* dentro del procedimiento *Recorrer_Columna*, hágalo antes de la instrucción *Do While*.

- Debajo de la instrucción *Desp_Fila = Desp_Fila + 1*, ponga
Pasos_De_Bucle = Pasos_De_Bucle+1

- Cambie las expresiones *Debug.Print* por,

```
Debug.Print "Paso de Bucle: " & Paso_De_Bucle & " Suma_Columna = " &  
Suma_Columna
```

```
Debug.Print "    Llamada : " & Contar_Llamadas & " Suma_Columna = " &  
Suma_Columna
```

- Ejecute el programa y observe la salida en la ventana Inmediato.

Por supuesto cuando el programa esté terminado y comprobado deberá quitar todas las instrucciones *Debug.Print*

Y con esto terminamos el tema de la depuración. Insistimos en la importancia de estas herramientas no sólo a la hora de localizar errores de programación sino también a la hora de comprobar la evolución del programa cuando se dan determinadas condiciones. Déjenos acabar el tema insistiendo de nuevo en la importancia que modularice sus programas, observe que si lo hace, también le resultará mucho más sencillo detectar errores de programación.

Errores de Ejecución.

Es imposible excluir del todo los errores en los programas. Si además, y como es de desear, el programa será utilizado por usuarios que no han tenido nada que ver en el proceso de desarrollo e implementación posiblemente (seguramente) se producirán errores debido a su incorrecta utilización. Estos errores son los que se deben prevenir. Errores de este tipo son, por ejemplo, intentar acceder a un archivo inexistente, entrar valores incorrectos a través de un cuadro de diálogo o formulario (datos tipo String cuando se requieren números,...). También entrarían en este tipo de errores aquellos casos excepcionales pero que deben ser previstos por el programador, como por ejemplo que se llene la unidad de disco, que la impresora se quede sin papel (este ya no es tan excepcional), etc.

Visual Basic y la mayoría de lenguajes de programación permiten implementar rutinas de tratamiento de errores cuya finalidad es interceptar determinados tipo de errores que se producen en tiempo de ejecución. La finalidad de estas rutinas es que el programa no se detenga, o al menos si se detiene, informar sobre la posible causa del error e intentar controlarlo de alguna forma. Estudiaremos a continuación como se realiza esto en visual basic.

Copie el módulo siguiente, será el que utilizaremos en los ejemplos. Es un simple procedimiento que pide dos valores al usuario, los suma y los guarda en la celda A1 de Hoja2.

Option Explicit

Sub Prueba()

Dim n1 **As Integer**
Dim n2 **As Integer**
Dim total **As Integer**

n1 = InputBox("Entrar un valor", "Entrada")
n2 = InputBox("Entrar otro valor ", "Entrada")

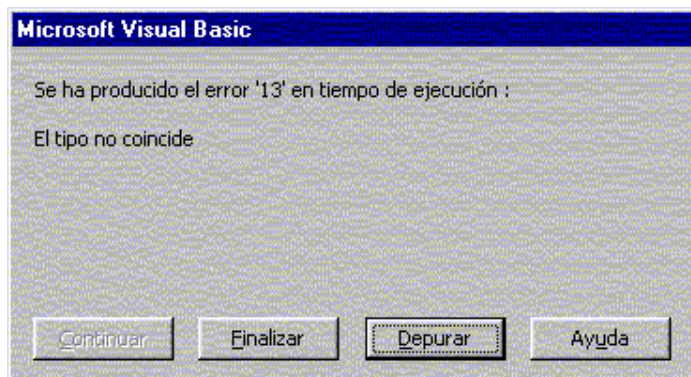
total = n1 + n2

Worksheets("Hoja2").Range("A1").Value = total

End Sub

Rutinas de tratamiento de errores.

Generalmente una rutina de tratamiento de errores reacciona ante casos esperados por el programador. En el ejemplo que nos ocupa, podría ser que el usuario entrara caracteres en lugar de números por lo que el programa generaría el error siguiente.



Mediante una rutina de tratamiento de errores informaremos del error que se ha producido y direcionaremos la ejecución del programa hacia donde interese. En visual basic el tratamiento de errores es una parte normal de la ejecución del programa. La instrucción para el tratamiento de errores es **ON ERROR GOTO línea**, línea es una etiqueta o marca de línea que indica hacia donde debe dirigirse el programa en caso de error. El formato general de un procedimiento o función donde se implementa una rutina de tratamiento de errores es la siguiente.

Sub prueba()

On Error GOTO Tratar_errores

'Instrucciones del procedimiento

Exit Sub *'Salir del procedimiento*

Tratar_Errores:

'Instrucciones de tratamiento de error

End Sub.

Con **On Error GOTO Tratar_Errores**, indicamos al programa que en caso que se produzca un error en tiempo de ejecución vaya a ejecutar las líneas que siguen a la etiqueta o marca de línea *Tratar_Errores*. Observe que antes de la etiqueta *Tratar_Errores* hay la instrucción **Exit Sub**, sirve para que si el procedimiento se ha desarrollado correctamente, salga en ese punto, tenga en cuenta que si no se pusiera esta línea la ejecución continuaría secuencialmente y se ejecutarían las líneas de la rutina *Tratar_Errores*. Si la rutina de tratamiento de errores se implementa en una función debe poner **Exit Function** en lugar de **Exit Sub**.

Escribir código de tratamiento de errores.

En este primer ejemplo haremos algo muy simple, mostrar información sobre el tipo de error que se ha producido y detener la ejecución del programa. Vale, esto es lo que hace ya Visual Basic sin necesidad de implementar nada, pero nos servirá para introducir el objeto **Err**.

El objeto Err.

Siempre que se produce un error en tiempo de ejecución Visual Basic genera (o dispara como dicen algunos autores) un objeto tipo **Err**, estudiaremos dos propiedades de este objeto **Number** y **Description**. **Number** es un número indicativo sobre el tipo de error que se ha producido, dicho de otra forma, cuando visual basic dispara un objeto **Err** en tiempo de ejecución, podemos revisar su propiedad **Number** para saber que es lo que ha causado el error. **Description** es el mensaje asociado al número de error y es una cadena de texto que describe brevemente el error. Ahora en nuestro ejemplo, implementaremos una rutina de error que muestre el número de error y su descripción, insistimos que es lo que hace visual basic por sí sólo, pero luego veremos como gracias al número de error podremos determinar la causa del error y controlar la ejecución del programa como más convenga. El procedimiento quedaría de la siguiente manera.

Sub Prueba()

```
On Error GoTo Tratar_Errores
```

```
Dim n1 As Integer
```

```
Dim n2 As Integer
```

```
Dim total As Integer
```

```
n1 = InputBox("Entrar un valor", "Entrada")
```

```
n2 = InputBox("Entrar otro valor ", "Entrada")
```

```
total = n1 + n2
```

```
Worksheets("Hoja2").Range("A1").Value = total
```

```
Exit Sub
```

Tratar_Errores:

```
MsgBox ("Número de Error : " & Err.Number & Chr(13) & "Descripción : " & Err.Description)
```

End Sub

Ejecute el procedimiento anterior y cuando se le pida el primer número entre el Texto "Hola". Observe que como se ha producido un error, el programa salta hacia la rutina de tratamiento de errores y muestra el número de error y su descripción asociada.

Continuar el programa después de la ejecución de la rutina de tratamiento de errores.

En la mayoría de los casos, las rutinas de tratamiento de errores son para que el programa no se detenga. El proceso habitual cuando se produce el error es tratar de corregirlo y continuar la ejecución del programa. La continuación del programa se consigue mediante la instrucción **Resume**. Esta instrucción da tres posibilidades, Ejecutar otra vez la instrucción que ha causado el error, continuar por la instrucción siguiente a la que ha causado el error o dirigir la ejecución hacia una línea marcada con una etiqueta

Resume. Ejecutar de nuevo la instrucción que ha causado el error.

Esto, en nuestro caso, sería volver a preguntar de nuevo el valor. La rutina de tratamiento de errores quedaría.

Tratar_Errores:

```
MsgBox ("Número de Error : " & Err.Number & Chr(13) & "Descripción : " &  
Err.Description)  
Resume
```

Podríamos cambiar el mensaje que ve el usuario para hacerlo más comprensivo. La rutina quedaría.

Tratar_Errores:

```
MsgBox ("Debe entrar un valor numérico")  
Resume
```

Resume Next. Continuar el programa por la instrucción siguiente a la que ha causado el error.

Tratar_Errores:

```
MsgBox ("Debe entrar un valor numérico")  
Resume Next
```

En este caso el programa informaría de la causa del error pero continuaría la ejecución por la línea siguiente a la que ha causado el error. No es una buena solución para este caso pero los hay en los que es la mejor.

Resume ETIQUETA. Continuar por la instrucción que sigue a la *ETIQUETA*

Haremos que el programa vuelva al principio cuando se produzca un error. Para ello debemos poner una etiqueta o marca de línea mediante la cual indicaremos en que punto debe continuar la ejecución del programa. Observe que hemos puesto la etiqueta Inicio: al principio del procedimiento. Evidentemente no siempre tiene que empezar desde el principio, puede dirigirse la ejecución hacia donde más convenga.

Sub Prueba()

On Error GoTo Tratar_Errores

Dim n1 **As Integer**

Dim n2 **As Integer**

Dim total **As Integer**

Inicio: *'Aquí vuelve el programa si se produce un error*

n1 = InputBox("Entrar un valor", "Entrada")

n2 = InputBox("Entrar otro valor ", "Entrada")

total = n1 + n2

Worksheets("Hoja2").Range("A1").Value = total

Exit Sub

Tratar_Errores:

```
MsgBox ("Debe entrar un valor numérico")
```

```
Resume Inicio
```

End Sub

Llegados a este punto ya sabemos como controlar errores simples y como redirigir la ejecución del programa. Ahora, pruebe lo siguiente, ejecute el programa y cuando se le pida el primer valor, ponga 50000.

Efectivamente, se produce un error, 50000 no es un valor válido para datos tipo Integer, recuerde que el rango para los datos tipo Integer es de -32768 a 32767. Observe que la rutina continua tratando el error y pidiendo de nuevo los datos, pero el mensaje que verá el usuario continuará siendo "Debe entrar un valor numérico", cosa que seguramente le dejará perplejo. Con esto hemos pretendido que vea que puede haber más de una causa que provoque error, consecuentemente las rutinas de tratamiento de errores deberán prevenir más de un error. En este nuevo caso el error lo produce un desbordamiento, cuyo número de error es el 6. Suponemos que ahora ya se habrá dado cuenta de la importancia del número de error ya que este nos permitirá tratar dentro de la rutina diferentes casos de error con la instrucción **Select Case**.

Nuestro ejemplo quedaría.

Sub Prueba()

```
On Error GoTo Tratar_Errores
```

```
Dim n1 As Integer
```

```
Dim n2 As Integer
```

```
Dim total As Integer
```

```
n1 = InputBox("Entrar el primer valor", "Entrada")
```

```
n2 = InputBox("Entrar el segundo valor ", "Entrada")
```

```
total = n1 + n2
```

```
Worksheets("Hoja2").Range("A1").Value = total
```

```
Exit Sub
```

Tratar_Errores:

```
Select Case Err.Number
```

```
Case 13:
```

```
MsgBox ("Debe introducir valores numéricos")
```

```
Case 6:
```

```
MsgBox ("El valor es demasiado grande o demasiado pequeño. " & _  
"El número "debe estar comprendido entre -32768 y 32767")
```

```
End Select
```

```
Resume
```

End Sub

Se estará preguntando si todavía se pueden producir más errores. La respuesta es que por supuesto que sí , por ejemplo, vaya al procedimiento y pruebe poner Hoja22 en lugar de Hoja2 en *WorkSheets(...)*. Recomendamos que termine siempre la rutina de errores con un case else de la forma siguiente,

Tratar_Errores:

```
    Select Case Err.Number
      Case 13:
        MsgBox ("Debe introducir valores numéricos")
      Case 6:
        MsgBox ("El valor es demasiado grande o demasiado pequeño. " & _
          El número debe estar comprendido entre -32768 y 32767")
      Case Else
        MsgBox ("Error no previsto. Llame al responsable facilitando la " & _
          "información que sigue " & Chr(13) & "Número de Error : " & _
          Err.Number & Chr(13) & "Descripción : " & Err.Description)
    Exit Sub ' O lo que sea

  End Select
Resume ' O lo que sea
```

Es decir, si se produce cualquier otro error no previsto, se informa al usuario que avise a quien proceda informando del número de error y su descripción.

...No se vayan todavía, aún hay más. La cosa se complica cuando en un programa se definen múltiples procedimientos y funciones, cada una de ellas puede tratar sus propios errores, pero ¿Qué pasa si el error se provoca debido a los valores que pasa la función que llama ? o ¿Cómo continuar la ejecución fuera del procedimiento?, es decir en el procedimiento que llama al que ha provocado el error. Copie los procedimientos siguientes mediante los cuales estudiaremos diferentes casos.

Sub Prueba()

```
    On Error GoTo Tratar_Errores
```

```
    Dim n1 As Integer
    Dim n2 As Integer
    Dim Total As Integer
```

```
    n1 = InputBox("Entrar el primer valor", "Entrada")
    n2 = InputBox("Entrar el segundo valor ", "Entrada")
```

```
    Call Poner_Calculo(n1, n2, 2)
```

```
Exit Sub
```

Tratar_Errores:

```
    Select Case Err.Number
      Case 13:
        MsgBox ("Debe introducir valores numéricos")
      Case 6:
        MsgBox ("El valor es demasiado grande o demasiado pequeño. " & _
          El número debe estar comprendido entre -32768 y 32767")
      Case Else
        MsgBox ("Error no previsto. Llame al responsable facilitando la " & _
          "información que sigue " & Chr(13) & "Número de Error : " & _
          Err.Number & Chr(13) & "Descripción : " & Err.Description)
    Exit Sub ' O lo que sea

  End Select
Resume ' O lo que sea
```

End Sub

Private Sub Poner_Calculo(n1 **As Integer**, n2 **As Integer**, Numero_Hoja **As Integer**)

Dim Total **As Integer**

Total = n1 + n2

Worksheets(Numero_Hoja).Range("A1").Value = Total

End Sub

Una vez entrados los valores en el procedimiento *Prueba* se llama a *Poner_Calculo*, simplemente suma los dos valores y los asigna a una hoja del libro de trabajo (observe que la hoja viene indicada por el parámetro *Numero_Hoja*).

Observe que el procedimiento *Poner_Calculo* no tiene rutina de tratamiento de errores, puede pensar que si se produce un error dentro de él se detendrá el programa, pero no es así. Si se produce un error dentro del procedimiento *Poner_Calculo* el programa pasa a ejecutar la rutina de tratamiento de errores definida dentro de *Prueba*, esto algunas veces (pocas) puede ser lo que conviene, pero compruebe el efecto indeseable que ocurre aquí.

Ejecute el programa e introduzca los valores siguientes n1=25000 y n2=20000, los dos valores entran dentro del rango de los datos Integer, pero su suma 45000 no, se producirá un error de desbordamiento en la instrucción Total = n1 + n2 del procedimiento *Poner_Calculo* y se procederá a ejecutar las instrucciones de la rutina de error definida en el procedimiento *Prueba*; aquí, después de mostrar el error, se vuelve a ejecutar la instrucción Total = n1+n2 debido a la instrucción Resume y el programa entra en un bucle infinito ya que no es posible cambiar los valores de n1 y n2. La solución aquí es sencilla, sólo debe cambiar *Resume* por *Resume Next* o *Resume Etiqueta*. Otra posibilidad es implementar una rutina de tratamiento de errores dentro del procedimiento *Poner_Calculo*, o simplemente desactivar la rutina de tratamiento de errores antes de llamar al procedimiento mediante las instrucción **On Error Goto 0**.

```
On Error Goto 0 'Desactivar el tratamiento de errores  
Call Poner_Calculo(n1, n2, 2)
```

Esto último conlleva la consecuencia que si se produce un error dentro del procedimiento *Poner_Calculo* el programa se detendrá.

Muchas veces este tipo de errores se pueden controlar mejor utilizando una variable global. Esta variable permitirá recoger el error que se ha producido en la función para luego tratarlo cuando la ejecución del programa vuelva a la función llamadora. Observe como quedaría nuestro ejemplo.

Option Explicit

Dim gError **As Integer** 'Variable que recogerá números de error

Sub Prueba()

On Error GoTo Tratar_Errores

Dim n1 **As Integer**

Dim n2 **As Integer**

Dim Total **As Integer**

n1 = InputBox("Entrar el primer valor", "Entrada")

n2 = InputBox("Entrar el segundo valor ", "Entrada")

```
Call Poner_Calculo(n1, n2, 2)
If gError <> 0 Then ' Si al volver de la función gError <> 0 --> Error dentro de la función
    MsgBox ("Error en la Función Poner Cálculo." & Chr(13) & gError)
    Exit Sub ' O lo que sea
End If
```

```
Exit Sub
```

```
Tratar_Errores:
```

```
Select Case Err.Number
    Case 13:
        MsgBox ("Debe introducir valores numéricos")
    Case 6:
        MsgBox ("El valor es demasiado grande o demasiado pequeño." & _
            El número debe estar comprendido entre -32768 y 32767")
    Case Else
        MsgBox ("Error no previsto. Llame al responsable facilitando la " & _
            "información que sigue " & Chr(13) & "Número de Error : " & _
            Err.Number & Chr(13) & "Descripción : " & Err.Description)
    Exit Sub ' O lo que sea
```

```
End Select
```

```
Resume ' O lo que sea
```

```
End Sub
```

```
Private Sub Poner_Calculo(n1 As Integer, n2 As Integer, Numero_Hoja As Integer)
```

```
    On Error GoTo Tratar_Errores2
```

```
    Dim Total As Integer
```

```
    gError = 0
```

```
    Total = n1 + n2
```

```
    Worksheets(Numero_Hoja).Range("A1").Value = Total
```

```
    Exit Sub
```

```
Tratar_Errores2:
```

```
    Error = Err.Number ' Si hay error gError recoge el número de error
```

```
End Sub
```

El funcionamiento aquí es bastante simple. Se construye una rutina de tratamiento de errores dentro de la función que simplemente asigna el número de error a la variable *gError*, cuando se devuelve el control al procedimiento llamador se inspecciona esta variable y si su valor es distinto de 0 se debe tratar el error (en el ejemplo simplemente se termina el programa, por supuesto el tratamiento podría ser diferente).

Definir errores propios.

Visual Basic incorpora la instrucción **Error** con la que es posible definir o generar errores propios. Puede que se esté preguntando para que quiere definirse errores propios, dicho de otra forma, para que provocar errores propios. Bueno, pues por ejemplo, para un mayor control en programas largos y complejos, para validar datos, y algunos procesos más.

Para provocar un error simplemente tiene que utilizar la instrucción **Error(Numero_De_Error)**, *Numero_De_Error* debe ser un valor comprendido entre 0 y 65535, es decir debe estar dentro del rango de errores definidos por Visual Basic, ahora bien, utilice siempre valores altos ya que si no, corre el riesgo de modificar un valor ya establecido por Visual Basic. Le recomendamos que empiece a partir del valor 65535 y vaya bajando, estos valores tan elevados nunca serán utilizados por Visual Basic.

Vea el siguiente ejemplo, es un caso en los que puede resultar interesante provocar un error. Es el ejemplo que hemos utilizado en la sección, el programa pide tres valores, luego llama a la función *Poner_Calculo* donde se suman *n1* y *n2* y el resultado se divide por *n3*. Observe que se ha definido una rutina de tratamiento de errores que según el error que se haya producido dentro de la función, asigna un valor a la variable global *gError* (ahora definida como tipo Long). Cuando el programa vuelve al procedimiento *Prueba* se comprueba el valor de *gError*, si es distinto de 0, se provoca un error **Error(gError)**, el programa salta entonces a la rutina de tratamiento de errores definida dentro del procedimiento donde gracias a los valores de error asignados sabemos que tipo de error se ha producido y en que función o procedimiento se produjo.

Option Explicit

Dim gError **As Long** *Variable que recogerá números de error*

Sub Prueba()

On Error GoTo Tratar_Errores

Dim n1 **As Integer**

Dim n2 **As Integer**

Dim n3 **As Integer**

Dim Total **As Integer**

n1 = InputBox("Entrar el primer valor", "Entrada")

n2 = InputBox("Entrar el segundo valor ", "Entada")

n3 = InputBox("Entrar el Tercer valor ", "Entrada")

Call Poner_Calculo(n1, n2, n3, 25)

If gError <> 0 **Then** *'Si al volver de la función gError <> 0 --> Error dentro de la función*
Error (gError) **Generar Error**

End If

Exit Sub

Tratar_Errores:

```

Select Case Err.Number
  Case 13:
    MsgBox ("Debe introducir valores numéricos")
  Case 6:
    MsgBox ("El valor es demasiado grande o demasiado pequeño. " & _
      "El número debe estar comprendido entre -32768 y 32767")
  Case 65535:
    MsgBox ("Se produjo un error en la función Poner Calculo. " & _
      "Número demasiado grande para un entero")
  Case 65534:
    MsgBox ("Se produjo un error en la función Poner Calculo. " & _
      "División por cero")
  Case 65533:
    MsgBox ("Se produjo un error en la función Poner Calculo. " & _
      "Número de Hoja no existe")
  Case Else
    MsgBox ("Error no previsto. Llame al responsable facilitando " & _
      la información que sigue " & Chr(13) & "Número de Error : " & _
      Err.Number & Chr(13) & "Descripción : " & Err.Description)
    Exit Sub ' O lo que sea

End Select
Resume Next ' O lo que sea
End Sub
```

Private Sub Poner_Calculo(n1 **As Integer**, n2 **As Integer**, n3 **As Integer**, Numero_Hoja **As Integer**)

```

  On Error GoTo Tratar_Errores2

  Dim Total As Integer

  gError = 0

  Total = n1 + n2
  Total = Total / n3

  Worksheets(Numero_Hoja).Range("A1").Value = Total
Exit Sub
```

Tratar_Errores2:

```

Select Case Err.Number
  ' Valor demasiado grande
  Case 6:
    gError = 65535


  ' División por cero
  Case 11:
    gError = 65534

  ' Subíndice fuera del intervalo (Número de hoja no existe)
  Case 9:
    gError = 65533
```



Cuadro de texto y Botón.

Lo primero que haremos es algo muy sencillo, simplemente copiaremos en Hoja2, los datos correspondientes a los registros de Hoja1 cuyos datos correspondientes a la columna *Nombre* coincidan con el que teclearemos en un cuadro de texto que insertaremos en Hoja2. Los datos se copiarán a partir de la celda A16. El botón simplemente servirá para invocar la macro que copiará los datos.

Insertar el cuadro de texto.

Sólo tiene que seleccionar el elemento  de la barra de controles y dibujarlo sobre la hoja (Hoja 2 en nuestro ejemplo, procure que coja más o menos el rango correspondiente a las celdas C2 y D2).

Insertar una etiqueta.

Las etiquetas sirven básicamente para acompañar los controles con texto descriptivo. Seleccione el botón  y dibuje en la hoja el rectángulo para insertar la etiqueta, póngalo al lado del control cuadro de texto.


Insertar un Botón.

Los botones se utilizan básicamente para invocar las macros que realizarán las acciones. No es el único control que puede invocar macros, cualquiera de los controles puede invocarlas, pero es el más habitual.

- **Cambiar las propiedades de los objetos.**

A continuación desplegaremos la ventana de propiedades para cambiar algunas de los objetos acabados de incrustar. Debe estar en modo diseño, el botón  de estar activado.

Cambiar el texto del control Label. Propiedad Caption.

1. Seleccione el control **Etiqueta**.
2. Pulse sobre el botón  de la barra de controles, se activa la ventana de **Propiedades**.
3. En la propiedad **Caption**, cambien el texto *Label1* por *Datos a Buscar*.
4. Ajuste la posición y el tamaño del control.

Cambiar el nombre del control Cuadro de Texto. Propiedad Name.

No es necesario cambiar el nombre de los controles pero si muy conveniente, tenga en cuenta que a través de los nombres de un control será como se refiera a ellos a través de las macros. Siempre es mejor llamar a un control por un nombre descriptivo que por Text1 o Command1, etc. Al control cuadro de texto le pondremos el nombre *Datos_Buscar*.

1. Seleccione el control Cuadro de Texto.
2. Si no tiene activada la ventana de propiedades, actívela.
3. En la propiedad **Name**, cambie el text1 por *Datos_Buscar*.

. Cambie la propiedad **Caption** del Botón por *Copiar Datos* y su propiedad **Name** por *Copiar_Datos* (debe poner el guión bajo ya que la propiedad **Name** no permite espacios en blanco).

Establecer la acción de copiar datos cuando se pulse el botón.

A continuación crearemos la macro que será invocada cuando se pulse el botón. La macro simplemente debe buscar en la columna A de la lista de Hoja1 el nombre que coincida con el teclado en el cuadro de texto y luego copiarlo hacia Hoja2 a partir de la casilla A16. La macro controlará que haya algo en el cuadro de texto. Se copiarán todas las coincidencias, es decir si hay dos nombres Ramón se copiarán los dos. Si no hay ninguna coincidencia se mostrará un mensaje avisando de ello.

Los eventos.

Cuando se programan controles bien sea directamente en la hoja como estamos haciendo ahora o desde un formulario, debe tener en cuenta los eventos. Un evento en cuando ocurre algo sobre un objeto, en entornos Windows constantemente se están produciendo eventos. Clicks con el ratón sobre un control, teclear sobre un cuadro de texto, etc. provocan eventos que son recogidos por el sistema. Programar un evento significa hacer que se ejecuten determinadas instrucciones cuando ocurra dicho evento. En el caso que nos ocupa ahora, haremos que las acciones necesarias para copiar los datos se ejecuten cuando se haga un clic sobre el botón *Copiar_Datos*. En general, todos los controles son capaces de capturar diferentes eventos. El sistema de eventos es bastante más complejo de lo que estudiaremos aquí, nosotros simplemente tendremos en cuenta que evento debemos elegir para lanzar la ejecución de determinado código. Veamos en la siguiente sección como asociar el código necesario para copiar datos cuando ocurre el evento click (pulsar el botón y soltarlo) sobre el botón *Copiar_Datos*.

Escribir código para el evento Click del Botón.

Deberá estar en modo **Diseño**, asegúrese que el botón  está pulsado.

1. Haga doble click sobre el botón, observe que se activa automáticamente la ventana de Visual Basic y aparece un esqueleto de función

```
Sub Copiar_Datos_Click()
```

```
End Sub
```

Es lo que se llama procedimiento de evento, es decir, este procedimiento está asociado al evento Click del Botón *Copiar_Datos*, observe que el procedimiento lleva un nombre compuesto por el nombre del control "Copiar_Datos", un guión bajo y el nombre del evento "Click", en general todos los procedimientos de evento se nombra de esta forma,

NombreDeControl_NombreDeEvento

Observe la lista de la parte superior derecha, la que tiene el elemento **Click**. Es la lista de eventos, si la despliega verá que además del elemento Click aparecen unos cuantos más **DbClick** (Doble Click) **Gotfocus** (Coger el foco), etc. todos ellos son eventos programables del control botón, es decir, podemos incluir código que se ejecutará cuando ocurren dichos eventos. Por otra parte, todos los controles tienen un evento "*por defecto*", dicho de otra forma, cuando se programa un evento del control casi siempre será ese. En el caso de nuestro botón (y de todos los botones), el evento por defecto es **Click**, observe que lo habitual es que queramos que el código se ejecute cuando se hace click sobre el botón, no cuando este coge el foco o cuando el puntero de ratón pasa sobre él, etc. El evento por defecto de un control es el que aparece cuando, en modo diseño, se hace doble clic sobre él, obviamente este se puede cambiar, por el que más nos convenga.

2. Teclar el código para llevar a cabo las acciones. Recuerde que lo que se desea hacer es copiar hacia hoja2 todos los nombres que coincidan con el que está en el cuadro de texto. El código será el que sigue, observe los comentarios.

Option Explicit

'Numero de columnas(campos) de las que consta cada registro

```
Const Num_Columnas = 6
```

```
Private Sub Copiar_Datos_Click()
```

```
Dim r1 As Range, r2 As Range
```

```
Dim encontrado As Boolean
```

'Si el cuadro de texto está vacío, no se busca nada

```
If Len(Datos_Buscar.Value) = 0 Then
```

```
MsgBox ("No hay datos que buscar")
```

```
Else
```

'Borrar los datos actuales

```
Call borrar_datos
```

'Activar Casilla A16 de Hoja2 y referenciarla con r2, Es la casilla donde se copiarán

'los datos en caso que se encuentren

```
Worksheets(2).Range("A16").Activate
```

```
Set r2 = ActiveCell
```

'Activar casilla A2 de Hoja1 y referenciarla con r1

```
Worksheets(1).Activate
```

```
Worksheets(1).Range("A2").Activate
```

```
' Recorrer todo el rango de datos de Hoja1
encontrado = False
Do While Not IsEmpty(ActiveCell)
    ' Si la casilla activa = Datos_Buscados
    If ActiveCell.Value = Datos_Buscar.Text Then
        encontrado = True
        ' Referenciar con r1 la celda donde están os datos
        Set r1 = ActiveCell
        ' Copiar los datos
        Call Copiar_Datos_Hojas(r1, r2)
        ' Referenciar con r2 la casilla donde se copiaran los próximos datos
        Set r2 = r2.Offset(1, 0)
    End If
    ActiveCell.Offset(1, 0).Activate
Loop

Worksheets(2).Activate
If encontrado Then
    MsgBox ("Datos Copiados")
Else
    MsgBox ("Ninguna coincidencia")
End If
End If

End Sub

' Procedimiento para borrar los datos de Hoja2 se llama antes de proceder a la nueva copia
Private Sub borrar_datos()

    Dim i As Integer

    Worksheets(2).Range("A16").Activate

    Do While Not IsEmpty(ActiveCell)

        For i = 0 To Num_Columnas - 1
            ActiveCell.Offset(0, i).Value = ""
        Next i
        ActiveCell.Offset(1, 0).Activate
    Loop

End Sub

' Procedimiento para copiar los datos de Hoja1 a Hoja3
' Parámetros.
'     r1 = Celda Origen
'     r2 = Celda Destino
Private Sub Copiar_Datos_Hojas(r1 As Range, r2 As Range)

    Dim i As Integer
    Dim Datos As Variant

    ' Recorrer las columnas del registro y copiar celda a celda
    For i = 0 To Num_Columnas - 1
        Datos = r1.Offset(0, i).Value
        r2.Offset(0, i).Value = Datos
    Next i

End Sub
```

- **Cuadros Combinados (ComboBox)** .

Con lo hecho hasta ahora podemos extraer de la tabla los registros cuyo nombre coincida con el teclado en el cuadro de texto. A continuación haremos que se pueda escoger el campo, es decir, podremos extraer coincidencias del *Nombre*, los *Apellidos*, la *Ciudad*, etc. Para ello incluiremos un cuadro combinado que permita escoger en que campo o columna tiene que buscarse la coincidencia. La lista, por supuesto, mostrará los nombres de las columnas.

Incluya un cuadro combinado en Hoja2 y póngale por nombre (propiedad **Name**). *Lista_Campos*

Propiedad ListFillRange.

Con esta propiedad deberemos definir los elementos que debe mostrar la lista, debe especificarse el rango que contiene los elementos a mostrar, el rango debe ser una columna (o dos, o tres, etc.). En nuestro caso el rango será J1:J6 de Hoja2 (Observe que en este rango están especificados los nombres de las columnas).

Propiedad LinKedCell.

En esta propiedad debe especificar en que celda debe copiarse el elemento seleccionado de la lista. En esta lista no utilizaremos esta propiedad.

*Cuidado con esta propiedad, tenga en cuenta que los elementos de la lista son tratados como datos de tipo **String** aunque contenga números o fechas, por lo que en estos casos, a veces será necesario aplicar funciones de conversión de datos antes que el dato se copie en la hoja. Por ejemplo, si alguna vez construye una lista con números verá que el dato seleccionado se alinea a la derecha, si son fechas, no se muestra con el formato correspondiente.*

Propiedad ListIndex.

Mediante esta propiedad podremos saber que elemento de la lista es el seleccionado por su número de orden. Es decir, si está seleccionado el primero, ListIndex valdrá 0, si está seleccionado el segundo valdrá 1, etc. Si no hay ningún elemento seleccionado valdrá -1. Tenga en cuenta que esta propiedad sólo está disponible en tiempo de ejecución, es decir la podremos leer mientras esté funcionando el programa, no se puede establecer en modo diseño, observe que no aparece en la ventana propiedades del cuadro combinado.

Bien, ya sabemos como funcionan las propiedades que utilizaremos para hacer que se extraigan de la tabla los elementos que coincidan con el valor del cuadro de texto y cuya columna o campo sea el seleccionado de la lista, veamos como quedará la macro.

En primer lugar cree un procedimiento llamado **Proceder** donde deberá copiar todo el código que ahora está en **Copiar_Datos**. Debemos hacer esto porque antes de proceder se deben hacer ciertas comprobaciones que ya iremos viendo conforme avanzamos, por el momento la comprobación a hacer es la de ver sobre que campo o columna se deben buscar las coincidencias con los datos tecleados en el cuadro de texto. La función **Copiar_Datos** quedará de la forma siguiente.

```
Private Sub Copiar_Datos_Click()
```

```
    Dim i As Integer
```

```
    ' Recoger el elemento seleccionado de la lista
```

```
    i = Lista_Campos.ListIndex
```

```
    ' Si i < 0 es que no hay ningún elemento seleccionado.
```

```
    If i < 0 Then
```

```
        MsgBox ("Debe Seleccionar un campo de la lista")
```

```
    Else
```

```
        ' Llamar a proceder para iniciar la copia.
```

```
        Call Proceder(i)
```

```
    End If
```

```
End Sub
```

La cabecera de la función proceder quedará de la forma siguiente.

```
' Procedimiento Proceder
```

```
' Inicia la copia de los datos coincidentes
```

```
' Parámetros:
```

```
'     Columna = Elementos seleccionado de la lista que coincidirá con la columna sobre la que se  
'             debe buscar
```

```
Private Sub Proceder(Columna As Integer)
```

```
.....
```

Ahora, dentro del procedimiento Proceder cambie la línea

```
    If ActiveCell.Value = Datos_Buscar.Text Then
```

Por

```
    If ActiveCell.Offset(0, Columna).Value = Datos_Buscar.Text Then
```

Explicación del proceso. Cuando se selecciona un elemento de la lista, su propiedad *ListIndex* es igual al orden que ocupa dicho elemento en la lista, supongamos que escogemos *Ciudad*, *ListIndex* valdrá 2. Este valor se pasa a *Proceder* y es recogido por el parámetro *Columna*. Ahora observe la línea

```
    If ActiveCell.Offset(0, Columna).Value = Datos_Buscar.Text Then
```

Es decir la coincidencia con el valor del cuadro de texto *Datos_Buscar* se busca respecto a la casilla que está a la derecha (offset) de la activa, tantas columnas como las expresadas por el valor de la variable *Columna*. Observe que en este caso la casilla activa siempre corresponde a una de la columna A, si la variable *Columna* vale 2 la coincidencia se buscará respecto al valor de la columna C (2 más hacia la derecha) y que coincide con la columna correspondiente a la *Ciudad*.

Segunda Lista.

Ahora crearemos una lista donde sea posible escoger la relación de comparación. Hasta ahora la extracción se realizaba con aquellos elementos iguales al valor entrado en el cuadro de texto **Datos_Buscar**, esta segunda lista permitirá escoger si los elementos a extraer deben ser *Igual, Menores, Mayores, Menores Iguales o Mayores Iguales* que el valor de **Datos_Buscar**. Para ello debe construir una segunda lista con las propiedades siguientes.

Name = Lista_Comparación.

ListFillRange = L1:L5 Observe que en este rango están los valores correspondientes a la operación relacional que se desea realizar (Igual, Menor, etc.)

Obviamente deberemos modificar las funciones para realizar las operaciones con respecto al elemento seleccionado en el cuadro de lista **Lista_Comparación**. Dejaremos el procedimiento **Proceder** de la forma siguiente y crearemos una función que haga las comparaciones, esta función a la que hemos llamado **Comparar** devuelva el valor True si el resultado de la comparación es Cierto y False si es falso.

' Procedimiento Proceder

' Inicia la copia de los datos coincidentes

' Parámetros:

*' Columna = Elementos seleccionado de la lista que coincidirá con la columna sobre la que se debe
' buscar*

Private Sub Proceder(Columna As Integer)

Dim r1 **As Range**, r2 **As Range**

Dim encontrado **As Boolean**

Dim Valor_Comparacion **As Boolean**

' Si el cuadro de texto está vacío, no se busca nada

If Len(Datos_Buscar.Value) = 0 **Then**

MsgBox ("No hay datos que buscar")

Else

' Borrar los datos actuales

Call borrar_datos

' Activar Casilla A16 de Hoja2 y referenciarla con r2' Es la casilla donde se copiarán

' los datos en caso que' se encuentren

Worksheets(2).Range("A16").Activate

Set r2 = ActiveCell

' Activar casilla A2 de Hoja1 y referenciarla con r1

Worksheets(1).Activate

Worksheets(1).Range("A2").Activate

encontrado = **False**

' Recorrer todo el rango de datos de Hoja 1

Do While Not IsEmpty(ActiveCell)

Valor_Comparacion = Comparar(ActiveCell.Offset(0, Columna).Value, _
Datos_Buscar.Value, Lista_Comparacion.ListIndex)

If Valor_Comparacion = True **Then**

encontrado = True

' Referenciar con r1 la celda donde están os datos

Set r1 = ActiveCell

' Copiar los datos

Call Copiar_Datos_Hojas(r1, r2)

' Referenciar con r2 la casilla donde se copiaran los próximos datos

```
                Set r2 = r2.Offset(1, 0)
            End If
            ActiveCell.Offset(1, 0).Activate
        Loop

        Worksheets(2).Activate
        If encontrado Then
            MsgBox ("Datos Copiados")
        Else
            MsgBox ("Ninguna coincidencia")
        End If
    End If

End Sub

' Función que compara dos valores con un operador relacional =, >, <, etc.
' La función devuelve True o False en función de la comparación.
' Parámetros.
' Valor1 y Valor2 = Valores que se comparan
' Signo = variable que sirve para escoger el operador relacional
' en función de su valor, ver estructura Select Case

Private Function Comparar(Valor1 As Variant, Valor2 As Variant, Operador As Integer) As Boolean

    Dim q As Boolean

    Select Case Operador
        Case 0:
            q = Valor1 = Valor2
        Case 1:
            q = Valor1 > Valor2
        Case 2:
            q = Valor1 < Valor2
        Case 3:
            q = Valor1 >= Valor2
        Case 4:
            q = Valor1 <= Valor2
    End Select

    Comparar = q

End Function
```

Observe la línea que llama a la función *Comparar*,

```
Valor_Comparacion = Comparar(ActiveCell.Offset(0, Columna).Value, _
                             Datos_Buscar.Value, Lista_Comparacion.ListIndex)
```

ActiveCell.Offset(0,Columna) serán los valores que se compararán con el valor del cuadro de texto.

Datos_Buscar.value es el valor del cuadro de texto.

Lista_Comparacion.ListIndex devuelve el índice del elemento seleccionado de la lista, observe como se utiliza este valor en la estructura **Select Case** de *Comparar* para determinar que operador utilizar.

Pero todo esto no funcionará.

Pruebe lo siguiente.

En el cuadro de texto escriba *Madrid* (o simplemente M).
Seleccione de la lista de Campos *Ciudad*.
Seleccione de la lista de operadores *Mayor*.
Pulse sobre el botón y observe que se copian todos los registros cuyo campo Ciudad sea superior a Madrid (o a la M). Hasta aquí todo correcto.

Ahora pruebe lo siguiente.

En el cuadro de texto escriba 100000
Seleccione de la lista de Campos *Cantidad*.
Seleccione de la lista de operadores *Mayor*.
Pulse sobre el botón y observe que no se copia nada a pesar que en cantidad hay registros con el valor superior a 100000.

Recuerde que los valores de un cuadro de texto son siempre datos tipo String, entonces en este caso estarán comparándose valores String (los del cuadro de texto) con valores numéricos, (los recogidos de la columna *Cantidad*). Tenga en cuenta siempre esta circunstancia cuando trabaje con elementos de formulario. Vea la sección siguiente donde se solucionará este problema y de paso se verá como construir Lista con más de una columna.

Listas con más de una columna.

Para solucionar el problema del apartado anterior utilizaremos una segunda columna cuyos elementos indicarán el tipo de datos del campo. Observe el rango K1:K6 de Hoja2, las letras significan lo siguiente *T campo tipo texto o string*, *N campo Numérico*, *F campo fecha*. Para incluir esta segunda columna en la lista deberá cambiar las propiedades siguientes.

ListFillRange, J1:K6
ColumnCount, 2 (Indicamos el número de columnas de la lista.

Además especificaremos el ancho de cada columna mediante la propiedad,

ColumnWidths, 4pt;0pt Debe separar el ancho de cada columna mediante un punto y coma.
Observe que la segunda columna no se mostrará debido a que hemos especificado el ancho a 0.

ColumnBound 1 significa que el dato que recogerá la propiedad Value corresponde al elemento seleccionado de la primera columna.

Si desea recoger datos le la segunda columna deberá utilizar la propiedad

Column(Numero de Columna, Indice del elemento seleccionado)

Las columnas empiezan a numerarse a partir de la 0.

La función *Comparar* y su correspondiente llamada quedarán de la forma siguiente. Observe que se han incluido variables que recogen los valores de *Lista_Comparación* y *Lista_Campos*, simplemente lo hemos hecho para que quede más claro.

```
.....  
Do While Not IsEmpty(ActiveCell)  
  
    'Recoger el Signo de comparación  
    Signo = Lista_Comparacion.ListIndex  
    'Recoger el tipo de datos  
    Tipo_Datos = Lista_Campos.Column(1, Columna)  
  
    Valor_Comparacion = Comparar(ActiveCell.Offset(0, Columna).Value, _  
        Datos_Buscar.Value, Signo, Tipo_Datos)
```

La función Comparar.

```
Private Function Comparar(Valor1 As Variant, Valor2 As Variant, Operador As Integer, Tipo As String) As Boolean
```

```
    Dim q As Boolean
```

```
    ' Conversión del tipo de datos de las variables
```

```
    Select Case Tipo
```

```
        Case "N": ' Convertir a número
```

```
            Valor2 = Val(Valor2)
```

```
        Case "F": ' Convertir a Fecha
```

```
            Valor2 = CDate(Valor2)
```

```
    End Select
```

```
    Select Case Operador
```

```
        Case 0:
```

```
            q = Valor1 = Valor2
```

```
        Case 1:
```

```
            q = Valor1 > Valor2
```

```
        Case 2:
```

```
            q = Valor1 < Valor2
```

```
        Case 3:
```

```
            q = Valor1 >= Valor2
```

```
        Case 4:
```

```
            q = Valor1 <= Valor2
```

```
    End Select
```

```
    Comparar = q
```

```
End Function
```


- **Control Numérico** .

Inserte un control de número y póngale por nombre (propiedad **Name**) *Numero*.

Establezca su propiedad **Orientation** a *fnOrientationVertical* para que se alinee verticalmente.

Este control se utiliza normalmente para aumentar y disminuir valores numéricos de un cuadro de texto, aunque por supuesto puede utilizarse para otras funciones. Utilizaremos un control de este tipo para aumentar los valores del Cuadro de Texto *Datos_Buscar* pero sólo si el campo seleccionado de *Lista_Campos* es de tipo numérico. Para ello activaremos este control únicamente cuando el campo seleccionado sea de este tipo. Para activar o desactivar un control se utiliza la propiedad **Enabled**, si está a true el control estará activado y sino estará desactivado.

Observe que la acción de activar o desactivar el control de número deberemos hacerlo cuando se seleccione un elemento de *Lista_Campos*. Es decir el código deberemos insertarlo en el evento **Change** (cambio) de *Lista_Campos*. Haga doble clic sobre el elemento *Lista_Campos* para desplegar su procedimiento de evento. El código para controlar la activación del control es el que sigue,

```
Private Sub Lista_Campos_Change()
```

```
    Dim i As Integer
```

```
    Dim Tipo_Datos As String
```

```
    i = Lista_Campos.ListIndex
```

```
    If i >= 0 Then
```

```
        Tipo_Datos = Lista_Campos.Column(1, i)
```

```
        If Tipo_Datos = "N" Then
```

```
            Numero.Enabled = True
```

```
        Else
```

```
            Numero.Enabled = False
```

```
        End If
```

```
    End If
```

```
End Sub
```

Establecer los valores del control de número.

Para establecer los valores que puede devolver un control de número se deben modificar las propiedades siguientes.

Max, establece el valor máximo que puede tener el control.

Min, establece el valor mínimo que puede tener el control.

Smallchange, establece el incremento o decremento para la propiedad value cada vez que se pulse sobre alguno de los dos botones.

Estos valores se pueden establecer en tiempo de diseño, pero lo que haremos a continuación será establecerlos en tiempo de ejecución dependiendo del campo que se seleccione en *Lista_Campos*. Estableceremos los valores siguientes.

Para campo *Edad*.

```
Max = 99
```

```
Min = 18
```

```
SmallChange = 1
```

Para campo cantidad.

```
Max = 500.000
Min = 10.000
SmallChange = 1.000
```

Deberemos modificar el código del procedimiento de evento *Lista_Campos_Change* de la forma siguiente.

```
Private Sub Lista_Campos_Change()

    Dim i As Integer
    Dim Tipo_Datos As String

    i = Lista_Campos.ListIndex

    If i >= 0 Then

        Tipo_Datos = Lista_Campos.Column(1, i)

        If Tipo_Datos = "N" Then
            Numero.Enabled = True
            If Lista_Campos.Value = "Edad" Then
                Numero.Min = 18
                Numero.Max = 99
                Numero.SmallChange = 1
                Datos_Buscar.Value = 0
                Numero.Value=0
            End If
            If Lista_Campos.Value = "Cantidad" Then
                Numero.Min = 10000
                Numero.Max = 500000
                Numero.SmallChange = 1000
                Datos_Buscar .Value= 0
                Numero.Value=0
            End If
        Else
            Numero.Enabled = False
        End If

    End If

End Sub
```

Y para terminar ya sólo debemos codificar el evento **Change** del control de número para que el Cuadro de texto vaya incrementando o decrementando su valor cada vez que se haga clic sobre el control.

```
Private Sub Numero_Change()

    Datos_Buscar.Value = Numero.Value

End Sub
```

Pero además debemos hacer que cuando se cambie el valor del cuadro de texto también se cambie el del control de número de forma que cuando se pulse sobre él, incremente o decremente a partir del valor que hay en el cuadro de texto. Si no se hace así, el incremento o decremento se hará en función del valor que tenga el control de número no el que está en el cuadro de texto. Modificaremos el evento **Change** del cuadro de texto. Observe que se controla que sólo se ejecute la acción si el control de número está activado, además se debe controlar también el valor máximo y mínimo que puede contener el cuadro de texto, si no se hiciera así se generaría un error al poner un valor mayor o menor que los definidos en las propiedades Max y Min del control de número.

```
Private Sub Datos_Buscar_Change()
```

```
    ' Si el numero de control está activado
```

```
    If Numero.Enabled Then
```

```
        ' No permite coger valores superiores a la propiedad Max
```

```
        If Val(Datos_Buscar.Value) > Numero.Max Then
```

```
            MsgBox ("Valor demasiado grande")
```

```
            Datos_Buscar.Value = Numero.Max
```

```
        Else
```

```
            ' No permite coger valores inferiores a la propiedad Min
```

```
            If Val(Datos_Buscar.Value) < Numero.Min Then
```

```
                MsgBox ("Valor demasiado pequeño")
```

```
                Datos_Buscar.Value = Numero.Min
```

```
            Else
```

```
                Numero.Value = Val(Datos_Buscar.Value)
```

```
            End If
```

```
        End If
```

```
    End If
```

```
End Sub
```

Antes de proceder con el siguiente control déjenos remarcar que la programación de controles implica que muchas veces unos dependan de otros por lo que debe ser extremadamente cuidadoso en la elaboración del código de los eventos. Observe las últimas operaciones que hemos realizado debido a la interdependencia de dos controles .

- **Casillas de verificación (CheckBox)** .

Estos controles se suelen utilizar para activar o desactivar la ejecución de determinadas acciones. Casi siempre implican una estructura condicional a la hora de hacer alguna cosa,

```
    Si las casilla está activada Entonces
```

```
        Acciones
```

```
        ....
```

```
    Fin Si
```

A continuación utilizaremos una casilla de verificación que si está activada provocará que los datos tipo texto se conviertan a mayúscula antes de copiarse, se utilizará la función **Ucase**. Simplemente se deberá comprobar que la casilla esté activada y si lo está proceder a la conversión (sólo si el dato es tipo texto).

Inserte un control casilla de verificación. Establezca las siguientes propiedades.

Name, Mayusculas.

Captión, Mayúsculas.

Para comprobar si la casilla está activada o no simplemente debe mirarse su propiedad **Value**. Si vale true es que está activada, sino valdrá False.

Vea como quedará el procedimiento *Copiar_Datos_Hojas*, observe que además de comprobar que la casilla esté activada se utiliza la función **TypeName** para comprobar si los datos que se van a copiar son del tipo String, si no lo fueran, la función **Ucase** provocaría un error. **TypeName(Expresión)** devuelve una cadena que indica el tipo de datos de la expresión.

```
Private Sub Copiar_Datos_Hojas(r1 As Range , r2 As Range )
```

```
    Dim i As Integer
```

```
    Dim Datos As Variant
```

```
    'recorrer las columnas del registro y copiar celda a celda
```

```
    For i = 0 To Num_Columnas - 1
```

```
        ' Si la casilla Mayúsculas está activada y el tipo de datos es String
```

```
        If Mayusculas.Value = True And TypeName(r1.Offset(0, i).Value) = "String" Then
```

```
            Datos = UCase(r1.Offset(0, i).Value)
```

```
        Else
```

```
            Datos = r1.Offset(0, i).Value
```

```
        End If
```

```
        r2.Offset(0, i).Value = Datos
```

```
    Next i
```

```
End Sub
```

• Botones de Opción (Option Button) .

Los botones de opción se utilizan para elegir una única opción entre una serie de ellas, es decir, de un grupo de opciones sólo permitirán que se escoja una. De la misma forma que las casillas de verificación, casi siempre implican una estructura condicional para comprobar cual de ellas está activada. El botón activado tendrá su propiedad **Value** igual a true.

Como ejemplo de su utilización crearemos dos botones de opción que sirvan para que a la hora de copiar datos hacia la hoja, se copie sólo los valores de *Nombre* y *Apellidos* o todos como hasta ahora.

Incluya dos botones de opción y establezca las siguientes propiedades.

Botón1.

Name, Todo.

Caption, Todo.

Botón2.

Name, Solo_Nombre.

Caption, Nombre y Apellidos.

Si está activado el primer botón deberán copiarse todos los datos mientras que si está activado el segundo solo se copiarán el *Nombre* y los *Apellidos*. El procedimiento *Copiar_Datos_Hojas* quedará de la forma siguiente.

Private Sub Copiar_Datos_Hojas(r1 **As Range** , r2 **As Range**)

Dim i **As Integer**

Dim Datos **As Variant**

Dim Final **As Integer**

' Si Botón Todo Activado, se copian todas las columnas

If Todo.Value = True **Then**

Final = Num_Columnas - 1

Else *' Sólo se copian las dos primera columnas*

Final = 1

End If

' recorrer las columnas del registro y copiar celda a celda

For i = 0 To **Final**

' Si la casilla Mayúsculas está activada y el tipo de datos es String

If Mayusculas.Value = True **And** TypeName(r1.Offset(0, i).Value) = "String" **Then**

Datos = UCase(r1.Offset(0, i).Value)

Else

Datos = r1.Offset(0, i).Value

End If

r2.Offset(0, i).Value = Datos

Next i

End Sub

Y aquí terminamos el estudio de cómo se pueden utilizar los controles de formulario dentro de una hoja de cálculo. Recordarle para terminar que debe ser extremadamente cuidadoso con el código que utilice en los procedimientos de evento sobre todo si los controles se interrelacionan entre ellos.

El siguiente Listado incluye todo el código que se ha utilizado. Puede probar su funcionamiento cargando el archivo **Lista7_2.xls** que debió descargarse junto a este tema.

Option Explicit

'Numero de columnas(campos) de las que consta cada registro

Const Num_Columnas = 6

Private Sub Copiar_Datos_Click()

Dim i **As Integer**

Dim x **As Integer**

'Recoger el elemento seleccionado de la lista

i = Lista_Campos.ListIndex

'Si i < 0 no está seleccionado ningún elemento

If i < 0 **Then**

MsgBox ("Debe Seleccionar un campo de la lista")

Else

x = Lista_Comparacion.ListIndex

If x < 0 **Then**

MsgBox ("Debe Seleccionar uno operador de Comparación")

Else

'llamar a proceder

Call Proceder(i)

End If

End If

End Sub

'Procedimiento Proceder

'Inicia la copia de los datos coincidentes

'Parámetros:

' Columna = Elementos seleccionado de la lista que coincidirá

' con la columna sobre la que se debe buscar

Private Sub Proceder(Columna **As Integer**)

Dim r1 **As Range**, r2 **As Range**

Dim encontrado **As Boolean**

Dim Valor_Comparacion **As Boolean**

Dim Signo **As Integer**

Dim Tipo_Datos **As String**

'Si el cuadro de texto está vacío, no se busca nada

If Len(Datos_Buscar.Value) = 0 **Then**

MsgBox ("No hay datos que buscar")

Else

'Borrar los datos actuales

Call borrar_datos

'Activar Casilla A16 de Hoja2 y referenciarla con r2

'Es la casilla donde se copiarán los datos en caso que se encuentren

Worksheets(2).Range("A16").Activate

Set r2 = ActiveCell

'Activar casilla A2 de Hoja1 y referenciarla con r1

Worksheets(1).Activate

Worksheets(1).Range("A2").Activate

'Recorrer todo el rango de datos de Hoja1

encontrado = False

Do While Not IsEmpty(ActiveCell)

```
' Recoger el Signo de comparación
Signo = Lista_Comparacion.ListIndex
' recoger el tipo de datos
Tipo_Datos = Lista_Campos.Column(1, Columna)

Valor_Comparacion = Comparar(ActiveCell.Offset(0, Columna).Value, _
    Datos_Buscar.Value, Signo, Tipo_Datos)

If Valor_Comparacion = True Then
    encontrado = True
    ' Referenciar con r1 la celda donde están os datos
    Set r1 = ActiveCell
    ' Copiar los datos
    Call Copiar_Datos_Hojas(r1, r2)
    ' Referenciar con r2 la casilla donde se copiaran los próximos datos
    Set r2 = r2.Offset(1, 0)
End If
ActiveCell.Offset(1, 0).Activate

Loop

Worksheets(2).Activate
If encontrado Then
    MsgBox ("Datos Copiados")
Else
    MsgBox ("Ninguna coincidencia")
End If
End If

End Sub

' Función que compara dos valores con un operador relacional =, >, <, etc.
' La función devuelve True o False en función de la comparación.
' Parámetros.
' Valor1 y Valor2 = Valores que se comparan
' Signo = variable que sirve para escoger el operador relacional
' en función de su valor, ver estructura Select Case

Private Function Comparar(Valor1 As Variant, Valor2 As Variant, Operador As Integer, Tipo As String) As Boolean

    Dim q As Boolean

    Select Case Tipo
        Case "N": ' Convertir a número
            Valor2 = Val(Valor2)
        Case "F": ' Convertir a Fecha
            Valor2 = CDate(Valor2)
    End Select

    Select Case Operador
        Case 0:
            q = Valor1 = Valor2
        Case 1:
            q = Valor1 > Valor2
        Case 2:
            q = Valor1 < Valor2
```

```
Case 3:
q = Valor1 >= Valor2
```

```
Case 4:
q = Valor1 <= Valor2
```

```
End Select
```

```
Comparar = q
```

```
End Function
```

' Procedimiento para borrar los datos de Hoja2 se llama antes de proceder a la nueva copia

```
Private Sub borrar_datos()
```

```
Dim i As Integer
```

```
Worksheets(2).Range("A16").Activate
```

```
Do While Not IsEmpty(ActiveCell)
```

```
For i = 0 To Num_Columnas - 1
ActiveCell.Offset(0, i).Value = ""
```

```
Next i
ActiveCell.Offset(1, 0).Activate
```

```
Loop
```

```
End Sub
```

' Procedimiento para copiar los datos de Hoja1 a Hoja3

' Parámetros.

' r1 = Celda Origen

' r2 = Celda Destino

```
Private Sub Copiar_Datos_Hojas(r1 As Range, r2 As Range)
```

```
Dim i As Integer
```

```
Dim Datos As Variant
```

```
Dim Final As Integer
```

' Si Botón Todo Activado, se copian todas las columnas

```
If Todo.Value = True Then
Final = Num_Columnas - 1
```

```
Else ' Sólo se copian las dos primera columnas
Final = 1
```

```
End If
```

' recorrer las columnas del registro y copiar celda a celda

```
For i = 0 To Final
```

' Si la casilla Mayúsculas está activada y el tipo de datos es String

```
If Mayusculas.Value = True And TypeName(r1.Offset(0, i).Value) = "String" Then
Datos = UCase(r1.Offset(0, i).Value)
```

```
Else
Datos = r1.Offset(0, i).Value
```

```
End If
r2.Offset(0, i).Value = Datos
```

```
Next i
```

```
End Sub
```



```
Private Sub Datos_Buscar_Change()  
  
    ' Si el número de control está activado  
    If Numero.Enabled Then  
        ' No permite coger valores superiores a la propiedad Max  
        If Val(Datos_Buscar.Value) > Numero.Max Then  
            MsgBox ("Valor demasiado grande")  
            Datos_Buscar.Value = Numero.Max  
  
        Else  
            ' No permite coger valores inferiores a la propiedad Min  
            If Val(Datos_Buscar.Value) < Numero.Min Then  
                MsgBox ("Valor demasiado pequeño")  
                Datos_Buscar.Value = Numero.Min  
  
            Else  
                Numero.Value = Val(Datos_Buscar.Value)  
  
            End If  
  
        End If  
  
    End Sub
```

```
Private Sub Lista_Campos_Change()  
  
    Dim i As Integer  
    Dim Tipo_Datos As String  
  
    i = Lista_Campos.ListIndex  
  
    If i >= 0 Then  
  
        Tipo_Datos = Lista_Campos.Column(1, i)  
  
        If Tipo_Datos = "N" Then  
            Numero.Enabled = True  
            If Lista_Campos.Value = "Edad" Then  
                Numero.Min = 18  
                Numero.Max = 99  
                Numero.SmallChange = 1  
                Datos_Buscar.Value = 0  
                Numero.Value=0  
  
            End If  
            If Lista_Campos.Value = "Cantidad" Then  
                Numero.Min = 10000  
                Numero.Max = 500000  
                Numero.SmallChange = 1000  
                Datos_Buscar .Value= 0  
                Numero.Value=0  
  
            End If  
  
        Else  
            Numero.Enabled = False  
  
        End If  
  
    End If
```

End Sub

```
Private Sub Numero_Change()  
    Datos_Buscar.Value = Numero.Value
```

End Sub