

# MS Excel 2007 con Programación de Macros en VBA



Introduce al participante en los conocimientos para utilizar la herramienta de Excel 2007 en aplicaciones que permitan optimizar su tiempo mediante la programación con el lenguaje Visual Basic.

Ing. Patricia Acosta, MSc.

E-mail: [acostanp@gmail.com](mailto:acostanp@gmail.com)

Website: <http://www.saccec.com/>

Blog:

<http://www.aulaexcelavanzado.blogspot.com/>

Mayo- 2010

CONTENIDO

<b>MACROS EN MS EXCEL.....</b>	<b>4</b>
<b>GRABAR UNA MACRO .....</b>	<b>4</b>
<b>SEGURIDAD EN MACROS .....</b>	<b>6</b>
<b>CODIGOS DE UNA MACRO DE EXCEL.....</b>	<b>9</b>
<b>CÓDIGOS MÁS COMUNES .....</b>	<b>16</b>
<b>CUADRO DE CONTROL – CONTROLES ACTIVEX.....</b>	<b>18</b>
<b>CREANDO FORMULARIOS Y PROGRAMÁNDOLOS .....</b>	<b>22</b>
<b>TRABAJANDO CON FORMULAS .....</b>	<b>28</b>
<b>CÓDIGO PARA CARGAR UN FORMULARIO DESDE EXCEL .....</b>	<b>30</b>
<b>ASIGNAR UNA MACRO A UNA AUTOFORMA.....</b>	<b>34</b>
<b>PROTEGER UNA HOJA EN AMBIENTE VBA.....</b>	<b>36</b>
<b>COLOCAR UNA CLAVE AL PROYECTO DE VBA .....</b>	<b>38</b>
OBJETOS, PROPIEDADES Y MÉTODOS. ....	39
<b>CONCEPTOS QUE ENCONTRAREMOS EN EXCEL .....</b>	<b>40</b>
<b>PRACTICA II .....</b>	<b>51</b>
<b>CÓDIGOS MÁS COMUNES .....</b>	<b>52</b>
<b>ESTRUCTURAS CONDICIONALES.....</b>	<b>66</b>
<b>ESTRUCTURA IF..ELSE .....</b>	<b>70</b>
<b>ESTRUCTURAS IF ANIDADAS.....</b>	<b>73</b>
<b>OPERADORES LÓGICOS. ....</b>	<b>75</b>
OPERADOR LÓGICO AND (Y). ....	75
OPERADOR LÓGICO OR (O).....	77
OPERADOR LÓGICO NOT (NO).....	79
<b>ESTRUCTURA SELECT CASE.....</b>	<b>80</b>
<b>LA FUNCIÓN MSGBOX.....</b>	<b>86</b>
LA INSTRUCCIÓN WITH. ....	90
<b>ESTRUCTURAS REPETITIVAS.....</b>	<b>93</b>
<b>ESTRUCTURA REPETITIVA PARA (FOR).....</b>	<b>95</b>
<b>PROPIEDADES ROW Y COLUMN.....</b>	<b>99</b>
<b>ESTRUCTURA REPETITIVA DO WHILE..LOOP (HACER MIENTRAS) .....</b>	<b>101</b>
ESTRUCTURA DO..LOOP UNTIL (HACER.. HASTA QUE SE CUMPLA LA CONDICIÓN).....	109
<b>PROCEDIMIENTOS Y FUNCIONES.....</b>	<b>112</b>
DEFINIR UN PROCEDIMIENTO.....	113
LLAMAR A UN PROCEDIMIENTO.....	113

<b>GENERALIZAR UNA FUNCIÓN.....</b>	<b>116</b>
PARÁMETROS.....	116
<b>VARIABLES LOCALES Y VARIABLES GLOBALES .....</b>	<b>120</b>
PASO POR REFERENCIA Y PASO POR VALOR. ....	124
<b>FUNCIONES .....</b>	<b>127</b>
APLICACIÓN DE EJEMPLO. ....	129
CUADRO DE TEXTO Y BOTÓN. ....	130
INSERTAR EL CUADRO DE TEXTO.....	130
INSERTAR UNA ETIQUETA.....	130
INSERTAR UN BOTÓN. ....	130
CAMBIAR LAS PROPIEDADES DE LOS OBJETOS. ....	131
CAMBIAR EL TEXTO DEL CONTROL LABEL. PROPIEDAD CAPTION. ....	131
CAMBIAR EL NOMBRE DEL CONTROL CUADRO DE TEXTO. PROPIEDAD NAME. ....	131
ESTABLECER LA ACCIÓN DE COPIAR DATOS CUANDO SE PULSE EL BOTÓN.....	132
<b>LOS EVENTOS. ....</b>	<b>132</b>
ESCRIBIR CÓDIGO PARA EL EVENTO CLICK DEL BOTÓN. ....	133
PROPIEDAD LISTFILLRANGE.....	138
PROPIEDAD LINKEDCELL. ....	138
PROPIEDAD LISTINDEX. ....	138
LISTAS CON MÁS DE UNA COLUMNA.....	146
ESTABLECER LOS VALORES DEL CONTROL DE NÚMERO.....	149
CELDA DE VERIFICACIÓN (CHECKBOX) . ....	153
BOTONES DE OPCIÓN (OPTION BUTTON) .....	155
<b>REFERENCIAS ELECTRÓNICAS .....</b>	<b>158</b>

### MACROS EN MS EXCEL

#### Introducción

Al trabajar con un libro personalizado, es decir, que nos hemos definido con una serie de características específicas como puedan ser el tipo de letra, el color de ciertas celdas, los formatos de los cálculos y características similares, perdemos mucho tiempo en formatear todo el libro si disponemos de muchas hojas.

Con las macros lo que se pretende es automatizar varias tareas y fusionarlas en una sola, añadiendo por ejemplo un botón en nuestro libro que al pulsar sobre él realice todas esas tareas.

#### GRABAR UNA MACRO

La forma más fácil e intuitiva de crear macros es crearlas mediante la grabadora de macros del que dispone Excel.

Esta grabadora de macros te permite grabar las acciones deseadas que posteriormente las traduce a instrucciones en VBA, las cuales podemos modificar posteriormente si tenemos conocimientos de programación.


Para grabar una macro debemos acceder a la pestaña **Vista** y despliega el submenú **Macros** y dentro de este submenú seleccionar la opción **Grabar macro...** Además de esta opción en el menú podemos encontrar las siguientes opciones:

**Ver Macros...** Donde accedemos a un listado de las macros creadas en ese libro.

**Usar referencias relativas** - Con esta opción utilizaremos referencias relativas para que las macros se graben con acciones relativas a la celda inicial seleccionada.

Antes de realizar una Macro es muy importante hablar sobre sus seguridades.

Se puede grabar las macros desde la ficha **Programador**, si no está disponible, haga lo siguiente para mostrarla:

Haga clic en el Botón Microsoft Office  y, a continuación, haga clic en

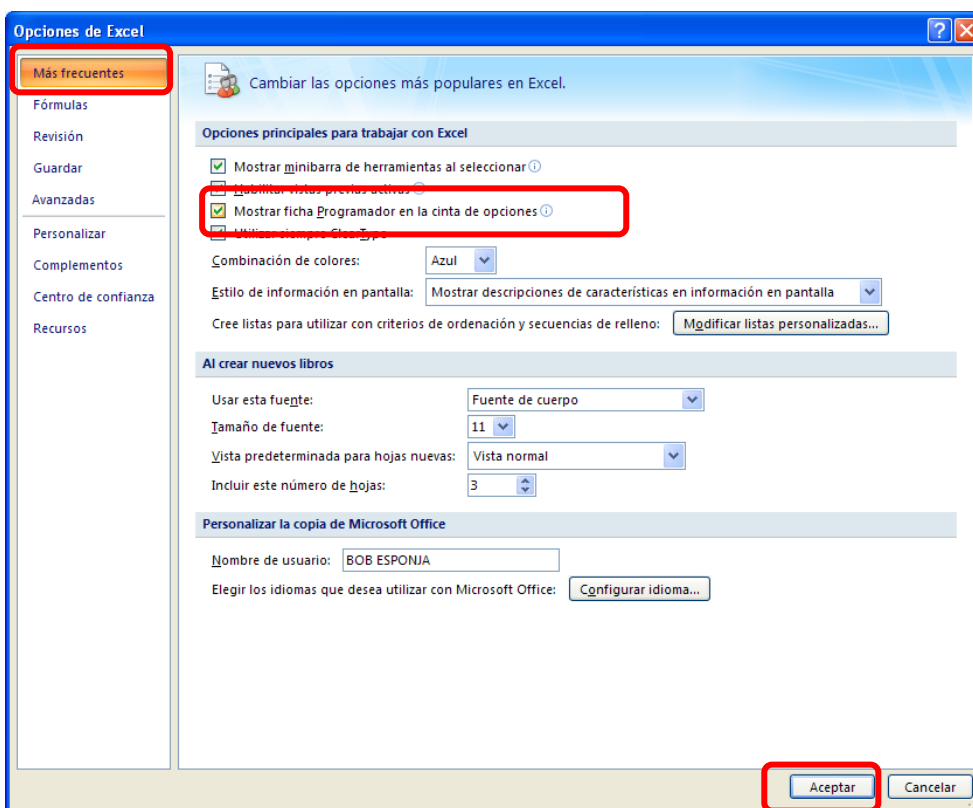
## Opciones de Excel.

Opción **Más frecuente**

Más frecuentes

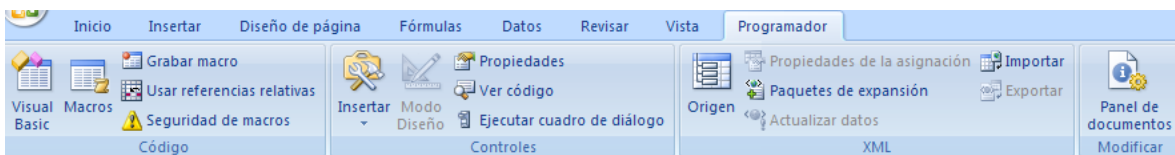
Active con un visto la opción  Mostrar ficha Programador en la cinta de opciones

Como se visualiza:

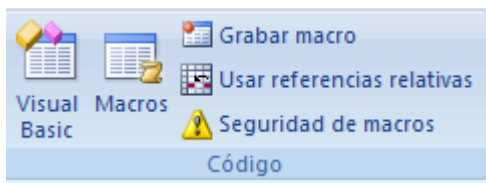


De clic en el Botón Aceptar.

Se visualiza la pestaña Programador que contiene:

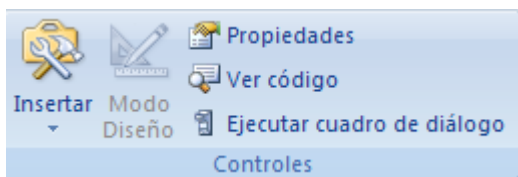


El grupo Código que se compone de:

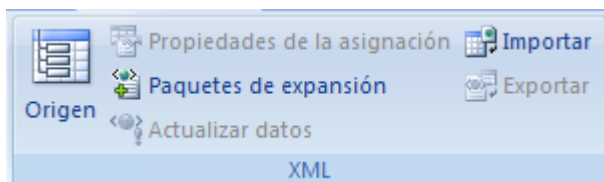


### Visual Basic y Macros

#### El grupo Controles



#### El grupo XML



#### El grupo Modificar



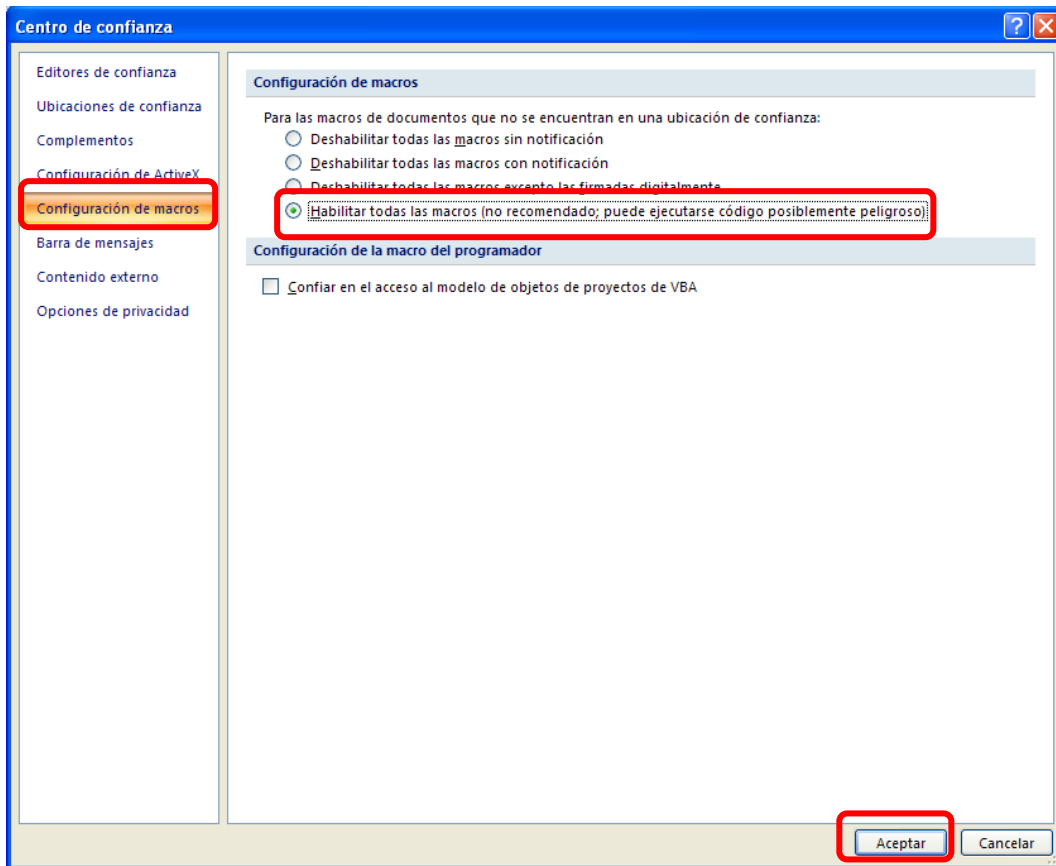
## SEGURIDAD EN MACROS

Para establecer el nivel de seguridad de manera que estén habilitadas temporalmente todas las macros, haga lo siguiente:

En la ficha **Programador**, en el grupo **Código**, haga clic en **Seguridad de macros**.



Se visualiza:



En **Configuración de macros**, haga clic en **Habilitar todas las macros** (no recomendado; puede ejecutarse código posiblemente peligroso) y, a continuación, haga clic en **Aceptar**.

**Nota** Para ayudar a evitar que se ejecute código potencialmente peligroso, recomendamos que vuelva a cualquiera de las configuraciones que deshabilitan todas las macros cuando termine de trabajar con las macros.

En la ficha **Programador**, en el grupo **Código**, haga clic en **Grabar macro**. En el cuadro **Nombre de la macro**, escriba un nombre para la macro.

**Nota** El primer carácter del nombre de la macro debe ser una letra. Los caracteres siguientes pueden ser letras, números o caracteres de subrayado. No se permiten *espacios en un nombre de macro, caracteres especiales ni palabras reservadas*; puede utilizarse un carácter de subrayado como separador de palabras.

Si utiliza un nombre de macro que también es una referencia de celda, puede aparecer un mensaje indicando que el nombre de la macro no es válido.

Para asignar una combinación de tecla de método abreviado (método abreviado: tecla o combinación de teclas de función, como F5 o CTRL+a, que utiliza para ejecutar un comando. Una tecla de acceso, por lo contrario es un combinación de teclas, como ALT+f, que mueve el enfoque a un menú, comando o control.) Con CTRL para ejecutar la macro, en el cuadro Tecla de método abreviado, escriba cualquier letra en mayúsculas o minúsculas que desee utilizar.

**Nota** La tecla de método abreviado suplantarán a cualquier tecla de método abreviado predeterminada equivalente en Excel mientras esté abierto el libro que contiene la macro.

En la lista **Guardar macro** en, seleccione el libro en el que desea almacenar la macro.

Sugerencia Si desea que la macro esté disponible siempre que utilice Excel, seleccione **Libro de macros personal**. Cuando se selecciona Libro de macros personal, Excel crea un libro oculto de macros personal (Personal.xlsm), si no existe todavía, y guarda la macro en este libro. En Microsoft Windows XP, este libro se guarda en la carpeta C:\Documents and Settings\nombre de usuario\Datos de programa\Microsoft\Excel\XLStart para que se pueda cargar automáticamente cada vez que se inicia Excel. En Microsoft Windows Vista, este libro se guarda en la carpeta C:\Usuarios\nombre de usuario\Datos de programa\Microsoft\Excel\XLStart.

Si desea que se ejecute automáticamente una macro del libro de macros personal en otro libro, también debe guardar ese libro en la carpeta XLStart, de forma que ambos libros se abran cuando se inicie Excel.

1. Para incluir una descripción de la macro, escriba el texto que desee en el cuadro Descripción.



2. Haga clic en **Aceptar** para iniciar la grabación.
3. Realice las acciones que desee grabar.
4. En la ficha **Programador**, en el grupo **Código**, haga clic en **Detener grabación**.

Sugerencia También puede hacer clic en Detener grabación en el lado izquierdo de la barra de estado.

### Practica I

Genera las siguientes Macros:

Grabe una **Macro** que se active con **Control + b** y que esta macro permita abrir un archivo.

Grabe una **Macro** que inserte una tabla con datos.

Grabe una Macro que abra un archivo existente.

Grabe una Macro que abra un nuevo archivo.

Grabe una Macro que inserte un logotipo.

Grabe una Macro que ordene alfabéticamente una lista de nombres.

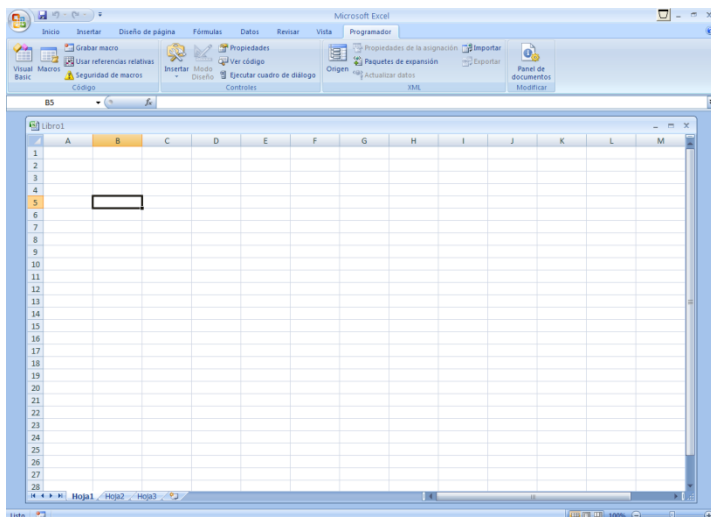
Grabe una Macro que imprima un formulario.

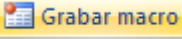
Nota: Recuerde que en la versión 2007 se debe guardar como un archivo de MS Excel habilitado para macros, es decir con la extensión .xlsm; caso contrario no guarda el código de las macros.

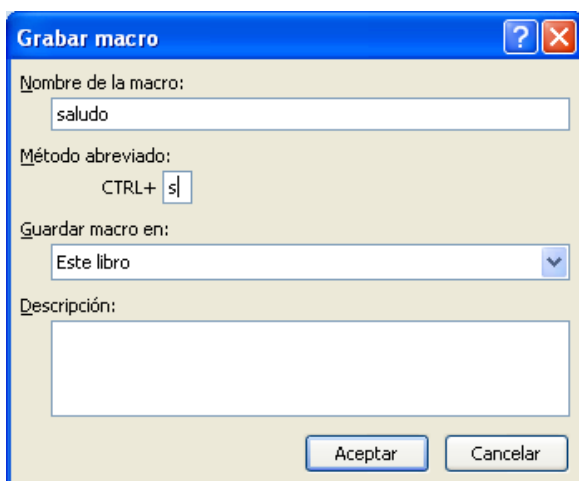
### CODIGOS DE UNA MACRO DE EXCEL

Para observar los códigos de una macro debemos seguir los pasos:

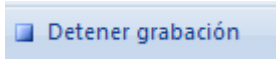
1. En primer lugar seleccione la celda **B5** antes de empezar la grabación de la Macro, se visualiza:

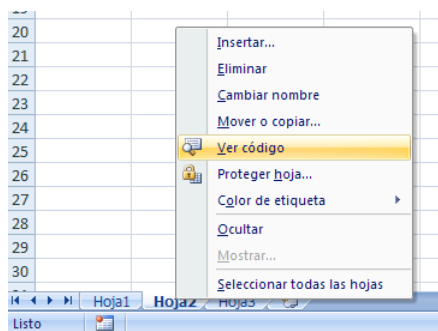


2. Presione el Botón **Grabar Macro**  del grupo **Código** MS Excel muestra el cuadro de Dialogo Grabar Macro:

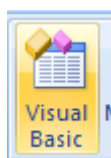


3. Ingrese un nombre de la macro por ejemplo **saludo**
4. En la opción **Método Abreviado** escriba la letra **s**, por lo tanto la macro se llamara con **Control + s**
5. En Guardar macro en: Seleccione en el lugar en donde desea guardar la macro, por ejemplo **Este libro**.

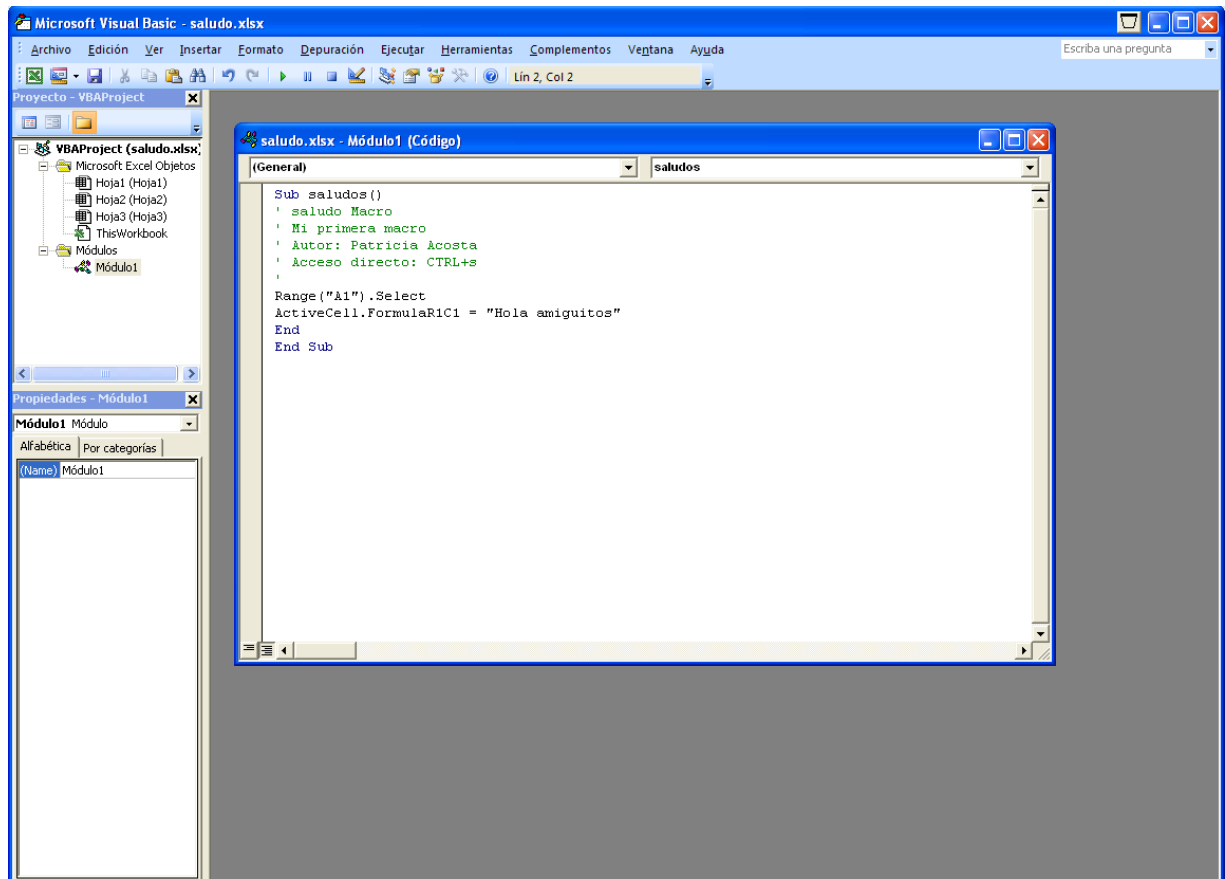
6. En Descripción puede agregar una descripción de lo que hace la macro, este punto es opcional. Solo le sirve para que usted recuerde acerca de lo que hace la macro, pues este código no es interpretado por el compilador.
7. Presione el botón **Aceptar**. Excel inicia la grabación del la Macro
8. Trasládese a la celda **A1** y escriba **Hola amiguitos**, después presione Enter para aceptar el valor en la celda.
9. Pare la grabación de la macro presionando el botón  **Detener Grabación** del grupo **Código**. Excel ha grabado los pasos y ha generado un código.
10. Para visualizar el código generado, presione la tecla Alt + la tecla de función F11(Alt + F11), o de un clic derecho en la hoja de cálculo:



11. Seleccione la opción Ver código. También puede acceder al grupo **Código**, al dar clic en la opción **Visual Basic**



12. Excel nos traslada al Editor de Visual Basic. Se visualiza:



13. Active los siguientes cuadros o ventanas:

- De clic en el Menú **Ver** y elija la opción **Explorador de Proyectos**
- De clic en el Menú **Ver** y elija la opción **Ventana Propiedades**

14. Del cuadro **Proyecto** de doble clic en **Módulos** o simplemente presione el signo de + que aparece en la opción **Módulos**. Se activara debajo de Módulos la Opción **Modulo1**.

**15.** De doble clic en **Modulo1**. Se mostrara en el Editor de Visual Basic el código de la macro que grabamos de la siguiente forma:

***Sub saludo()***

***" saludo Macro***

***' Mi primera macro***

***' Autor: Patricia Acosta***

***' Acceso directo: CTRL+s***

***'Range("A1").Select***

***ActiveCell.FormulaR1C1 = "Hola amiguitos"***

***End Sub***

16. Que es lo que significa esto nos preguntaremos asombrados, a continuación se da una explicación de lo que ha hecho **Excel**:

- **Sub** y **End Sub** indican el inicio y el final del procedimiento de la macro **saludo**
- Todo lo que aparece con un apóstrofe ´ indica que no se tomara en cuenta que es solo texto o comentarios y ese texto aparece en color verde.
- **Range("A1").Select** Indica que lo primero que hicimos al grabar la macro fue trasladarnos a la celda **A1**. La orden **Range** nos permite trasladarnos a una celda.
- **ActiveCell.FormulaR1C1 = "Hola amiguitos"** Esto indica que se escribirá en la celda en que se encuentra el valor de texto **Hola amiguitos**. Todo lo que aparece entre comillas siempre será un valor de texto. La orden **ActiveCell.FormulaR1C1** nos permite escribir un valor en la celda activa. Para comprender alteraremos el código dentro del editor de Visual Basic.

***Sub saludo()***

***' saludo Macro***

***' Mi primera macro***

***' Autor: Patricia Acosta***

***' Acceso directo: CTRL+s***

***Range("A1").Select***

***ActiveCell.FormulaR1C1 = "Hola amiguitos"***

***Range("B1").Select***

***ActiveCell.FormulaR1C1 = "Bienvenidos al curso de Excel"***

***End Sub***

17. Al alterar el código y cuando regrese a **Excel** y ejecute la macro con **Control + s** hará lo siguiente:

En A1 escribirá ***Hola amiguitos***

En B1 escribirá ***Bienvenidos al curso de Excel***

Al alterar el código y cuando regrese a **Excel** y ejecute la macro con **Control + s** hará

En A1 escribirá ***Hola amiguitos***

En B1 escribirá ***Bienvenidos al seminario de Excel.***

Se visualiza:

```
Sub saludos()
```

```
' saludo Macro
```

```
' Mi primera macro
```

```
' Autor: Patricia Acosta
```

```
' Acceso directo: CTRL+s
```

```
,
```

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

```
ActiveCell.FormulaR1C1 = "Hola amiguitos"
```

```
,
```

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

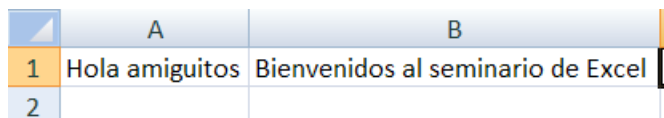
```
ActiveCell.FormulaR1C1 = "Bienvenidos al seminario de Excel"
```

```
End
```

```
End Sub
```

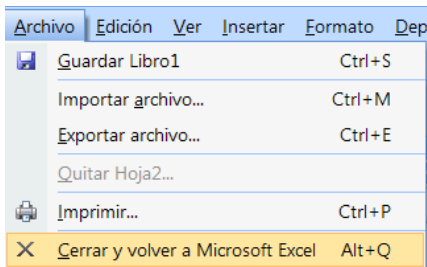
Al alterar el código y cuando regrese a **Excel** y ejecute la macro con **Control + s** hará:


En A1 escribirá ***Hola amiguitos.*** En B1 escribirá ***Bienvenidos al seminario de Excel.***

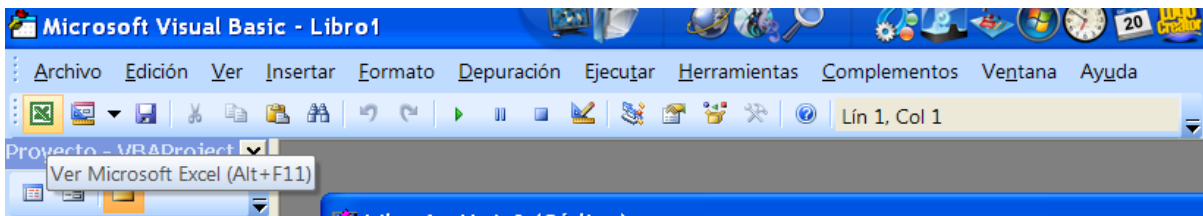


	A	B
1	Hola amiguitos	Bienvenidos al seminario de Excel
2		

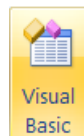
Para salir del editor de clic en el **Menú Archivo** y elija la opción **Cerrar y volver a Microsoft Excel.**

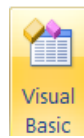


Si no desea salir por completo de clic en el **botón Microsoft Excel**  que se encuentra activado en:



Cuando desee volver al editor de clic en: la pestaña Programador



De clic en el icono  del grupo Código

### Practica II

- Genere una **Macro** que escriba un nombre en una celda y lo ponga negrita y observe el **Código**.
- Genere una **Macro** que escriba un nombre en una celda y lo Centre y observe el **Código**.
- Genere una **Macro** que escriba un nombre en una celda y cambie el tamaño de la letra a 20 puntos y observa el **Código**.

## CÓDIGOS MÁS COMUNES

### Trasladarse a una Celda

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

### Escribir en una Celda

```
Activecell.FormulaR1C1="Paty Acosta"
```

### Letra Negrita

```
Selection.Font.Bold = True
```

### Letra Cursiva

```
Selection.Font.Italic = True
```

### Letra Subrayada

```
Selection.Font.Underline = xlUnderlineStyleSingle
```

### Centrar Texto

```
With Selection
```

```
    .HorizontalAlignment = xlCenter
```

```
End With
```

### Alinear a la izquierda

```
With Selection
```

```
    .HorizontalAlignment = xlLeft
```

```
End With
```

### Alinear a la Derecha

```
With Selection
```

```
    .HorizontalAlignment = xlRight
```



End With

### **Tipo de Letra(Fuente)**

With Selection .Font

.Name = "AGaramond"

End With

### **Tamaño de Letra(Tamaño de Fuente)**

With Selection.Font

.Size = 15

End With

### **Copiar**

Selection.Copy

### **Pegar**

ActiveSheet.Paste

### **Cortar**

Selection.Cut

### **Ordenar Ascendente**

Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Header:=xlGuess, \_

OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom

### **Orden Descendente**

Selection.Sort Key1:=Range("A1"), Order1:=xlDescending, Header:=xlGuess, \_

OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom

### **Buscar**

```
Cells.Find(What:="Paty Acosta", After:=ActiveCell, LookIn:=xlFormulas, LookAt _  
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _  
False).Activate
```

### **Insertar Fila**

```
Selection.EntireRow.Insert
```

### **Eliminar Fila**

```
Selection.EntireRow.Delete
```

### **Insertar Columna**

```
Selection.EntireColumn.Insert
```

### **Eliminar Columna**

```
Selection.EntireColumn.Delete
```

### **Abrir un Libro**

```
Workbooks.Open Filename:="C:\Mis documentos\miarchivo.xls"
```

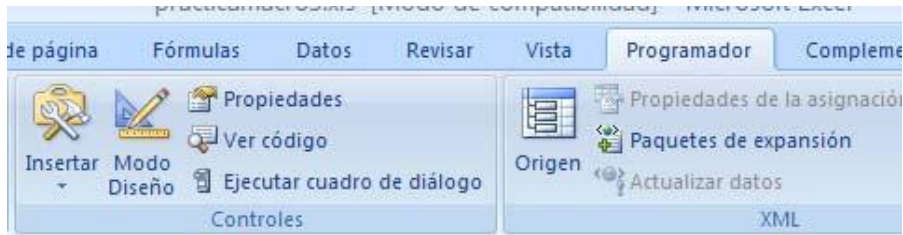
### **Grabar un Libro**

```
ActiveWorkbook.SaveAs Filename:="C:\Mis documentos\tauro.xls", FileFormat _  
:=xlNormal, Password:="", WriteResPassword:="", ReadOnlyRecommended:= _  
False, CreateBackup:=False
```

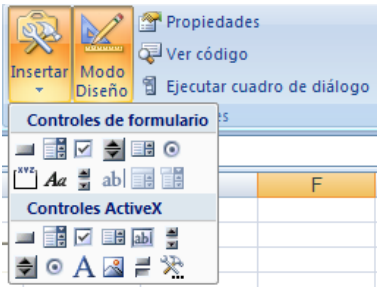
## **CUADRO DE CONTROL – CONTROLES ACTIVEX**

Una de las opciones más interesantes que tiene el Excel es la de utilizar los “cuadros de control”. Los cuadros de control se usan para crear verdaderos programas en Excel y pueden ser de mucha utilidad.

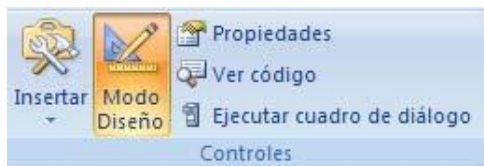
Esta herramienta está ubicada en:



En Excel 2007 se encuentra el grupo Controles de la pestaña Programador

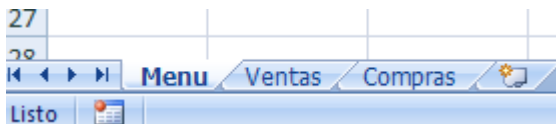


Este grupo de Controles cuenta con tres opciones muy importantes como:



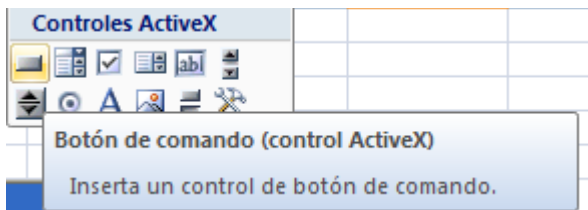
1. Modo diseño: permitirá trabajar en el diseño de los controles de ActiveX
2. Propiedades: permiten activar la propiedad de cada control
3. Ver código: permite agregar código a cada control.

Para iniciar cree las hojas: Menú, Ventas y Compras



Seleccione la hoja Menú para allí crear dos botones.

Para trabajar con estos controles es necesario Activar el modo de diseño y dar clic en Insertar, seleccione el Botón de comando.

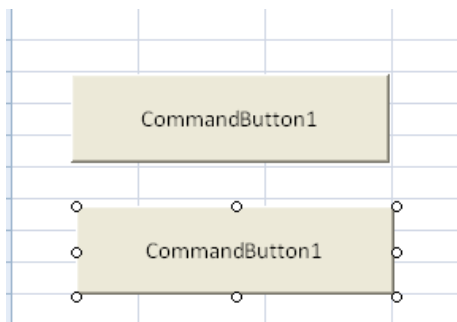


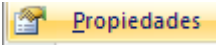
Esta vez haremos un botón que cuando se presione pase a otra hoja del Excel. Por ejemplo se puede hacer un menú con varios botones que al presionarlos pasen a las distintas opciones.

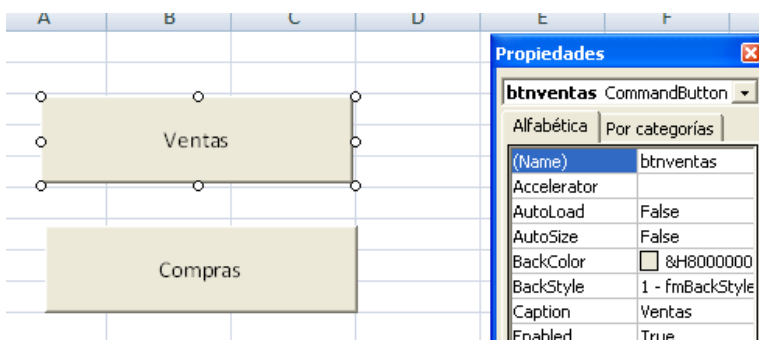
Comencemos...

En la hoja Menú cree dos “botones de comando”.

Por Ejemplo:



Seleccione el primer botón y de un clic derecho en la opción y  muestre las propiedades. Cambie la Propiedad “Caption” por: “Ventas” En Name: btnventas

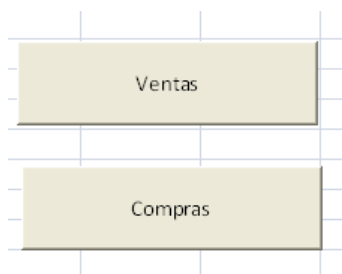


Seleccione el segundo botón y muestre las propiedades

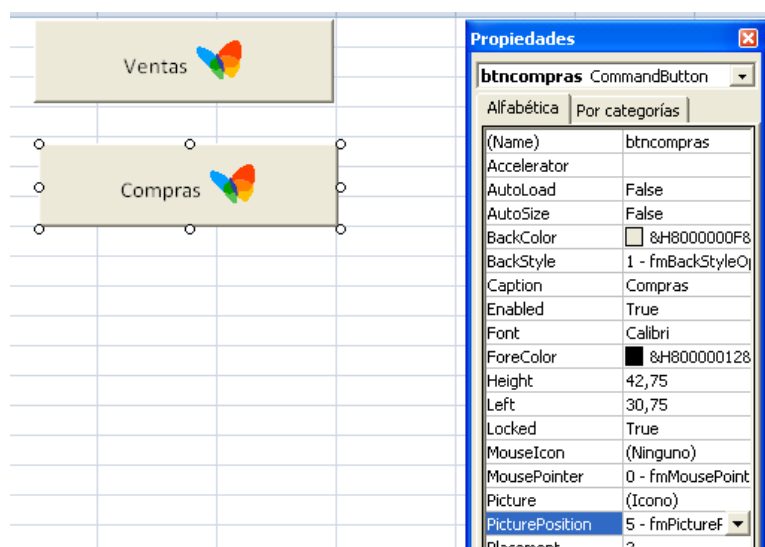
Cambie la Propiedad "Caption" por: "Compras"

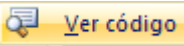
En Name: btncompras

Si realizó bien estos pasos Ud. Debería ver lo siguiente:



Si desea colocar iconos en los botones seleccione la propiedad Picture e inserte una imagen de extensión .ico.



Para que visualice el texto cambie la posición PicturePosition a: 5 Seleccione el primer botón y haga clic en ver código 

En esta parte se abrirá el Editor de Visual Basic y debe escribir lo siguiente:

**Hoja2.Activate**

Cierre el editor de Visual Basic (nota: cada vez que cierre el editor de Visual Basic, hágalo del cuadro de cerrar “X” que está mas arriba, porque puede confundirse y cerrar la ventana de editar código, no se preocupe que no está cerrando Excel.)

Seleccione el segundo botón y haga clic en ver código 

Escriba: **Hoja3.activate**

Salga del modo de diseño y navegue con los botones que programó.

Más adelante utilizaremos estos botones para cargar formularios desde VBA en Excel.

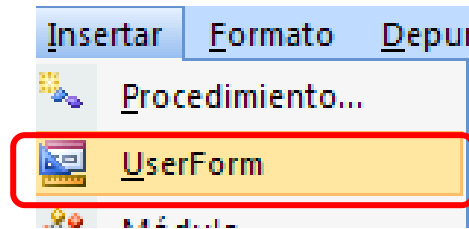
O añada el código de las macros que grabó con la grabadora.

### **CREANDO FORMULARIOS Y PROGRAMÁNDOLOS**

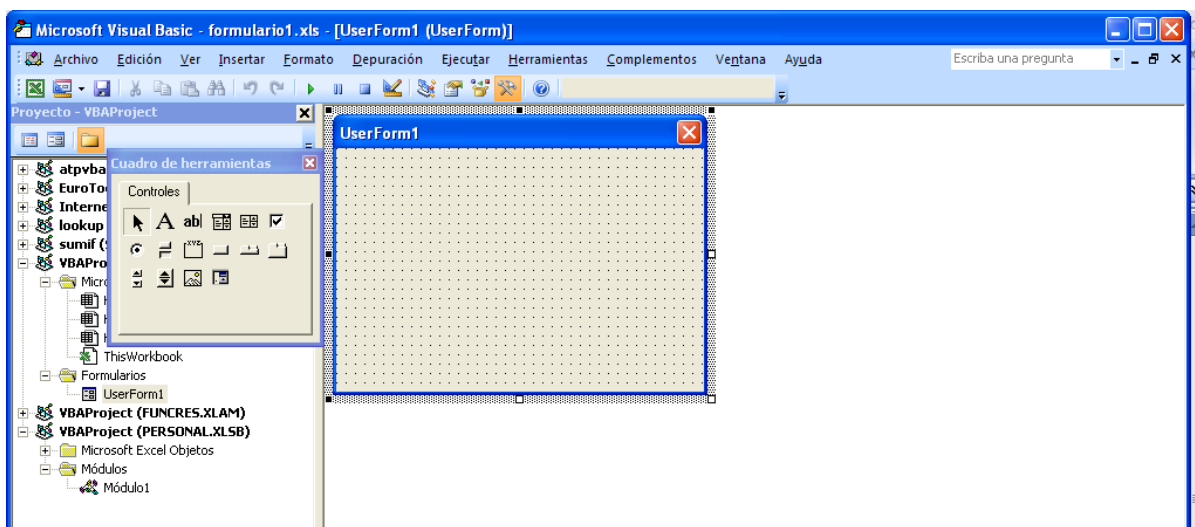
Ahora aprenderemos a dominar lo máximo de Excel que es crear formularios y programarlos, bueno un formulario es una ventana que se programa por medio de controles y estos controles responden a sucesos que nosotros programamos. Todo esto se encuentra dentro de Visual Basic.

A continuación Muestro como crear un formulario y como programarlo:

1. Presione La Teclas **Alt + F11**, para entrar al editor de **Visual Basic**.
2. Activa las siguientes opciones:
  - De clic en el **Menú Ver** y elija la opción **Explorador de Proyectos**
  - De clic en el **Menú ver** y elija la opción **Ventana Propiedades**
3. Del **Menú Insertar** elija la Opción **UserForm**.



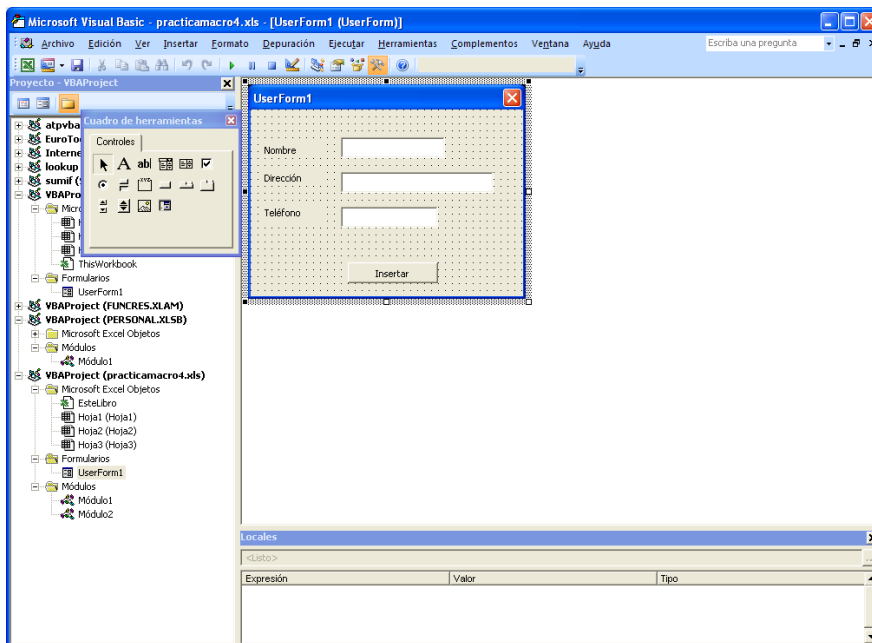
Esto inserta el Formulario que programaremos con controles. En el **Explorador de Proyecto** se observara que se inserto el **UserForm**.



También cuando de clic en el Formulario **USERFORM1** se debe de activar el **Cuadro de Herramientas**, si no se activa de clic en el **Menú Ver** y elija la opción **Cuadro de Herramientas**.

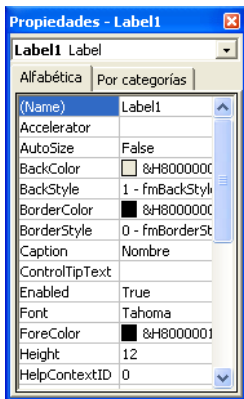
4. Elija del **Cuadro de Herramientas** el Control **Etiqueta** el que tiene la **A** y Arrastre dibujando en el Formulario **USERFORM1** la etiqueta. Quedara el nombre Label1, después de un clic en la etiqueta dibujada y podrá modificar el nombre de adentro y pondremos ahí **Nombre**. Si por error da doble clic en la etiqueta y lo manda a la pantalla de programación de la etiqueta, solo de doble clic en **UserForm1** que se encuentra en el **Explorador de Proyecto**.

5. Elija del **Cuadro de Herramientas** el control **Cuadro de Texto** el que tiene **ab** y arrastre dibujando en el formulario **USERFORM1** el cuadro de texto a un lado de la etiqueta que dice **Nombre**. El cuadro de texto debe de estar vacío y su nombre será **Textbox1**, el nombre solo aparecerá en el control.
6. Haga los dos pasos anteriores igualmente poniendo **Dirección** en la **Label2** y **Teléfono** en la **Label3** y también dibújeles su Textbox. Esto quedara así después de haberlo hecho.



*Si tiene algún problema* al dibujar las etiquetas o los cuadros de texto, solo cámbiele el nombre a la etiqueta o el cuadro de texto en la **Ventana Propiedades** la opción se llama **(Name)**. El Error que marque puede ser **Nombre Ambiguo**, pero si le cambia el Nombre al control se quitara el error. Puede ponerle cualquier nombre en lugar de Label1.



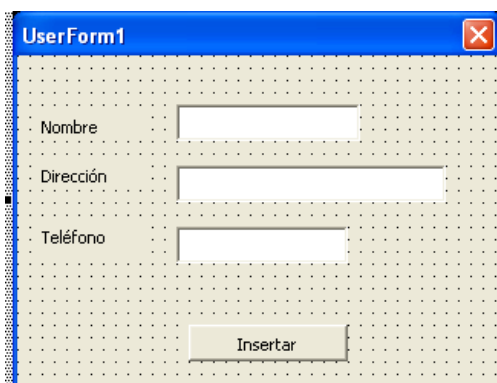


Los controles como las Etiquetas y Cuadros de Textos pueden modificárseles algunas opciones en la Ventana Propiedades Para hacer esto es necesario tener conocimiento sobre las propiedades de los controles. No altere las propiedades si no las conoce.

7. Elija del **Cuadro de Herramientas** el control **Botón de Comando** y Arrastre dibujando en el Formulario **USERFORM1** el Botón, después de un clic en el nombre del Botón dibujado y podrá modificar el nombre y pondremos ahí **Insertar**.

Si por error da doble clic en la Botón y lo manda a la pantalla de programación de la etiqueta, solo de doble clic en **UserForm1** que se encuentra en el **Explorador de Proyecto**.

Así quedara el Formulario formado por los controles:



8. Ahora de doble clic sobre el control **Textbox1** para programarlo y después inserte el siguiente código:

```
Private Sub TextBox1_Change()
```

```
Range("A9").Select
```

```
ActiveCell.FormulaR1C1 = TextBox1
```

```
End Sub
```

Esto indica que se vaya a **A9** y escriba lo que hay en el **Textbox1**

**Nota.**-Lo que esta en azul lo genera Excel automáticamente, usted solo escribirá lo que esta en Negrita.

Para volver al **Formulario** y programar el siguiente Textbox de doble clic en **UserForm1** que se encuentra en el **Explorador de Proyecto**, o simplemente de clic en **Ver Objeto** en el mismo **Explorador de Proyecto**.

9. Ahora de doble clic sobre el control **Textbox2** para programarlo y después inserte el siguiente código:

```
Private Sub TextBox2_Change()
```

```
Range("B9").Select
```

```
ActiveCell.FormulaR1C1 = TextBox2
```

```
End Sub
```

Esto indica que se vaya a **B9** y escriba lo que hay en el **Textbox2**.

Para volver al **Formulario** y programar el siguiente Textbox de doble clic en **UserForm1** que se encuentra en el **Explorador de Proyecto**, o simplemente de clic en **Ver Objeto** en el mismo **Explorador de Proyecto**.

10. Ahora de doble clic sobre el control **Textbox3** para programarlo y después inserte el siguiente código:

```
Private Sub TextBox3_Change()
```

```
Range("C9").Select
```

```
ActiveCell.FormulaR1C1 = TextBox2
```

```
End Sub
```

Esto indica que se valla a **C9** y escriba lo que hay en el **Textbox3**

Para volver al **Formulario** y programar el **Botón de Comando *Insertar*** de doble clic en **UserForm1** que se encuentra en el **Explorador de Proyecto**, o simplemente de clic en **Ver Objeto** en el mismo **Explorador de Proyecto**.

11. Ahora de doble clic sobre el control **Botón de Comando** para programarlo y después inserte el siguiente código:

```
Private Sub CommandButton1_Click()
```

```
'inserta un renglón
```

```
Selection.EntireRow.Insert
```

```
'Empty Limpia Los Textbox
```

```
TextBox1 = Empty
```

```
TextBox2 = Empty
```

```
TextBox3 = Empty
```

```
'Textbox1.SetFocus Envía el cursor al Textbox1 para volver a capturar los
```

```
datos
```

```
TextBox1.SetFocus
```

End Sub

**Nota.**-El comando **Rem** es empleado para poner comentarios dentro de la programación, el comando **Empty** es empleado para vaciar los Textbox.

12. Ahora presione el botón **Ejecutar User/Form** que se encuentra en la barra de herramientas o simplemente la tecla de función **F5**.

Se activará el **Userform1** y todo lo que escriba en los Textbox se escribirá en Excel y cuando presione el botón Insertar, se insertara un renglón y se vaciaran los Textbox y después se mostrara el cursor en el **Textbox1**.

### TRABAJANDO CON FORMULAS

Es de suma importancia saber aplicar **Formulas** en **Macros de Excel**, ya que la mayoría de las hojas de cálculos las involucran, por ejemplo los Inventarios, las Nominas o cualquier otro tipo de hoja las llevan, es por eso que en la siguiente **Fase** se muestra como manejar **Formulas** en **Macros de Excel**.

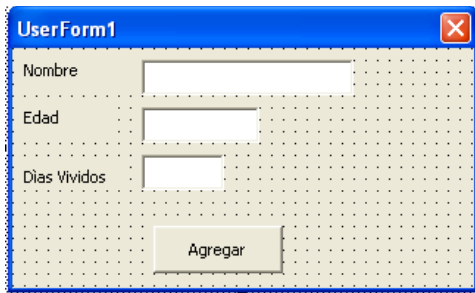
Presione La Teclas **Alt + F11**, para entrar al editor de **Visual Basic**.

Activa las siguientes opciones:

- De clic en el **Menú Ver** y elija la opción **Explorador de Proyectos**
- De clic en el **Menú ver** y elija la opción **Ventana Propiedades**

Del **Menú Insertar** elija la Opción **UserForm**. Esto inserta el Formulario que programaremos con controles. En el **Explorador de Proyecto** se observara que se inserto el **UserForm**.

Ahora crearas un formulario con el siguiente aspecto:



El formulario tendrá:

- Tres etiquetas
- Tres Textbox
- Un Botón de Comando

Los datos que se preguntaran serán Nombre y Edad, los Días Vividos se generaran automáticamente cuando insertes la edad. A continuación se muestra como se deben de programar estos Controles

Programación de los Controles:

```
Private Sub CommandButton1_Click()
```

```
Selection.EntireRow.Insert
```

```
TextBox1 = Empty
```

```
TextBox2 = Empty
```

```
TextBox3 = Empty
```

```
TextBox1.SetFocus
```

```
End Sub
```

```
Private Sub TextBox1_Change()
```

```
Range("A9").Select
```

```
ActiveCell.FormulaR1C1 = TextBox1
```

```
End Sub
```

```
Private Sub TextBox2_Change()
```

```
Range("B9").Select
```

```
ActiveCell.FormulaR1C1 = TextBox2
```

' aquí se crea la Fórmula

```
TextBox3 = Val(TextBox2) * 365
```

'El Textbox3 guardara el total de la multiplicación del Textbox2 por 365

'El Comando Val permite convertir un valor de Texto a un Valor Numérico

'Esto se debe a que los Textbox no son Numéricos y debemos de Convertirlos

End Sub

```
Private Sub TextBox3_Change()
```

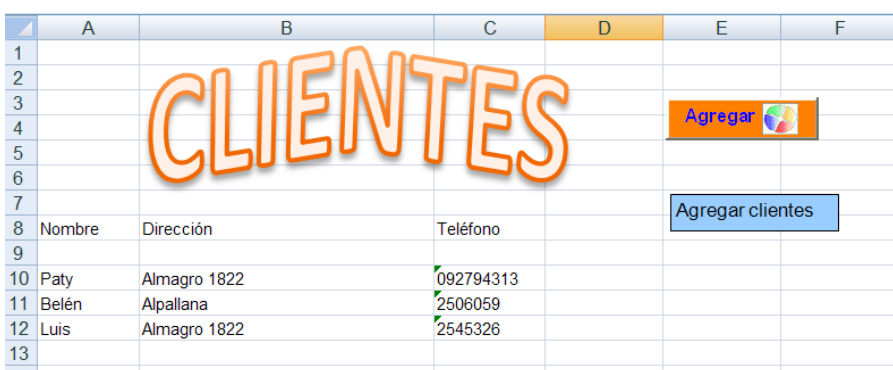
```
Range("C9").Select
```

```
ActiveCell.FormulaR1C1 = TextBox3
```

End Sub

## CÓDIGO PARA CARGAR UN FORMULARIO DESDE EXCEL

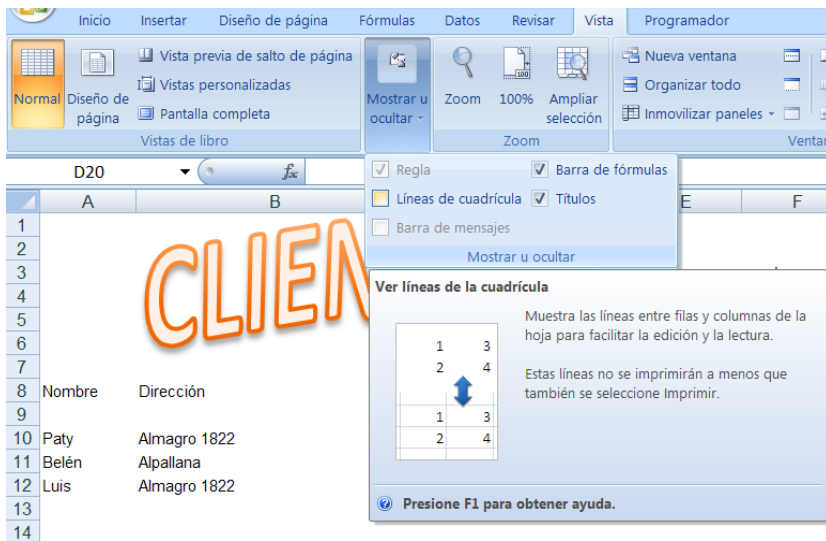
Una vez que haya diseñado su formulario en el ambiente de Visual Basic Application, se requiere que este sea cargado desde MS Excel, para esto diseñe una interfaz por ejemplo:



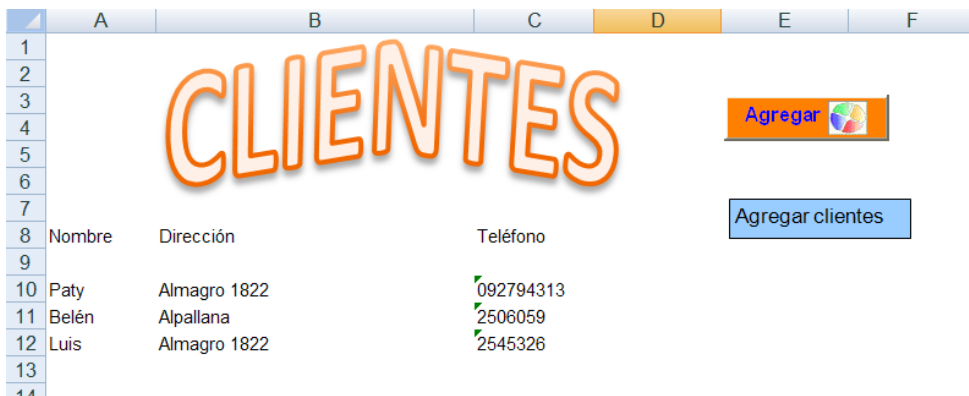
	A	B	C	D	E	F
1						
2						
3						
4					Agregar	
5						
6						
7					Agregar clientes	
8	Nombre	Dirección	Teléfono			
9						
10	Paty	Almagro 1822	092794313			
11	Belén	Alpallana	2506059			
12	Luis	Almagro 1822	2545326			
13						

Si desee quitar la cuadrícula seleccione en la pestaña **Vista** en el grupo **Zoom**, la opción **Mostrar un ocultar** y desactive (quitar con un clic el visto) la opción Líneas de cuadrícula  Líneas de cuadrícula


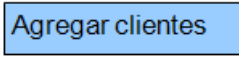
# MS Excel con Programación de Macros en Visual Basic Application

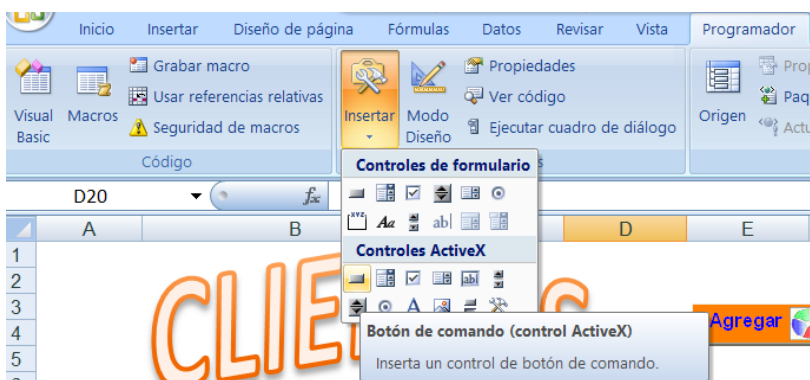


La interfaz lucirá así:

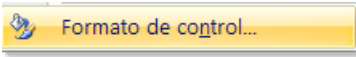


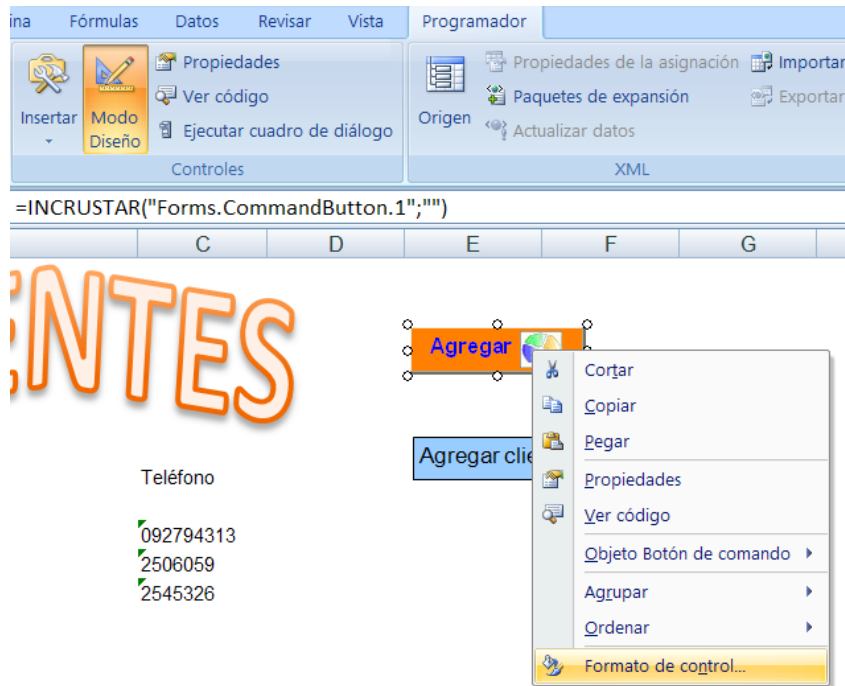
La interfaz debe contener un botón de comando de Controles ActiveX

 o una autoforma  para poder cargar el diseño del formulario



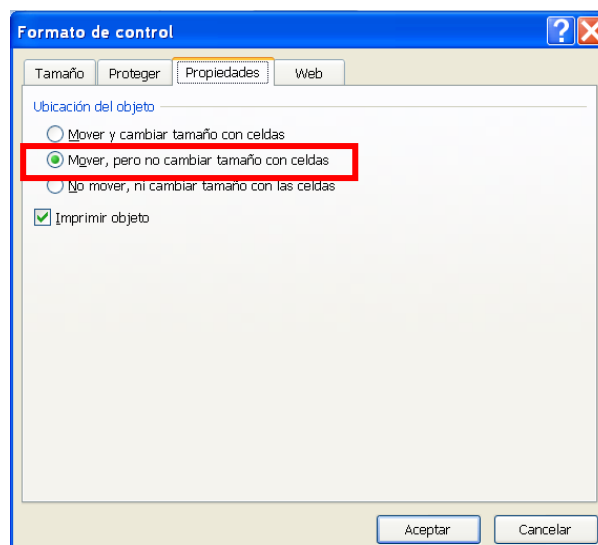
## MS Excel con Programación de Macros en Visual Basic Application

Para que el botón de comando se mantenga sin moverse es decir esté fijo y no se imprima, en el modo de Diseño, seleccione el botón de un clic derecho elija la opción Formato de control  como se visualiza en la imagen.




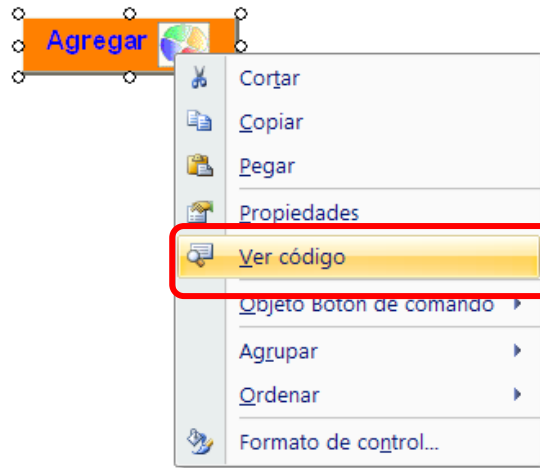
Se visualiza el siguiente cuadro de diálogo, seleccione la pestaña **Propiedades** y active con un punto la opción **No mover, ni cambiar tamaño con las celdas**.

Mover, pero no cambiar tamaño con celdas Para que no se imprima el botón desactive (quite el visto) de la opción Imprimir objeto.





Para programar el botón, de doble clic sobre el objeto o de un clic derecho sobre el objeto en la opción:  Ver código Como se observa en la imagen:



Se ingresa al ambiente de Visual Basic Application

```
Private Sub btnagregar_Click()
```

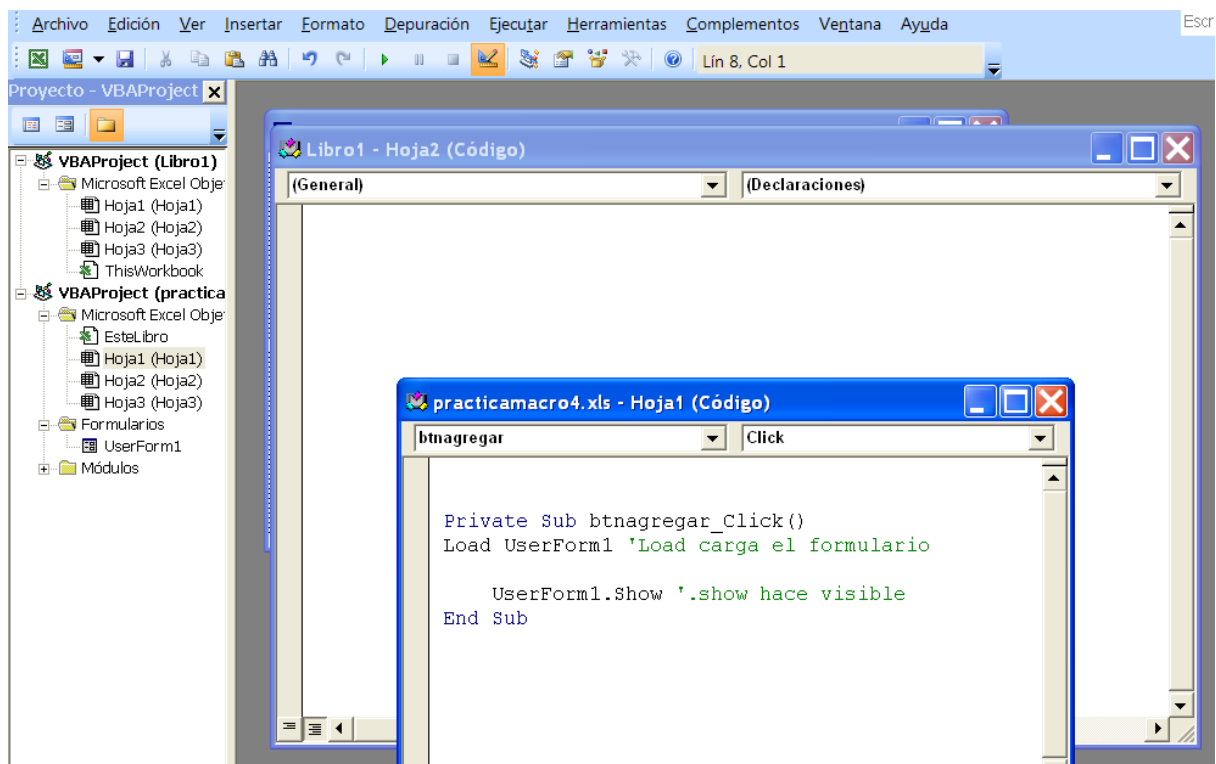
```
End Sub
```

En donde titila el cursor se digita el siguiente código:

```
Load UserForm1 'Load carga el formulario
```

```
UserForm1.Show '.show hace visible
```

En la imagen se visualiza:



### ASIGNAR UNA MACRO A UNA AUTOFORMA

El caso es similar al anterior para esto debe crear un módulo en el ambiente de Visual Basic Application y digitar el código:

**Sub Agregar()**

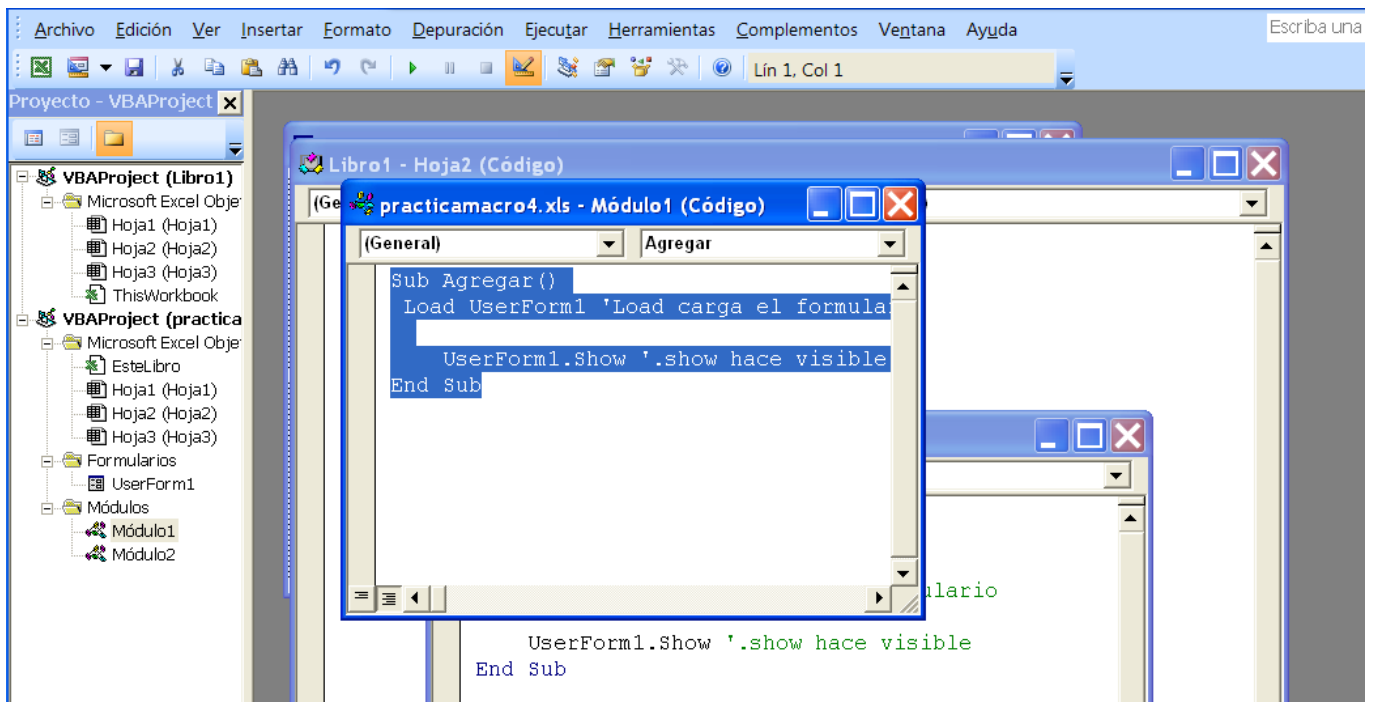
**Load UserForm1 'Load carga el formulario**

**UserForm1.Show '.show hace visible al formulario**

**End Sub**

Se observa en la imagen:

## MS Excel con Programación de Macros en Visual Basic Application



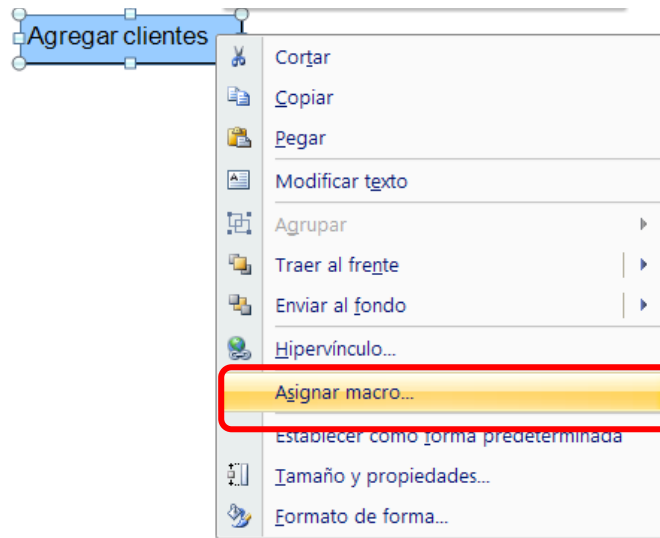
Una vez listo el módulo, se inserta una autoforma en el ambiente de MS Excel y se asigna la macro Agregar que creo en el módulo. Observe la siguiente imagen:

The screenshot shows an Excel spreadsheet with a table of customer data. The table has columns for 'Nombre', 'Dirección', and 'Teléfono'. The data rows are:

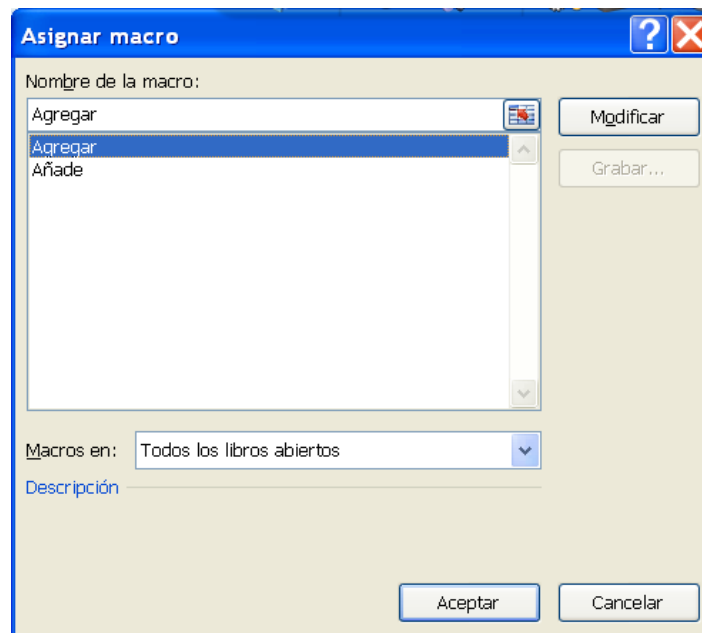
Nombre	Dirección	Teléfono
Paty	Almagro 1822	092794313
Belén	Alpallana	2506059
Luis	Almagro 1822	2545326

Below the table, there is a button labeled 'Agregar clientes' with a small icon above it.

Luego de un clic derecho sobre la autoforma seleccione la opción Asignar macro...



Se visualiza:



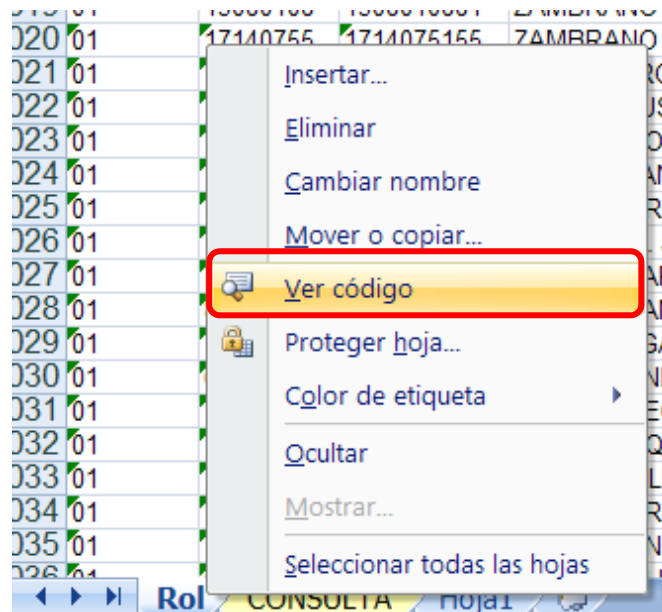
Seleccione la macro **Agregar**.

De clic en Aceptar.

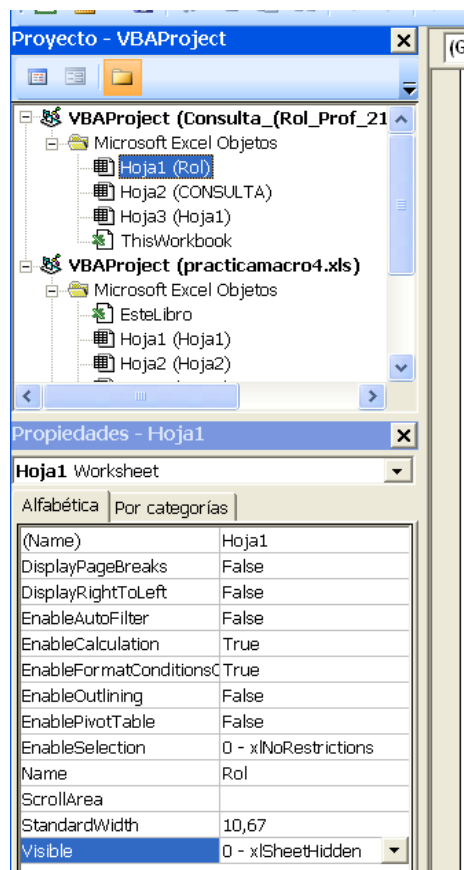
### PROTEGER UNA HOJA EN AMBIENTE VBA

Si desea proteger una hoja de Excel para que no se visualice sobretodo si contiene datos confidenciales y que sólo es necesario acceder para una consulta realice lo siguiente:

Seleccione la hoja a ocultar por ejemplo Rol y de un clic derecho en la opción Ver código



Se ingresa al ambiente de VBA en el que debe activar las propiedades de la hoja, se visualiza:



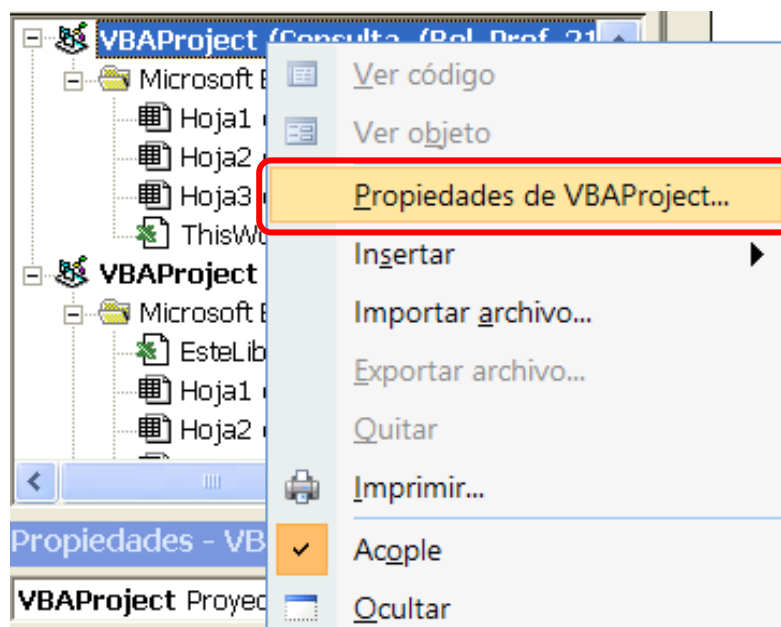
En la propiedad Visible seleccione: 0 – xlssheethidden

Pero para conseguir una protección más segura se sugiere colocar una clave al proyecto de VBA.

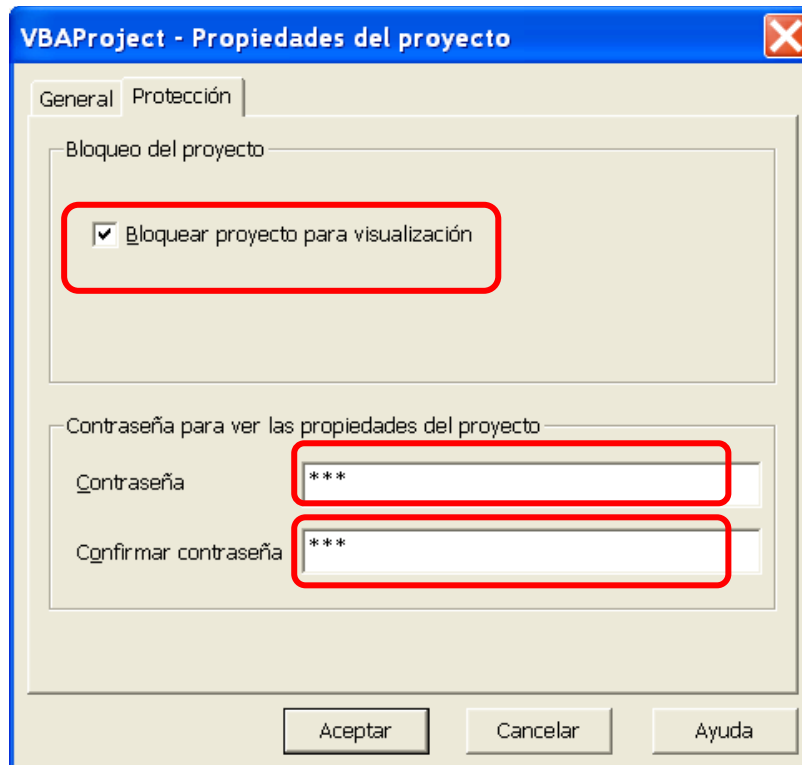
### COLOCAR UNA CLAVE AL PROYECTO DE VBA

Para esto en el mismo ambiente realice lo siguiente:

Seleccione el proyecto de un clic derecho sobre el proyecto y elija la opción Propiedades de VBAProject...



Se visualiza:



En el cuadro de diálogo seleccione la pestaña Protección para bloquear el proyecto active con un visto la opción Bloquear proyecto para visualización.

Y para finalizar coloque una contraseña.

De esta forma no podrán visualizar ni la hoja oculta ni el código de programación generado en la aplicación.

### ***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 abstracto que puede ser cualquier cosa. Por ejemplo podemos referirnos a objetos como coche, silla, casa, etc.

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.

## **CONCEPTOS QUE ENCONTRAREMOS EN EXCEL**

**WorkSheet** (Objeto hoja de cálculo)

**Range** (Objeto celda o rango de celdas).



Un objeto **Range** está definido por una clase donde se definen sus propiedades, recordemos que una propiedad es una característica, modificable o no, de un objeto. Entre las propiedades de un objeto **Range** están **Value**, que contiene el valor de la celda, **Column** y **Row** que contienen respectivamente la fila y la columna de la celda, **Font** que contiene la fuente de los caracteres que muestra la celda, 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 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

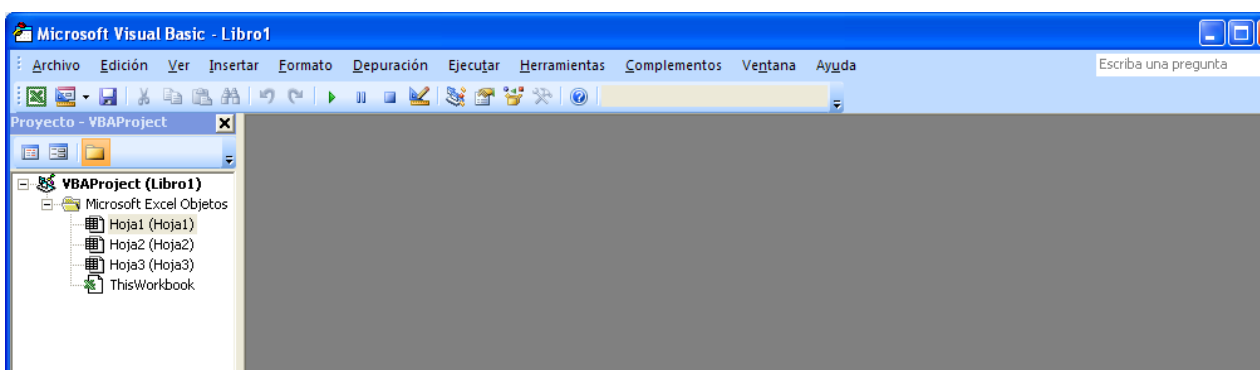
Propiedades de un objeto que serán también Objetos. Dicho de otra forma, hay propiedades que devuelven objetos, por ejemplo, la propiedad **Range** de un objeto **WorkSheet** devuelve un objeto de tipo **Range**.

### ***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.

### **Editor de Visual Basic.**

El editor de Visual Basic 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.



Maximice 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.***

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 celda A1.

### **Ejemplo**

**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 celda A1, encerramos entre paréntesis esta referencia. 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 celda A1 de la hoja activa, mientras que en el segundo es en la celda 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 siguiente manera:

**Sub Primero**

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

**End Sub**

Si desea poner "Hola" (o cualquier valor) en la celda activa, puede utilizar la propiedad (objeto) **Activecell** de **Worksheets**. Así para poner "Hola" en la celda 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 dependen 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.

1. 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 están todas las macros incluidas en el libro de trabajo.
2. Seleccione la macro de la lista y pulse sobre el botón **Ejecutar**.

### Ejemplo

En este ejemplo simplemente ampliaremos la funcionalidad de la macro del ejemplo

1. Además de escribir "Hola" en la celda A1, le 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 ***Celda\_Inicial:Celda\_Final***. Por ejemplo aplicar el último ejemplo al rango de celdas 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**

## OBSERVANDO LOS CÓDIGOS DE UNA MACRO DE EXCEL

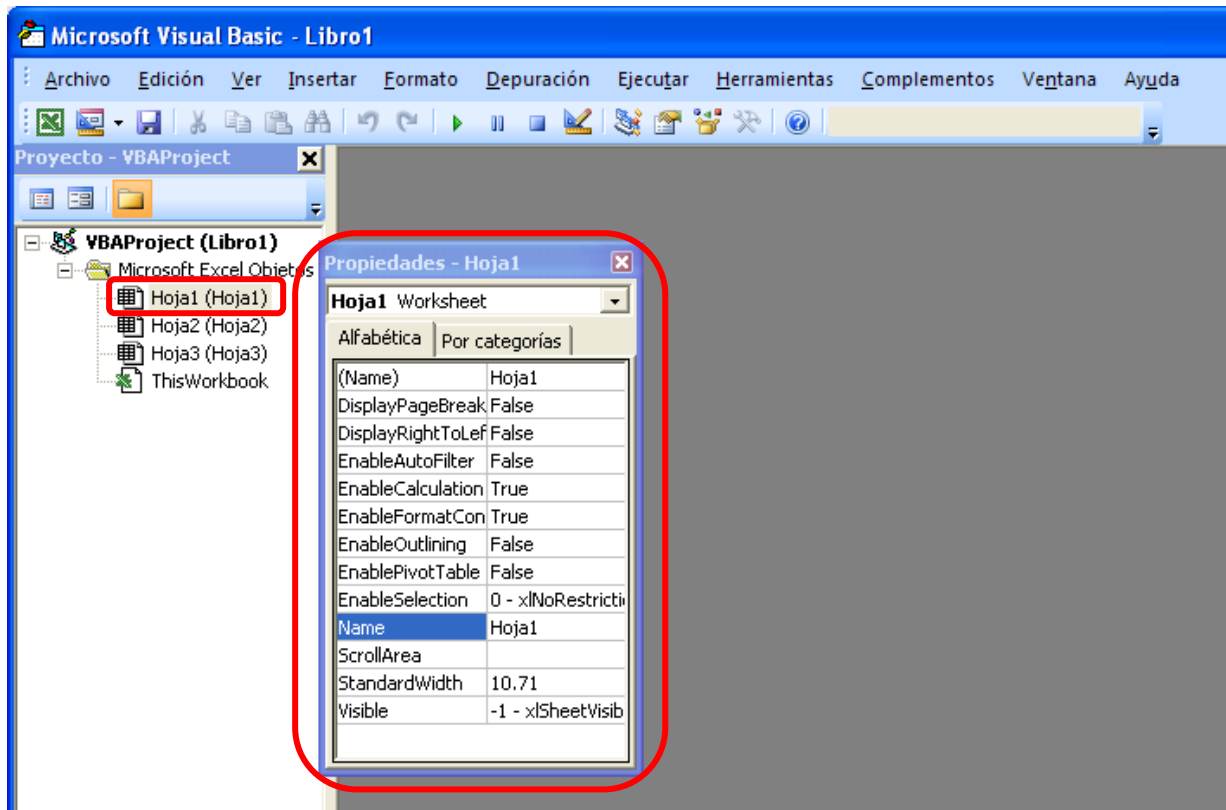
Crearemos una macro y veremos sus códigos:

Para observar los códigos de una macro debemos seguir los pasos:

1. Primeramente trasládese a la celda **A5** antes de empezar la grabación de la Macro
2. Presione el Botón **Grabar Macro** de la barra de Herramientas **Visual Basic**. **Excel** muestra el cuadro de Dialogo Grabar Macro
3. en la opción **Método Abreviado** escriba la letra **r**, por lo tanto la macro se llamara con **Control + r**
4. Presione el botón **Aceptar**. **Excel** inicia la grabación del la **Macro1**
5. Trasládese a la celda **A1** y escriba **su nombre**, después presione **Enter** para aceptar el valor en la celda
6. Pare la grabación de la macro presionando el botón **Detener Grabación** de la barra de herramientas **Visual Basic**. **Excel** a grabado los pasos y a generado un código, Observémoslos:
7. Presione la tecla **Alt + la tecla de función F11(Alt + F11)**. **Excel** nos traslada al Editor de Visual Basic. Si este editor no se activa es que **Excel** no esta bien instalado o se a borrado.
8. Active los siguientes cuadros o ventanas:
  - De clic en el **Menú Ver** y elija la opción **Explorador de Proyectos**
  - De clic en el **Menú ver** y elija la opción **Ventana Propiedades**

Estas dos opciones deben de estar siempre activadas ya que de ahí depende todo lo que vallamos a hacer.





9. Del cuadro **Proyecto** de doble clic en **Módulos** o simplemente presione el signo de + que aparece en la opción **Módulos**. Se activara debajo de **Módulos** la Opción **Modulo1**

10. De doble clic en **Modulo1**. Se mostrara en el Editor de Visual Basic el código de la macro que grabamos de la siguiente forma:

## Sub Macro1()

```
' Macro1 Macro  
' Macro grabada el 08/04/2005 por ....  
' Acceso directo: CTRL+r  
Range("A1").Select  
ActiveCell.FormulaR1C1 = "Paty"  
Range("A2").Select
```

End Sub

Que es lo que significa esto nos preguntaremos asombrados, a continuación se da una explicación de lo que ha hecho **Excel**:

- **Sub y End Sub** indican el inicio y el final del procedimiento de la **Macro1**
- Todo lo que aparece con un apóstrofe ´ indica que no se tomara en cuenta que es solo texto o comentarios y ese texto debe de aparecer en un color, ya sea el color verde.
- **Range("A1").Select** Indica que lo primero que hicimos al grabar la macro fue trasladarnos a la celda **A1**. La orden **Range** nos permite trasladarnos a una celda
- **ActiveCell.FormulaR1C1 = "Paty"** Esto indica que se escribirá en la celda en que se encuentra el valor de texto **Paty**. Todo lo que aparece entre comillas siempre será un valor de texto. La orden **ActiveCell.FormulaR1C1** nos permite escribir un valor en la celda activa.
- **Range("A2").Select** Otra vez indicamos que se traslade a la celda A2. Esto se debe a que cuando escribimos el nombre de **Paty** en **A1** presionamos **Enter** y al dar **Enter** bajo a la celda **A2**. Para comprender alteraremos el código dentro del editor de Visual Basic.

```
Sub Macro1()  
' Macro1 Macro  
' Macro grabada el 01/01/2009 por ....  
' Acceso directo: CTRL+r  
Range("A1").Select  
ActiveCell.FormulaR1C1 = "Paty"  
Range("B1").Select  
ActiveCell.FormulaR1C1 = "Almagro 1822"  
Range("C1").Select  
ActiveCell.FormulaR1C1 = "092794313"  
Range("D1").Select  
ActiveCell.FormulaR1C1 = "Alpallana"  
Range("E1").Select  
ActiveCell.FormulaR1C1 = "SACCEC"  
  
End Sub
```

Así es acabo de alterar el código y cuando regrese a **Excel** y ejecute la macro con **Control + r** hará lo siguiente:

**En A1 escribirá Paty**

**En B1 escribirá Almagro 1822**

**En C1 escribirá 092794313**

**En D1 escribirá Alpallana**

**En E1 escribirá SACCEC**

Así que salgamos del editor dando clic en el **Menú Archivo** y eligiendo la opción **Cerrar y volver a Microsoft Excel**. Si no desea salir por completo de clic en **el botón Microsoft Excel** que se encuentra activado en la barra de tareas y cuando deseé volver al editor de clic en el **botón Microsoft Visual Basic** que se encuentra en la barra de Tareas.

Ahora ya que salimos de **Visual Basic** y estamos en **Excel** de Nuevo ejecutemos la macro presionando **Control + r** y veamos los resultados de nuestra modificación.

### **Practica II**

Genera una **Macro** que escriba un nombre en una celda y lo ponga negrita y observa el **Código**.

Genera una **Macro** que escriba un nombre en una celda y lo Centre y observa el **Código**.

Genera una **Macro** que escriba un nombre en una celda y cambie el tamaño de la letra a 20 puntos y observa el **Código**.

### **CÓDIGOS MÁS COMUNES**

#### **Trasladarse a una Celda**

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

#### **Escribir en una Celda**

```
Activecell.FormulaR1C1="Paty"
```

#### **Letra Negrita**

```
Selection.Font.Bold = True
```

#### **Letra Cursiva**

```
Selection.Font.Italic = True
```

#### **Letra Subrayada**

```
Selection.Font.Underline = xlUnderlineStyleSingle
```

#### **Centrar Texto**

```
With Selection
```

```
    .HorizontalAlignment = xlCenter
```

```
End With
```

#### **Alinear a la izquierda**

```
With Selection
```

```
    .HorizontalAlignment = xlLeft
```

```
End With
```

#### **Alinear a la Derecha**

With Selection

.HorizontalAlignment = xlRight

End With

### **Tipo de Letra(Fuente)**

With Selection.Font

.Name = "AGaramond"

End With

### **Tamaño de Letra(Tamaño de Fuente)**

With Selection.Font

.Size = 15

End With

### **Copiar**

Selection.Copy

### **Pegar**

ActiveSheet.Paste

### **Cortar**

Selection.Cut

### **Ordenar Ascendente**

Selection.Sort Key1:=Range("A1"), Order1:=xlAscending, Header:=xlGuess, \_

OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom

### **Orden Descendente**

Selection.Sort Key1:=Range("A1"), Order1:=xlDescending, Header:=xlGuess, \_

OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom

### **Buscar**

Cells.Find(What:="Paty", After:=ActiveCell, LookIn:=xlFormulas, LookAt \_  
:=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= \_  
False).Activate

### **Insertar Fila**

Selection.EntireRow.Insert

### **Eliminar Fila**

Selection.EntireRow.Delete

### **Insertar Columna**

Selection.EntireColumn.Insert

### **Eliminar Columna**

Selection.EntireColumn.Delete

### **Abrir un Libro**

Workbooks.Open Filename:="C:\Mis documentos\video safe 3.xls"

### **Grabar un Libro**

ActiveWorkbook.SaveAs Filename:="C:\Mis documentos\piscis.xls", FileFormat \_  
:=xlNormal, Password:="", WriteResPassword:="", ReadOnlyRecommended:= \_  
False, CreateBackup:=False

### ***Variables.***

A continuación vamos a repetir el programa Ejemplo1, pero en lugar de poner "Hola" en la celda A1 de la hoja activa, dejaremos que el usuario entre un texto desde teclado y a continuación guardaremos ese valor en esa celda. Observe que el valor que entre del usuario debe guardarse en algún lugar para poder ponerlo después en la celda 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, declararemos la variable de tipo **String** (tipo texto), y lo haremos de la siguiente forma:

**Dim** Texto **As String**

Con esto estamos indicando que se reserve un trozo de memoria, 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

### 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 celda 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 celda  
A1", "Entrada de datos")

### End Sub

## Ejemplo

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

## Option Explicit

### Sub Entrar\_Valor

**Dim Celda As String**

---

### **Dim Texto As String**

```
Celda = InputBox("En que celda quiere entrar el valor", "Entrar Celda")
```

```
Texto = InputBox("Introducir un texto " & Chr(13) & "Para la celda " & Celda ,  
"Entrada de  
datos")
```

```
ActiveSheet.Range(Celda).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 Celda 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 celda, 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.

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 celda 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 celda 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.

### Tipos de datos en Visual Basic para Excel.

Tipos de datos	Tamaño de almacenamiento	Intervalo
<b>Byte</b>	1 byte	0 a 255
<b>Boolean</b>	2 bytes	True o False
<b>Integer</b>	2 bytes	-32.768 a 32.767
<b>Long</b> (entero largo)	4 bytes	-2.147.483.648 a 2.147.483.647
<b>Single</b> (coma flotante/ precisión simple)	4 bytes	-3,402823E38 a -1,401298E-45 para valores negativos; 1,401298E-45 a 3,402823E38 para valores positivos
<b>Double</b> (coma flotante/ precisión doble)	8 bytes	-1,79769313486232E308 a -4,94065645841247E-324 para valores negativos; 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos
<b>Currency</b> (entero a escala)	8 bytes	-922.337.203.685.477,5808 a 922.337.203.685.477,5807



<b>Decimal</b>	14 bytes	+/- 79.228.162.514.264.337.593.543.950.33 5 sin punto decimal; +/- 7,9228162514264337593543950335 con 28 posiciones a la derecha del signo decimal; el número más pequeño distinto de cero es +/- 0,000000000000000000000000000001
<b>Date</b>	8 bytes	1 de enero de 100 a 31 de diciembre de 9999
<b>Object</b>	4 bytes	Cualquier referencia a tipo Object
<b>String</b> (longitud variable)	10 bytes + longitud de la cadena	Desde 0 a 2.000 millones
<b>String</b> (longitud fija)	Longitud de la cadena	Desde 1 a 65.400 aproximadamente
<b>Variant</b> (con números)	16 bytes	Cualquier valor numérico hasta el intervalo de un tipo  Double
<b>Variant</b> (con caracteres)	22 bytes + longitud de cadena	El mismo intervalo que para un tipo String de longitud variable
<b>Definido por el usuario</b> (Type)	Número requerido por los elementos	El intervalo de cada elemento es el mismo que el  Intervalo de su tipo de datos.

### **Conversión de Tipos de datos.**

Copie el siguiente Ejemplo. Simplemente se piden dos números, se suman y se guardan en la celda A1 de la hoja activa.

#### **Ejemplo**

#### **Option Explicit**

#### **Sub Sumar()**

```
    Dim Numero1 As Integer
```

```
    Dim Numero2 As Integer
```

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

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

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

#### **End Sub**

Ejecute el procedimiento y ponga respectivamente los valores 25 y 25. Observe que todo ha ido correctamente y en la celda 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 numéricos, 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.





### **Objeto Cells(fila, columna).**

Sirve, como el objeto range, para referenciar una celda o rango de celdas, pero en lugar de utilizar la referencia de la forma A1, B1, X320,... utiliza la fila y la columna que ocupa la celda dentro de la hoja (objeto WorkSheet). Por ejemplo, para poner hola en la celda A1 de la hoja activa seria:

```
ActiveSheet.Cells(1,1).Value="Hola"
```

### **Utilizar Cells para referenciar un rango.**

Esto seria el equivalente a Range("Celda\_Inicial:Celda\_Final"). La forma que se obtiene utilizando

Cells es un poco más larga, pero se verá que a veces resulta mucho más funcional que utilizando únicamente range. Para referirnos al rango A1:B8, pondremos,

```
Range(Cells(1, 1), Cells(8, 2)).Value = "Hola"
```

Otra forma interesante de Cells es la siguiente,

```
Range("A5:B10").Cells(2, 1).Value = "Hola"
```

Pondrá en la celda A6 el valor "Hola", observe que en este ejemplo Cells comienza a contar filas y columnas a partir del rango especificado en el objeto Range.

### ***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, 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)**

### Ejemplo

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**

### 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**

Ingresar una cantidad que representa el precio de algo utilizando 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 celda 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 celdas 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 celda 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**

El mismo que el anterior pero utilizando variables.

**Option Expl icit**

**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**

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, 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**

Macro que compara los valores de las celdas 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

Ingresa por teclado una cantidad que representa el precio de algo 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 celda 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** *' Sinó 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

Restar los valores de las celda 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 = AactiveSheet.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

Comparar los valores de las celdas 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 .

#### **Ejemplo**

Ingresar 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"))

Precio = 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.

### **Ejemplo**

Ingresar 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"))

```
Precio = 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**

Ingresar 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 celda 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** celda, **una** expresión, etc. deberán llevarse a cabo diferentes acciones o grupos de acciones.



## Ejemplo

Macro que suma, resta, multiplica o divide los valores de las celdas 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

#### 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**

Programa que pide tres notas de un alumno mediante la función InputBox. Las notas van a parar

respectivamente a las celdas 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**

### 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

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.

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.

Constante	Valor	Descripción
VbOK	1	Aceptar
VbCancel	2	Cancelar
VbAbort	3	Anular
VbRetry	4	Reintentar



VbIgnore	5	Ignorar
VbYes	6	Sí
VbNo	7	No

### Con Valor Descpción

### 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**

### Ejemplo

Ingresar 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**

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 ()

**Dim** Nota **As Integer**

**Dim** Media **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

Ingresar 10 valores utilizando la función InputBox, sumarlos y guardar el resultado en la celda 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 celdas de una hoja de cálculo.**

Una operación bastante habitual cuando se trabaja con Excel es el recorrido de rangos de celdas 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 celdas, observe la utilización de las propiedades **Cells** y **Offset**.

### **Propiedad Cells.**

Sirve para referenciar una celda o un rango de celdas según coordenadas de fila y columna.

### **Ejemplo**

Llenar el rango de las celdas 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**

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** Celda\_Inicial **As** String

**Dim** i **As** Integer

**Dim** Fila **As** Integer, Columna **As** Integer

Celda\_Inicial = InputBox("Introducir la celda Inicial : ", "Celda Inicial")

ActiveSheet.Range(Celda\_Inicial).Activate

*' Coges el valor de fila de la celda activa sobre la variable Fila*

Fila = ActiveCell.Row

*' Coges 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 celda activa.

Otra forma de solucionar el ejemplo 23 sería.

**Sub** Ejemplo\_23()

**Dim** Celda\_Inicial **As String**

**Dim** i **As Integer**

**Dim** Fila **As Integer**, Columna **As Integer**

Celda\_Inicial = InputBox("Introducir la celda Inicial : ", "Celda Inicial")

ActiveSheet.Range(Celda\_Inicial).Activate

Fila = 1

**For** i = 1 **To** 10

    ActiveSheet.Range(Celda\_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 celda referenciada por **Range**.

## Ejemplo

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**

## 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**

Programa para ingresar 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()

**Dim** Nombre **As** String

**Dim** Ciudad **As** String

**Dim** Edad **As** Integer

**Dim** fecha **As** Date

*‘ Activar hoja1*

WorkSheets("Hoja1").Activate

*‘ Activar celda 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 celdas 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

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 celda vacía siguiente y cambiar en la instrucción **ActiveSheet.Range("A2").Activate** , la referencia **A2** por la que corresponde a la primera celda 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 celda 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

Es la unión de los dos programas anteriores. Es decir habrá un bucle **Do While** que buscará la primera celda vacía de la base de 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 ingresar 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("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")
```

```
'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**

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

Ingresar 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 celdas, 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.

### **Procedimientos y funciones.**

Se define como procedimiento i/o función a un bloque de código que realiza alguna tarea.

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.

### ***Definir un procedimiento***

**Sub** Nombre\_Procedimento

Sentencias.

**End Sub.**

### ***Llamar a un procedimiento.***

Para llamar un procedimiento desde otro se utiliza la instrucción **Call** *Nombre\_Procedimento*.

**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 continua ejecutándose en el procedimiento *P\_Uno* a partir de la sentencia que sigue a **Call** *P\_Dos*.

### Ejemplo

Es el mismo programa que el visto en el ejemplo 29 pero el código que salta celda 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 celdas 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 celda 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 celda 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 Salto\_Celdas\_Llenas tiene dos parámetros Hoja y Celda\_Inicial que reciben respectivamente la hoja donde está el rango a recorrer y la celda inicial del rango.

**Sub** Ejemplo\_33()

**Dim** Nombre **As** String

**Dim** Ciudad **As** String

**Dim** Edad **As** Integer

**Dim** fecha **As** Date

*' Llamada a la función Salto\_Celdas\_Llenas, observar que mediante dos parámetros se*

*' Al procedimiento en que hoja está el rango a saltar y en la celda donde debe empezar.*

**Call** Salto\_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.
```

```
' Celda_Inicial : Celda Inicial de la columna
```

**Sub** Saltar\_Celdas\_Llenas(Hoja **As String**, Celda\_Inicial **As String**)

WorkSheets(Hoja).Activate

ActiveSheet.Range(Celda\_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** Celda\_Inicial **As String**

Hoja = InputBox("En que hoja está la base de datos : ", "Entrar Nombre de Hoja")

Celda\_Inicial = InputBox("En que celda comienza la base de datos","Celda 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, Celda\_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 celda 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 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 funcionaria 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.***

El paso de parámetros por valor y el paso de parámetros por referencia, sólo indican 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** Celda\_Inicial As String)

**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.

## 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 sería,

**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)

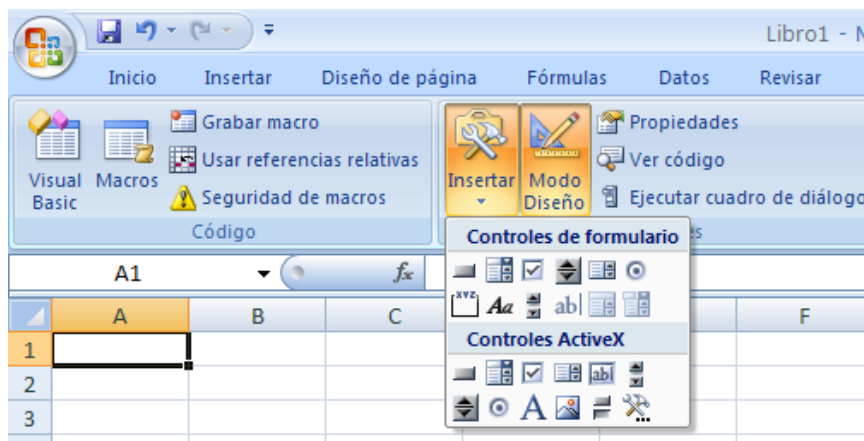
### **Aplicación de ejemplo.**

Para ver el funcionamiento de los distintos controles, construiremos una pequeña aplicación que nos sirva para gestionar una pequeña tabla de registros, básicamente extraer datos que cumplan una determinada condición. La mayoría de funciones que aplicaremos pueden hacerse directamente desde las utilidades del menú **Datos/ Filtro avanzado** que lleva incorporado el propio Excel pero creemos que será un buen ejemplo para ver las posibilidades de los controles.

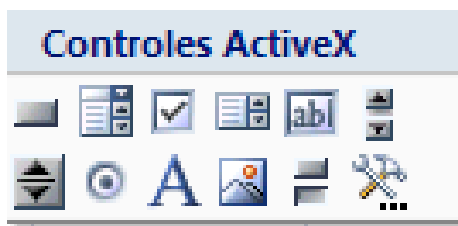
Abra su aplicación Excel y active la hoja de cálculo *Lista7.xls*, en la primera hoja está la lista de registros que utilizaremos en los ejemplos que veremos a continuación. En la Hoja2 se encuentran algunos datos necesarios para los controles.

### **Mostrar la barra de herramientas para Controles ActiveX.**

Para insertar controles en la hoja deberá tener activada la barra de controles ActiveX que se encuentra en la pestaña Programador opción Insertar, allí se encuentran las barras de Controles de formulario y de Controles ActiveX.




La barra de Controles ActiveX permite programar cada uno de los controles.




### ***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.

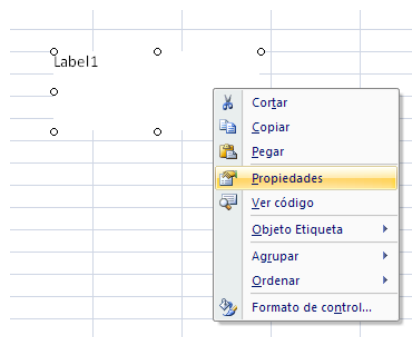
### ***Insertar un Botón.***

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**.

2. 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 pro **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 celda 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 es 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 Celda A16 de Hoja2 y referenciarla con r2, Es la celda donde se copiarán*

*' los datos en caso que se encuentren*

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

**Set** r2 = ActiveCell

*' Activar celda 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 celda 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 celda 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 LinkCell.**

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.

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 celda 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 celda 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*, *Menor*, *Mayor*, *Menor Igual* o *Mayor Igual* 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 Celda A16 de Hoja2 y referenciarla con r2' Es la celda donde se copiarán*

*' los datos en caso que' se encuentren*

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

**Set** r2 = ActiveCell

*' Activar celda A2 de Hoja1 y referenciarla con r1*

Worksheets(1).Activate

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

encontrado = **False**

*' Recorrer todo el rango de datos de Hoja1*

**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 celda 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\_Comparación.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 de la segunda columna deberá utilizar la propiedad

**Column**(*Numero de Columna, Índice 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 **fmOrientationVertical** 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 .

### ***Celdas 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** la celda está activada **Entonces**

Acciones

....

**Fin Si**

A continuación utilizaremos una celda 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 celda esté activada y si lo está proceder a la conversión (sólo si el dato es tipo texto).

Inserte un control celda de verificación. Establezca las siguientes propiedades.

**Name**, Mayusculas.

**Captión**, Mayúsculas.

Para comprobar si la celda 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 celda 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 celda 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 celdas 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 celda 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
```

### REFERENCIAS ELECTRÓNICAS

1. Ayuda de Microsoft Excel 2007.

2. <http://office.microsoft.com/es-hn/infopath/CH011097053082.aspx>

Es un sitio oficial de Microsoft que presenta varios enlaces a artículos relacionados con la Validación de datos en Excel.

3. <http://office.microsoft.com/es-hn/excel/HA010346573082.aspx>

En este sitio encontrarás ejemplos sencillos acerca de la validación de datos en Excel.

4. <http://office.microsoft.com/es-es/excel/HP100725993082.aspx>

Este es un sitio en línea de Microsoft para el área de Excel que presenta ejemplos e información más detallada que incluye las diferentes versiones de Excel desde el 2000 hasta el 2007.

5. <http://office.microsoft.com/es-hn/infopath/CH011097053082.aspx>

Es un sitio oficial de Microsoft que presenta varios enlaces a artículos relacionados con funciones en Excel.

6. <http://www.uv.mx/iip/enrique/sistemasII/apuntesexcel.pdf>

Es un sitio del Instituto Tecnológico Autónomo de México en donde encontrarás generalidades de Excel y reglas para el uso de las bibliotecas de funciones.

7. <http://www.mat21.etsii.upm.es/ayudainf/aprendainf/Excel2000/Excel2000.pdf>

Es un sitio de la Universidad Politécnica de Madrid, que contiene un archivo .pdf que permite aprender Excel desde lo básico e introduce a la utilización de fórmulas y funciones en Excel.

8. <http://www.eumed.net/libros/finanzas.htm>

En este sitio encontraras libros gratuitos con funciones financieras.

9. <http://office.microsoft.com/>

Este es un sitio en línea de Microsoft para el área de Excel en el que podrás consultar información más detallada sobre cualquier tópico de MS Excel.

10. <http://office.microsoft.com/es-es/excel/HP052047113082.aspx?pid=CH062528393082>