MACROS EN EXCEL (Parte I)

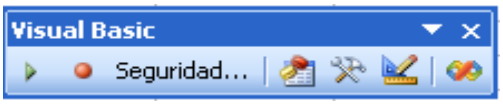
Excel es un programa que tiene un gran potencial, pero la mayoría de la gente lo maneja de una forma muy simple, utilizando solo opciones básicas, pero hay algo muy importante que tengo que decirles. **Excel** cuenta con un lenguaje muy poderoso llamado **Visual Basic**, este es solo una parte del lenguaje, pero permite hacer o resolver los problemas mas fácilmente, solo debemos aprender a programarlo y para eso es este curso, podría la gente decir que este curso es un nivel muy alto de **Excel** y quizás si lo sea, pero es fácil de aprender ya que se manejaran términos sencillos, a mi me gusta hablar con palabras que todo el mundo entienda y eso lo hace mas fácil. La programación que emplea en este curso o las estructuras que aparecen son creadas por su servidor, ya que para manejar la programación de **Visual Basic** con **Excel** es necesario tener mucha creatividad, cada persona puede crear estructuras diferentes pero que trabajen igual. Así que manos a la obra.

Fase 1

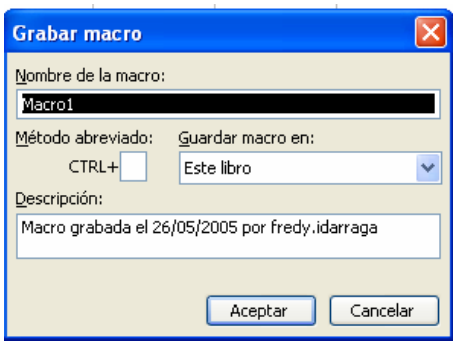
Primeramente debemos de saber que es una Macro y a continuación se explica el término:

Una Macro son una serie de pasos que se almacenan y se pueden activar con alguna tecla de control y una letra. Por ejemplo, todos los días empleo frecuentemente en mis celdas los mismos pasos: Tamaño de Fuente (Tamaño de la letra), Negrita, Fuente (Tipo de letra) y Color de Fuente (Color de Letra), para no estar repitiendo estos pasos los puedo almacenar en una macro y cuando ejecute la macro los pasos antes mencionados se ejecutaran las veces que yo lo desee. A continuación te muestro como grabar una macro y ejecutarla:

1. Trasládate a la celda **A1** y escribe tu Nombre. Por ejemplo, **Fredy** y presiona Enter
2. Regrésate a la celda **A1**, porque cuando diste Enter bajo de celda o cambio el rumbo.
3. Da clic en el **Menú Ver**, seguido por la Opción **Barra de Herramientas** y elija **Visual Basic**. Se activara la barra de herramientas Visual Basic.

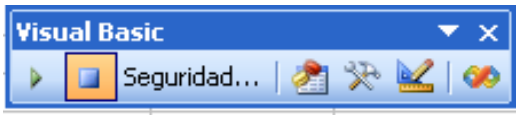


4. Da clic en el botón **Guardar Macro**, el que tiene la ruedita Roja. Windows activa el cuadro de dialogo **Grabar Macro**, el cual permitirá darle el nombre a la macro y cual será el método abreviado para ejecutarla. El método Abreviado se refiere con que letra se va activar la macro, obviamente se activara con la tecla Control y la letra que usted quiera, de preferencia en minúscula, porque si activa las mayúsculas la macro se activara presionando la tecla Shift + Control + la letra que usted indico.



5. Donde dice **Nombre de la macro** ya aparece el nombre que llevara la macro en este caso **Macro1**. si desea cambiar el nombre escriba uno nuevo, pero yo le recomiendo que así lo deje.
6. En la opción **Método Abreviado** aparece que se activara con la tecla **Control (CTRL)** + la letra que usted indica, de clic en el cuadrito y ponga una letra, por ejemplo ponga la letra **a** (en minúsculas). La macro se activara cuando este lista con la tecla **Control + a**

7. De clic en el **Botón Aceptar**. Windows empezara a grabar todos los pasos en la **Macro1**.y el botón de la ruedita azul cambiara de forma ahora será un cuadrito Azul, se llamara **Detener grabación**. Lo utilizaremos cuando terminemos de indicarle los pasos para detener la grabación.
8. Cambie el Tipo de Letra en el **Botón Fuente** de la barra de herramientas Formato
9. Cambie el tamaño de la letra en el **Botón Tamaño de Fuente** de la barra de herramientas Formato
10. Presione el **Botón Negrita** de la barra de herramientas Formato
11. Cambie el color de la letra en el **Botón Color de Fuente** de la barra de herramientas Formato. Recuerde que todos estos pasos están siendo almacenados en la macro que estamos grabando y también recuerde que estos pasos se están efectuando en la celda **A1**.
12. Presione el Botón **Detener Grabación** de la barra de Herramientas de **Visual Basic.** El que tiene el cuadrito azul presionado.



Listo Excel guardo los pasos en la Macro1 que se activara presionado la tecla **Control + a**

13. Escribe otro nombre en la celda **C1** y presiona **Enter**, después regresa a la celda **C1**.
14. Presiona la tecla **Control + a**. Windows efectuara todos los pasos grabados sobre la celda **C1**, esto quiere decir que el nombre que esta en **C1** tendrá las características del que esta en **A1**. Tipo de letra, tamaño, negrita y el color que indicaste al grabar la macro.

Nota. Cada vez que presiones **Control + a** Excel ejecutara la macro y efectuara los pasos en la celda que te encuentres. Puedes grabar todas las macros que desees.

Ahora te recomiendo que domines estos pasos antes de pasar a la siguiente fase. Trata de crear macros que almacenen pasos como estos, recuerda los pasos los vas a indicar tu, que no se te olvide detener la grabación después de que indicaste los pasos, repite este ejercicio las veces que sea necesario para aprendértelo bien.

Practica I

Genera las siguientes Macros:

- Graba una **Macro** que se active con **Control + b** y que esta macro permita abrir un archivo
- Graba una **Macro** que se active con **Control + c** y que esta macro permita insertar un WordArt

Fase 2

Bien, ahora después de practicar la **Fase 1** con diferentes ejemplos o **Macros** pasaremos a la siguiente **Fase** que nos permitirá observar los códigos que hemos generados con nuestra macros. Te recomiendo que salgas de **Excel** y vuelvas a entrar, para que trabajes limpio sin ninguna macro y empezando de la macro1 de nuevo.

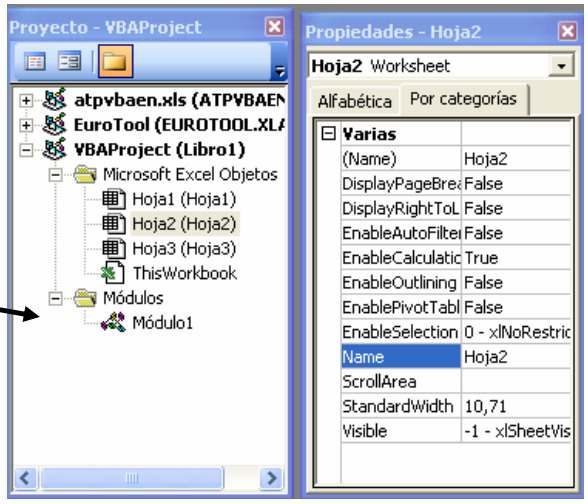
OBSERVANDO LOS CODIGOS DE UNA MACRO DE EXCEL

Crearemos una macro y veremos sus códigos:

Para observar los códigos de una macro debemos de seguir los siguientes 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 **Fredy**, 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. También puede acceder desde el **Menú Herramientas, Macro y Editor de Visual Basic**.
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 26/05/2005 por FREDY IDARRAGA
' Acceso directo: CTRL+r
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Fredy"
    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 apostrofe ´ 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 = "Fredy"** Esto indica que se escribirá en la celda en que se encuentra el valor de texto **Fredy**. 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 **Fredy** en **A1** presionamos **Enter** y al dar **Enter** bajo a la celda **A2**.

Para comprender mejor alteraremos el código dentro del editor de Visual Basic:

```
Sub Macro1()
' Macro1 Macro
' Macro grabada el 26/05/2005 por FREDY IDARRAGA
' Acceso directo: CTRL+r
    Range("A1").Select
    ActiveCell.FormulaR1C1 = "Fredy"
    Range("B1").Select
    ActiveCell.FormulaR1C1 = "Carrera 49 No. 7 Sur 50 Avenida las Vegas"
    Range("C1").Select
    ActiveCell.FormulaR1C1 = "261.95.00 Ext. 429"
    Range("D1").Select
    ActiveCell.FormulaR1C1 = "La Aguacatala"
    Range("E1").Select
    ActiveCell.FormulaR1C1 = "UNIVERSIDAD EAFIT"
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 Fredy
En B1 escribirá Carrera 49 No. 7 Sur 50 Avenida las Vegas
En C1 escribirá 261.95.00 Ext. 429
En D1 escribirá La Aguacatala
En E1 escribirá UNIVERSIDAD EAFIT

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.

Que te parece es sencillo o No?, Claro necesitamos practicar bastante para dominar esto, así que repasa la **Fase 2** cuantas veces sea necesario, otra cosa no trates de generar códigos muy complejos en tus macros porque te vas a enredar, poco a poco se va lejos.

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="Fredy"`
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:="Fredy", 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
```

Estos serian algunos códigos muy comunes en **Excel**, pero si usted desea puede generar más códigos de otras opciones, es cuestión de que los ocupe.

Antes de pasar a la **Fase 3** es necesario que domines generar **Macros** y Observar sus códigos que se encuentran en la **Fase 2.** Recuerda esto es de mucha práctica y para eso es necesario aprender bien las fases Anteriores.

Fase 3

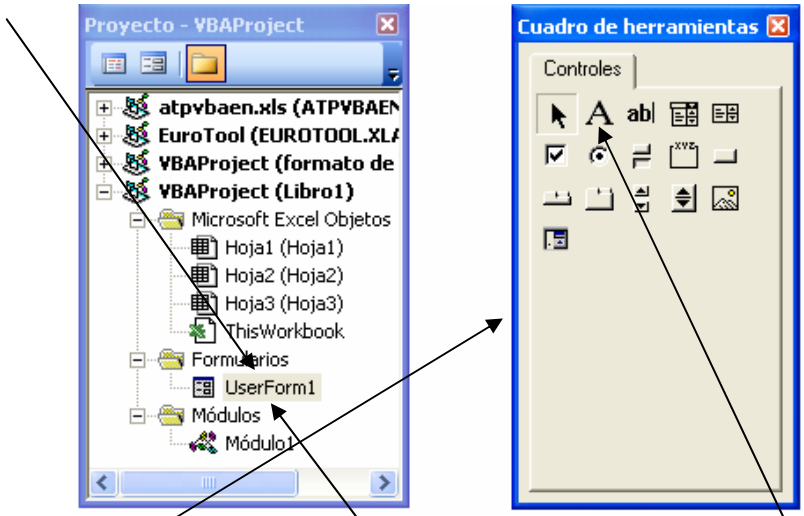
CREANDO FORMULARIOS Y PROGRAMÁNDOLOS

Antes de Empezar esta nueva fase te recomiendo que salgas de **Excel** y vuelvas a entrar, esto es por si estuviste practicando los códigos, para que no quede una secuencia de Macros.

Ahora te enseñare 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:

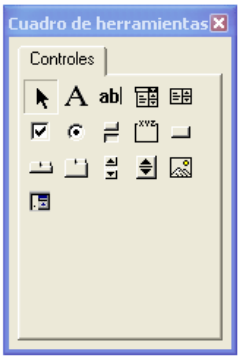
1. Presione La Teclas **Alt + F11,** para entrar al editor de **Visual Basic**.
2. Activa las siguientes opciones:
   - De clic en el **Menú Ver** y elija la opción **Explorador de Proyectos**
   - De clic en el **Menú ver** y elija la opción **Ventana Propiedades**
3. Del **Menú Insertar** elija la Opción **UserForm.** Esto inserta el Formulario que programaremos con controles. En el **Explorador de Proyecto** se observara que se inserto el **UserForm**.

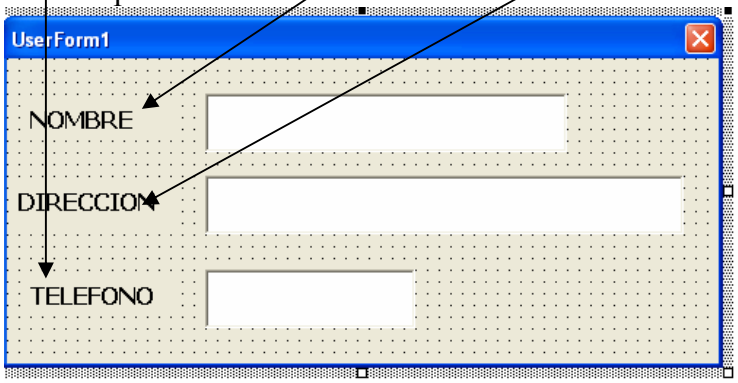


También cuando de clic en el Formulario **USERFORM1** se debe de activar **el Cuadro de Herramientas**, si no se activa de clic en el **Menú Ver** y elija la opción **Cuadro de Herramientas**.

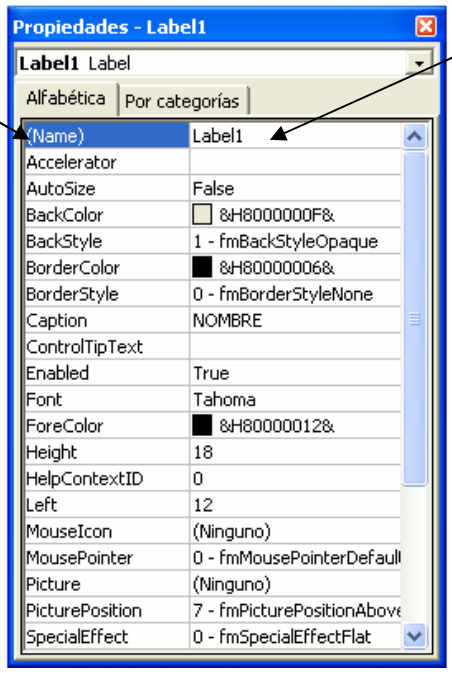
4. Elija del **Cuadro de Herramientas** el Control **Etiqueta** el que tiene la **A** y Arrastre dibujando en el Formulario **USERFORM1** la etiqueta. Quedara el nombre Label1, después de un clic en la etiqueta dibujada y podrá modificar el nombre de adentro y pondremos ahí **Nombre.** Si por error da 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





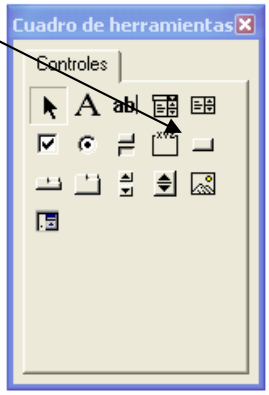
Si tiene algún problema al dibujar las etiquetas o los cuadros de texto, solo cámbiele el nombre a la etiqueta o el cuadro de texto en la **Ventana Propiedades** la opción se llama **(Name).** El Error que marque puede ser **Nombre Ambiguo,** pero si le cambias el Nombre al control se quitara el error.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

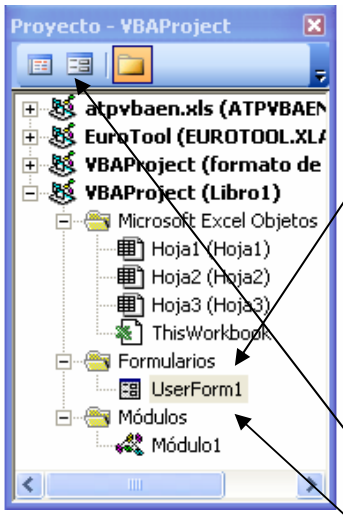


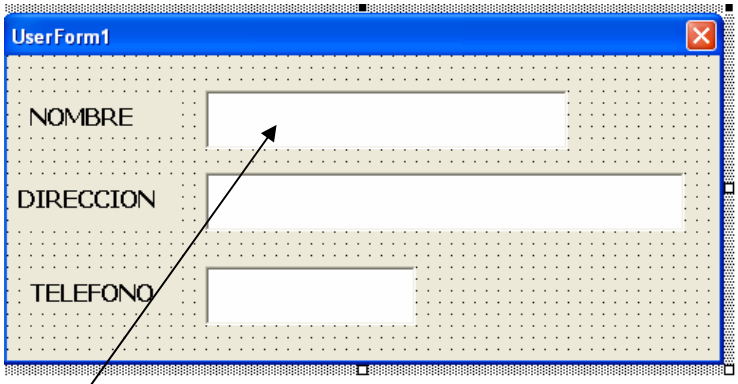
Solo altera esto si te marca error, si NO déjalo así.

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 valla 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 valla 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()
        Rem inserta un renglón
        Selection.EntireRow.Insert
        Rem Empty Limpia Los Textbox
        TextBox1 = Empty
        TextBox2 = Empty
        TextBox3 = Empty
        Rem 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 activara 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**.

En este archivo que usted bajo se encuentra una hoja de Excel Libre de Virus o sea que esta limpio, ábralo sin ningún problema, ya que ahí viene un ejemplo de la Macro ya realizada y solo la ejecutara y vera como trabajan las Macros. Espero y estés pendiente porque vienen mas partes sobre este interesante curso de Macros.

El Archivo Viene con el Nombre de Macro1

MACROS EN EXCEL (Parte II)

En la segunda parte de **Guía de Macros en Excel** que te será de gran utilidad, se manejaran **Formulas** en los Formularios, **Búsquedas de Texto** y **El Asistente de Windows**. Entonces empecemos Amigos.

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**.

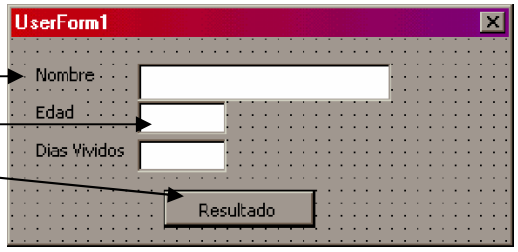
Fase I

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**.

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
```

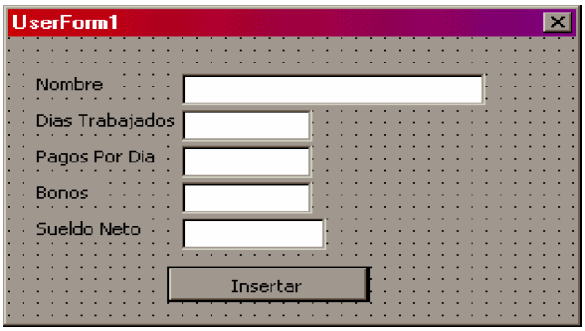
```
Private Sub TextBox1_Change()
        Range("A9").Select
        ActiveCell.FormulaR1C1 = TextBox1
End Sub

Private Sub TextBox2_Change()
        Range("B9").Select
        ActiveCell.FormulaR1C1 = TextBox2
        Rem aquí se crea la Formula
        TextBox3 = Val(TextBox2) * 365
        Rem El Textbox3 guardara el total de la multiplicación del Textbox2 por 365
        Rem El Comando Val permite convertir un valor de Texto a un Valor Numérico
        Rem 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
```

Esto va permitir que cuando se ejecute el formulario y se de la edad el resultado de los días vividos aparecerá en el **Textbox3** y se escribirá también en **Excel**. El comando **Val** es un comando de **Visual Basic** que te permite convertir un valor de texto a un valor numérico. Recuerden el Comando Rem se utiliza para poner Comentarios únicamente y no afecta a la programación.

Este Archivo de esta **Macro** se llama **Macros de Edad** y viene incluido aquí.

Generaremos otro ejemplo, Crea el Siguiente Formulario con los siguientes datos:

- 5 Etiquetas y
- 5 Textbox
- 1 Botón de Comando



Los datos que se preguntaran serán Nombre, Días Trabajados, Pago por Día, Bonos y Sueldo Neto.

Genera el siguiente código:

```
Private Sub CommandButton1_Click()
        Selection.EntireRow.Insert
        TextBox1 = Empty
        TextBox2 = Empty
        TextBox3 = Empty
        TextBox1.SetFocus
End Sub
```

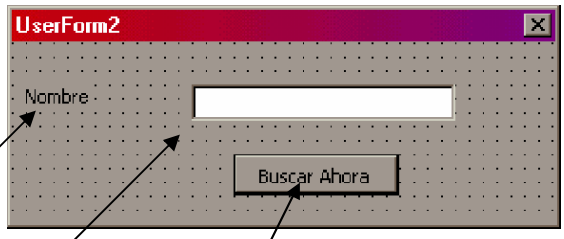
```
Private Sub TextBox1_Change()
        Range("A9").Select
        ActiveCell.FormulaR1C1 = TextBox1
End Sub

Private Sub TextBox2_Change()
        Range("B9").Select
        ActiveCell.FormulaR1C1 = TextBox2
End Sub

Private Sub TextBox3_Change()
        Range("C9").Select
        ActiveCell.FormulaR1C1 = TextBox3
End Sub

Private Sub TextBox4_Change()
        Range("D9").Select
        ActiveCell.FormulaR1C1 = TextBox4
        Rem aquí se crea la formula
        TextBox5 = Val(TextBox2) *  Val(TextBox3) + Val(TextBox4)
        Rem El TextBox5 guardara  el total
End Sub

Private Sub TextBox5_Change()
        Range("E9").Select
        ActiveCell.FormulaR1C1 = TextBox5
End Sub
```

Cuando se introduzca el Bonos automáticamente se generara el Sueldo Neto.

Este ejemplo viene en el Archivo **Macros de Sueldo Neto**

BUSANDO INFORMACIÓN CON UN TEXTBOX

Se puede buscar información con un Textbox programándolo de la siguiente forma:



Dibuje una **Etiqueta**, un **Textbox** y un **Botón de Comando** y agregue el siguiente Código:

```
Private Sub TextBox1_Change()
        Range("a9").Select
        ActiveCell.FormulaR1C1 = TextBox1
End Sub

Private Sub CommandButton1_Click()
        Cells.Find(What:=TextBox1, After:=ActiveCell, LookIn:=xlFormulas, LookAt _
            :=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _
            False).Activate
End Sub
```

Si te fijas incluí en la programación del **Botón Buscar Ahora** que buscara lo que en el Textbox1 a la hora de Presionarse.

Este proceso viene en el Archivo Completo **Macro2**

TRABAJANDO CON EL ASISTENTE

El asistente es el personaje de **Office** que se activa para ayudarnos y una de las ventajas es que podemos Manipularlo, por ejemplo se le puede dar animación, Moverse, Hacer Preguntas, Etc.

A continuación se muestran algunos códigos del Asistente:

Este código permite hacer visible el ayudante o sea mostrarlo. Si deseas ocultarlo solo cambia la opción **True** por **False**.

Assistant.Visible = True

Este código permite Mover el Asistente a un nuevo lugar, solo cambia los valores numéricos y cambiara de posición.

Assistant.Move 430, 230

Este código permite activar un efecto de animación, cuando escribas el signo Igual después de Assistant.Animation = aparecerá un menú con diferentes efectos de animación

Assistant.Animation = msoAnimationListensToComputer

Este ejemplo permite crear un **Nuevo Asistente** para poderlo manipular con una pregunta y que tu contestes. La variable **t** guardara el valor de la respuesta, si el valor es -3 significa que es Si y por lo tanto borrara el renglón.

```
With Assistant.NewBalloon
    .Text = "Deseas Borrar este Registro"
    .Button = msoButtonSetYesNo
    .Heading = "Advertencia"
     t = .Show
End With


If  t = -3 Then
    Assistant.Animation = msoAnimationEmptyTrash
    Selection.EntireRow.Delete
End If
```

Este ejemplo viene en el archivo **Macro2**

MACROS EN EXCEL (Parte III)

Esta unidad será de gran utilidad, ya que se manejaran **Consultas en los Formularios, accesos a las Macros desde Excel sin necesidad de entrar a Visual Basic y algunos métodos de trabajar más fácil**.

ELABORANDO UNA CONSULTA

Todo **Registro de información** debe de tener su propia **Consulta, Baja y Modificación**, es por eso que en este nuevo capitulo nos concentramos en ello, primeramente en poder **consultar la información** que ya se escribió en la **Hoja de Excel**, obviamente desde una **Macro** combinada con **Visual Basic**, observemos el siguiente ejemplo:

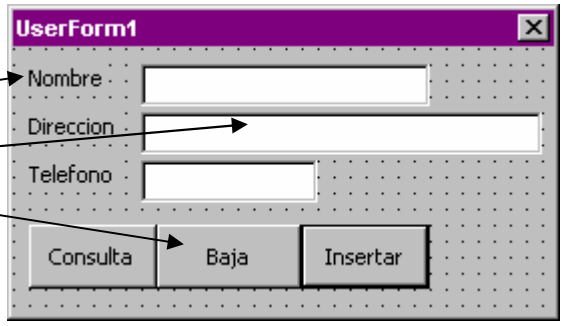
Fase I

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**.

Ahora crearas un formulario con el siguiente aspecto:

el formulario tendrá:

- Tres etiquetas
- Tres Textbox
- Tres Botones de Comando



Los datos que se preguntaran serán **Nombre, Dirección y Teléfono**. Los tres botones nos servirán para lo siguiente: **Consultar** consultara la información que hayamos insertado desde el botón insertar. **Baja** podrá eliminar algún dato que se consulto y no lo queremos. **Insertar** tendrá la función de insertar los registros que vayamos dando de alta, es como los ejercicios anteriores. A continuación se muestra como se deben de programar estos Controles:

Programación de los Controles:

BOTON DE CONSULTA

```
Private Sub CommandButton1_Click()
    Cells.Find(What:=TextBox1, After:=ActiveCell, LookIn:=xlFormulas, LookAt _
        :=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _
        False).Activate
    ActiveCell.Offset(0, 1).Select
```

```
        TextBox2 = ActiveCell
        Rem la línea que contiene el ActiveCell.Offset(0, 1).Select permite moverse una
            columna a la derecha, por lo tanto después de la búsqueda de las primeras líneas
            con Cell.Find si encuentra el Nombre de la persona se mueve a la siguiente
            columna y la línea TextBox2 = ActiveCell Permite capturar el valor de la celda al
            Textbox2 y así mostrar el dato de la celda en el TextBox2.
        ActiveCell.Offset(0, 1).Select
        TextBox3 = ActiveCell
        Rem Cada vez que se escriba la línea ActiveCell.Offset(0, 1).Select significa que se
            tiene que moverse una columna a la derecha.
        Rem Si el nombre que tratas de consultar no se encuentra podría generar un error
            porque fallaría el Cell.Find esto puede ocurrir en el Word 97, yo trabajo con el
            Word 2000 o XP y no tengo ese problema. Pero esto se solucionaría con una
            trampa de error.
End Sub
```

BOTON BAJA

```
Private Sub CommandButton2_Click()
        Selection.EntireRow.Delete
        Range("A9").Select
        TextBox1 = Empty
        TextBox2 = Empty
        TextBox3 = Empty
        TextBox1.SetFocus
End Sub
```

BOTON INSERTAR

```
Private Sub CommandButton3_Click()
        Range("A9").Select
        Selection.EntireRow.Insert
        TextBox1 = Empty
        TextBox2 = Empty
        TextBox3 = Empty
        TextBox1.SetFocus
End Sub
```

CUADROS DE TEXTO

```
Private Sub TextBox1_Change()
        Range("A9").FormulaR1C1 = TextBox1
        Rem esta primer línea reemplaza a estas dos…… que te parece todavía mas corta
        Range("A9").Select
        ActiveCell.FormulaR1C1 = TextBox1
End Sub
```

```
Private Sub TextBox2_Change()
        Range("B9").FormulaR1C1 = TextBox2
End Sub

Private Sub TextBox3_Change()
        Range("C9").FormulaR1C1 = TextBox3
End Sub
```

Si con el **Botón Consulta** tienes un error cuando no encuentra a la persona, entonces tendrás que agregar esto a tu código del **Botón Consultar**

BOTON DE CONSULTA

```
Private Sub CommandButton1_Click()
        On Error Goto noencontro
        Rem esta línea genera una trampa de error si Excel encuentra un error se le dice que se
            vaya a la etiqueta noencontro que esta definida mas adelante en el código. No use
            la trampa de error si no tiene problemas a la hora de que no encuentra a la persona.
            Recuerde si usted comete cualquier error Excel se dirigirá a la etiqueta
            noencontro.y esquivara cualquier error, hasta uno que usted cometa en la
            programación.
        Cells.Find(What:=TextBox1, After:=ActiveCell, LookIn:=xlFormulas, LookAt _
            :=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _
            False).Activate
        ActiveCell.Offset(0, 1).Select
        TextBox2 = ActiveCell
        ActiveCell.Offset(0, 1).Select
        TextBox3 = ActiveCell
        Rem También se puede utilizar este código para leer la información de las celdas lo
            que esta en azul. La diferencia es que se asignan los valores a variables y después
            se descargan a los TextBoxs.
        ActiveCell.Offset(0, 1).Select
        Direccion = Activecell
        ActiveCell.Offset(0, 1).Select
        Telefono = Activecell
        TextBox2 = Direccion
        TextBox3 = Telefono

        noencontro:
        Rem Aquí se esquiva el error
End Sub
```

Que te parece es increíble como una Macro combinada con Visual Basic puede hacer hasta lo imposible

Bueno ya tenemos elaborado un ejercicio de consultas de datos, ahora accesaremos al formulario desde **Excel** sin necesidad de entrar al **Editor de Visual Basic**.

Para realizar este ejercicio debemos permanecer dentro del **Editor de Visual Basic** para poder introducir el código en un **Modulo**, por lo tanto deberás seguir los siguientes pasos:

- De clic en el **Menú Insertar** y elija la opción **Modulo**
- Escriba dentro del **Modulo** el nombre del modulo en este caso **Sub Entrada**

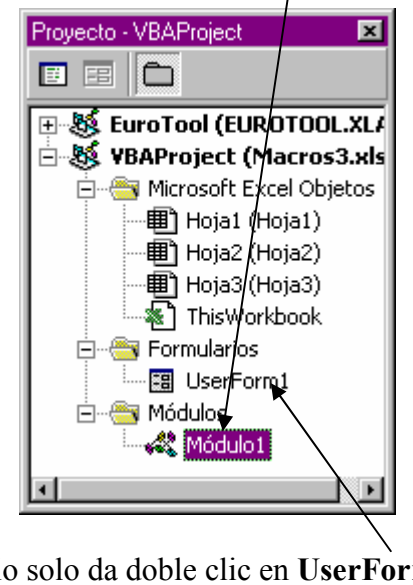
Cuando usted escriba Sub Entrada aparecerá de la siguiente manera:

```
Sub Entrada()
        Load UserForm1
        UserForm1.Show
End Sub
```

Usted deberá escribir las dos líneas que están en medio que son:
Load UserForm1
UserForm1.Show

La primer línea significa que cargue a la memoria el formulario que se llama **UserForm1**, la segunda línea significa que lo muestre, esto quiere decir que en el modulo estamos escribiendo el código de una macro que permitirá cargar el formulario desde Excel sin necesidad de entrar al **Editor de Visual Basic**.

Si te fija en el explorador de proyecto aparece el **Modulo** que creamos.



si queremos volver al formulario solo da doble clic en **UserForm1**
Bueno ya esta listo ahora salgamos del **Editor de Visual Basic** y volvamos a **Excel.**

- De clic en el **Menú Archivo** del **Editor de Visual Basic**
- Elija la opción **Cerrar y volver a Microsoft Excel**

Ya que estamos en **Excel**, podemos insertar una imagen o un botón o cualquier grafico, por ejemplo:

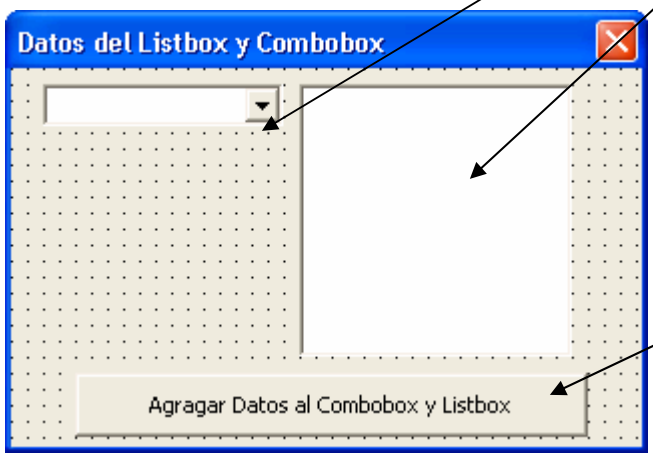
- De clic en el **Menú Insertar**
- Elija la opción **Imagen**, seguido por **Imagen Prediseñada**
- inserte cualquier imagen y dele el tamaño que usted desea.
- De clic derecho sobre la Imagen
- Elija la opción **Asignar Macro**
- De clic en la **Macro** que se llama **Entrada**, es obvio la única que hicimos
- De Clic en **Aceptar**
- De clic fuera de la imagen en cualquier celda y listo si presionas la imagen cargara el formulario.

Este ejemplo viene en el archivo Macros3 junto con esta guía.

MACROS EN EXCEL (Parte IV)

En esta nueva entrega veremos como trabajar con Listbox y Combobox, que son listas de opciones y cuadro de opciones, también aprenderemos como ejecutar una hoja de Excel y activar automáticamente un Macro, como ordenar información, como convertirla en mayúscula o minúscula y como crear modificación automática de X información.

Para agregar información a un **Combobox** y un **Listbox**, primeramente deberás crear el siguiente formulario dentro de **Visual Basic**, recuerda desde Excel se utiliza la tecla **ALT + F11** para entrar a **Visual Basic**, seguido del **Menú Insertar** y después **Userform**, bueno creo que ya lo sabes. Inserta Un **Combobox y un Listbox** y un **Botón**.



Ya creaste la Interfaz vamos a programar el botón, veremos como se le puede agregar información por medio de código a estos dos controles.

Da doble clic en el **Botón** y escribe las siguientes líneas dentro del procedimiento.

```
Private Sub CommandButton1_Click()
        ComboBox1.AddItem "Juan Jose"
        ComboBox1.AddItem "Pedro de la Fuente"
        ComboBox1.AddItem "Salvador de la Luz"
        ListBox1.AddItem "Juan José"
        ListBox1.AddItem "Pedro de la Fuente"
        ListBox1.AddItem "Salvador de la Luz"
End Sub
```

Bueno vamos a analizar el significado de estas líneas:
ComboBox1.AddItem "Juan José "

La opción **AddItem** significa que vas a agregar un dato de texto, por lo tanto se entiende como vas a agregar a Juan José al **Combobox1**, por lo tanto yo puedo agregar los datos que quiera a un **Combobox** o un **Listbox** con la opción AddItem, entonces al presionar el botón aparecerán los datos que se encuentra escritos y podrás seleccionar cualquiera de ellos, recuerda que la información la vas a agregar según tus necesidades.

Si deseas agregar números a un Combobox o ListBox escribe el siguiente código en un botón:

```
Private Sub CommandButton1_Click()
        For X=1 to 50
                Listbox1.AddItem str(x)
        Next
End Sub
```

La Instrucción **For-Next** es un ciclo contador que te permite contar desde un numero hasta otro. Por ejemplo le digo que cuente desde el 1 hasta el 50 y lo que se encuentre dentro del ciclo **For-Next** se ejecutara el número de veces, la X es una variable numérica que guarda el valor, cada vez que el ciclo da una vuelta aumenta un numero, por lo tanto X va a valer desde 1 hasta 50, y la instrucción **Str** es para convertir el valor numérico de la X en valor de Texto, ya que la opción **AddItem** guarda solo texto, claro esta que también puede funcionar sin esta instrucción en algunos casos.

Por lo tanto el Listbox1 va a guardar los número del 1 al 50, sin necesidad de irlos poniendo de uno por uno, imagínatelo.

Listbox1.AddItem "1"
Listbox1.AddItem "2"
Listbox1.AddItem "3"

Ya te quiero ver en el código para que llegues al 50, largo verdad.

Bueno esto es para introducirle datos a un **Listbox y Combobox**, pero como puedo usar estos datos para enviarlos para una celda, en el siguiente ejemplo te lo explico:

Da doble clic en el **Listbox** y escribe el siguiente código:

```
Private Sub ListBox1_Click()
    Range("a9").Select
    ActiveCell.FormulaR1C1 = ListBox
End Sub
```

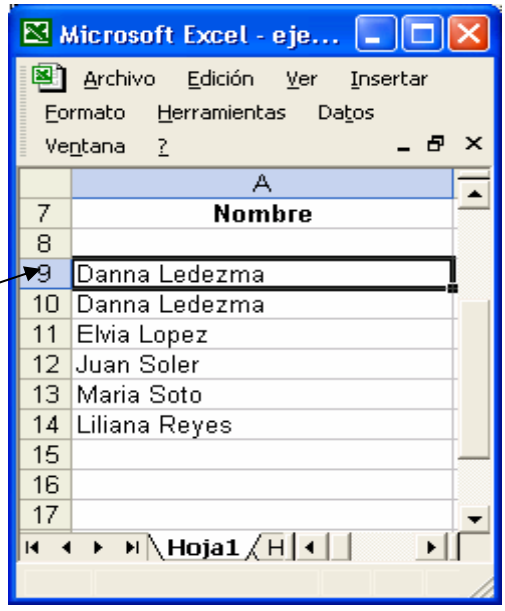
Así de de fácil cada vez que escojas un dato que se encuentre en un **Listbox1** lo enviara a la celda **a9**, escribiéndolo ahí. Si lo deseas hacer lo puedes hacer en un Combobox, solo cambia **Listbox1** por **Combobox1** y se acabo.

Ahora si deseas agregar los datos al **Listbox** o **Combobox** sin ningún botón que presionar escribe el siguiente código:

```
Private Sub UserForm_Activate()
    ComboBox1.AddItem "Juan Jose"
    ComboBox1.AddItem "Pedro de la Fuente"
    ComboBox1.AddItem "Salvador de la Luz"
    ListBox1.AddItem "Juan José"
    ListBox1.AddItem "Pedro de la Fuente"
    ListBox1.AddItem "Salvador de la Luz
End Sub
```

La Clave esta en el procedimiento **UserForm_Activate()** esto quiere decir que cuando se active el formulario cargara lo que tu le indiques, en este caso va a introducir los datos al **Listbox1** y **Combobox1** automáticamente, que te parece.

Ahora si deseas tomar información de una celda y enviarla a un **Combobox** o **Listbox** escribe el siguiente código en un Botón:




```
Private Sub CommandButton1_Click()
        Range("a9").Select
        Do While ActiveCell <> Empty
           ActiveCell.Offset(1, 0).Select
           ListBox1.AddItem ActiveCell
        Loop
End Sub
```

Fíjate bien, primeramente muevo el rango a la celda **a9** porque ahí esta el inicio de mi información, después la línea **Do While Activecell<> Empty** significa Hazlo mientras la celda no se encuentre vacía, la siguiente línea que es **ActiveCell.Offset(1, 0).Select,** significa Baja un Renglón, la siguiente línea **ListBox1.AddItem ActiveCell,** agrega la información de la celda al **Listbox1** y la línea **Loop** es parte del ciclo **Do While,** siempre cierra el ciclo, como el **For-Next.** Por lo tanto todos los nombres que estén delante de **a9** serán enviados al **Listbox1** y cuando tope con la celda **a15** que se encuentra vacía la condición del **Do While** parara la ejecución de su código. Esto funciona caminando renglones hacia abajo, pero si deseas moverte hacia la derecha por columnas solo cambia la línea **ActiveCell.Offset(1, 0).Select** por **ActiveCell.Offset(0, 1).Select,** quiere decir que se mueva por columna, no por renglón. **ActiveCell.Offset(Renglón, Columna).Select**

Si cambias el 1 por otro número se moverá el numero de veces que tu le indiques, por ejemplo si quiero bajar 10 renglones de un golpe:

```
ActiveCell.Offset(10, 0).Select
```

Si quiero moverme 20 columnas a la derecha

```
ActiveCell.Offset(0, 20).Select
```

Así funciona esto.

Ahora veremos como se ejecuta una macro a la hora de abrir un libro

Primeramente inserta un **Modulo** del **Menú Insertar** dentro de **Visual Basic** y escribe el siguiente código:

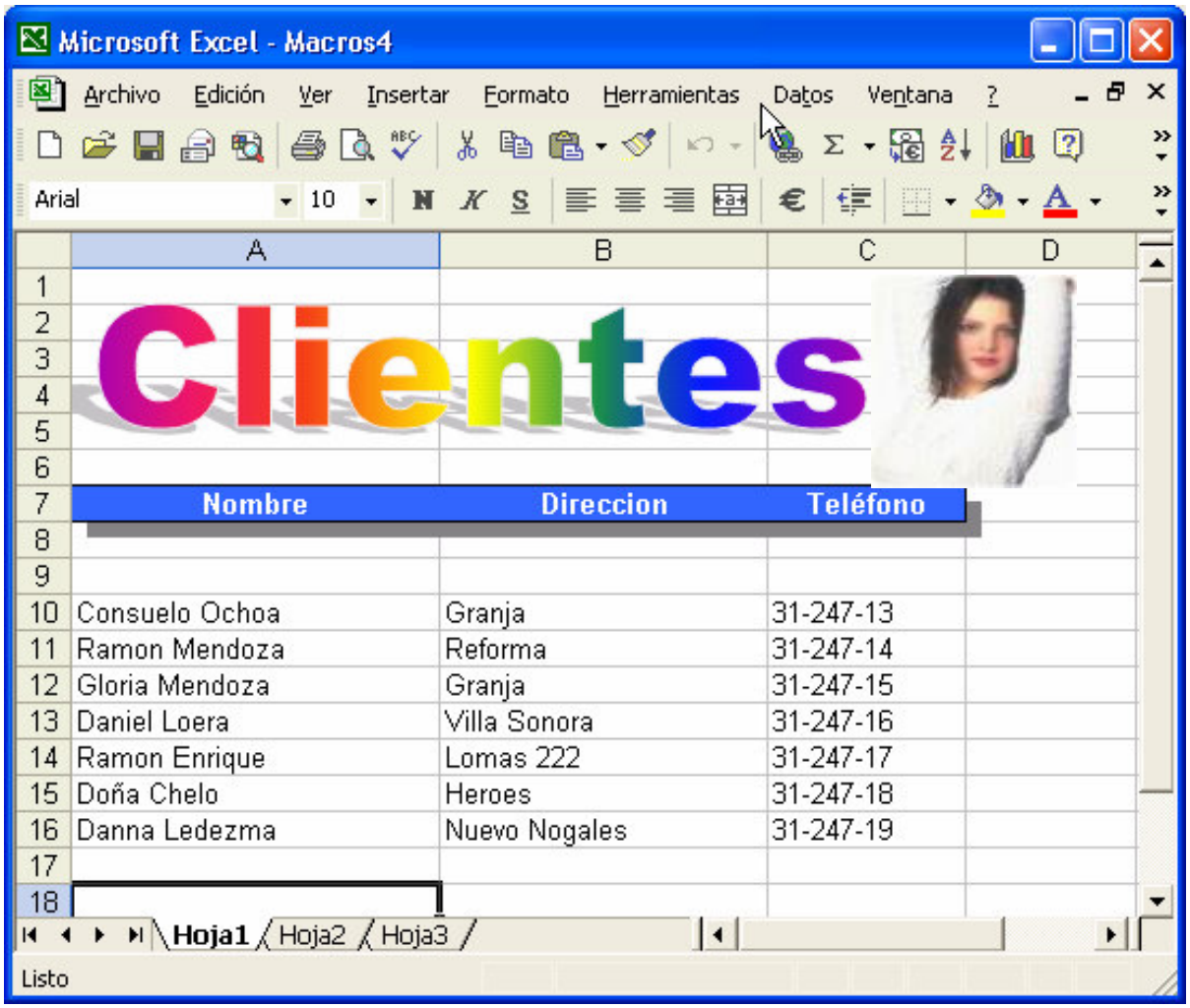
```
Sub Auto_open()
        Load UserForm1
        UserForm1.Show
End Sub
```

La magia esta en el procedimiento **Auto_open()** que permite ejecutar automáticamente lo que se encuentre dentro de el cuando abras un libro que contenga este código, en este ejemplo cuando se abre el libro se activa el formulario 1 que programe.

Así que todo lo que agregues dentro de este procedimiento se ejecutara automáticamente cuando abras un libro, que te parece.

A continuación veremos como ordenar una información por orden alfabética ascendente, es un código muy completo y bueno que te permite localizar los datos y ordenarlos, sin pasarse un renglón en blanco.

Observemos el siguiente ejemplo y aprendamos de el:
Si se fijan en la siguiente pantalla tengo datos en una hoja que empinan en el renglón **A10** y terminan en **C16**, el siguiente código detectara donde debe detenerse para poder ordenar los datos. Es necesario crear el código para ordenar datos, pero aquí yo te lo muestro:



Programa esto en el botón1

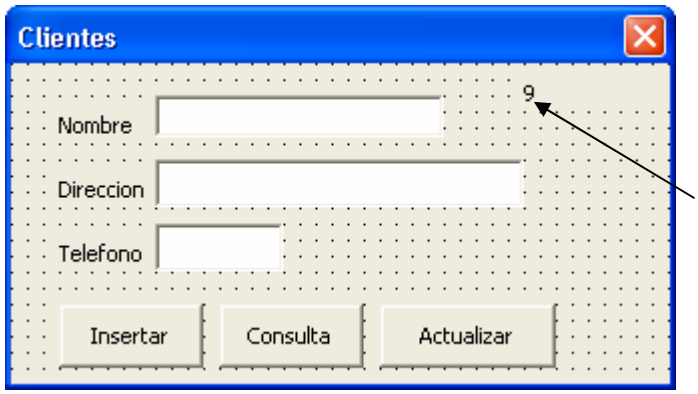
```
Private Sub CommandButton1_Click()
        Rem este código localiza el ultimo registro por medio del renglón
        Range("a10").Select
        Do While ActiveCell <> Empty
            ActiveCell.Offset(1, 0).Select
        Loop
        Rem llega hasta el a17 donde no hay información y se regresa un renglón para ser
            exacto con la siguiente línea.
            ActiveCell.Offset(-1, 0).Select
        Rem este código localiza la última columna del último dato
        Do While ActiveCell <> Empty
            ActiveCell.Offset(0, 1).Select
        Loop
            ActiveCell.Offset(0, -1).Select
        Rem esta línea guarda en la variable celdaactiva la celda exacta donde esta el
            ultimo dato de la ultima columna de información, en este caso C16.
            celdaactiva = ActiveCell.Address
        Rem este código toma el rango desde A10 donde empieza la información, hasta
            donde encontró el ultimo dato C16, que lo guarda la variable celdaactiva.
            Selecciona de A10 hasta C16.
          Range("A10:" + celdaactiva).Select
        Rem este código ordena los datos en orden ascendente, el código fue generado en
            Excel, así que si no sabes generarlo solo copialo de aquí.
        Selection.Sort Key1:=Range("A10"), Order1:=xlAscending, Header:=xlGuess, _
            OrderCustom:=1, MatchCase:=False, Orientation:=xlTopToBottom
End Sub
```

Así es como funciona este código de Macros de Excel ordenando exactamente desde A10 hasta donde están los datos finales.

Bueno ahora para convertir la información a Minúscula o Mayúscula es muy parecido el código, solo obsérvalo:

```
Private Sub CommandButton1_Click()
Range("a10").Select
Do While ActiveCell <> Empty
    ActiveCell.FormulaR1C1 = LCase(ActiveCell)
    ActiveCell.Offset(1, 0).Select
Loop
End Sub
```

Así es la magia esta en **Lcase** que convierte a Minúsculas y **Ucase** a Mayúsculas, empieza en A10 y hasta que no encuentra datos deja de convertir a Minúsculas.



El siguiente Formulario y código muestra la fuerza de cómo se puede consultar y modificar el dato que se encontró.

Etiqueta 4, escríbele el numero 9 dentro.

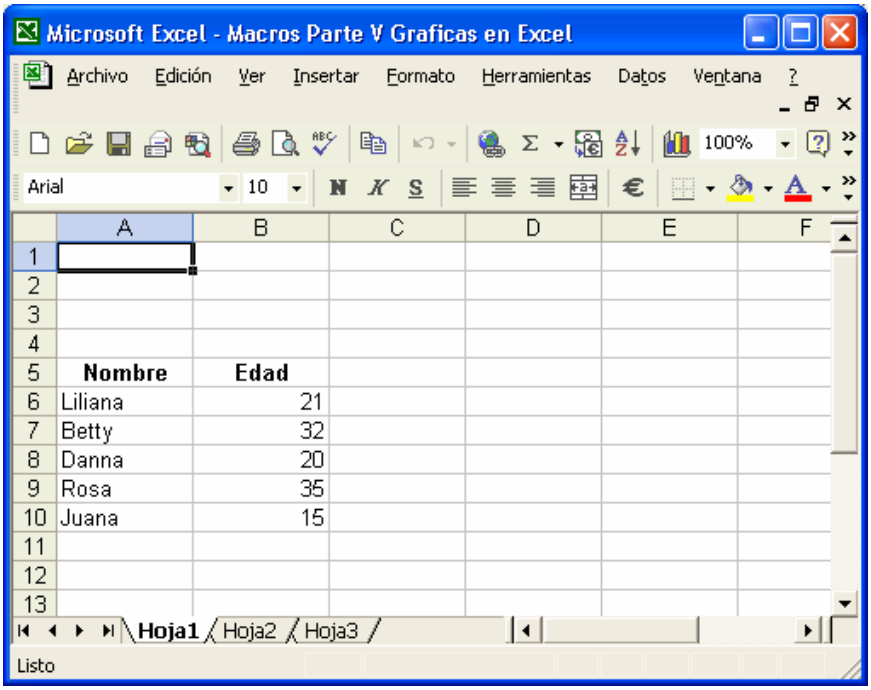
Crea la siguiente Interfaz, 4 Etiquetas, 3 Textbox y 3 Botones

Copia el siguiente código:

```
Private Sub CommandButton1_Click()
        Rem si no se escribe nada en los Textboxs a la hora de insertar escribe No Tiene
        If  TextBox1 = Empty Then Range("A9").FormulaR1C1 = "No Tiene"
        If  TextBox2 = Empty Then Range("B9").FormulaR1C1 = "No Tiene"
        If  TextBox3 = Empty Then Range("C9").FormulaR1C1 = "No Tiene"

        Range("A9").Select
        Selection.EntireRow.Insert
        TextBox1 = Empty
        TextBox2 = Empty
        TextBox3 = Empty
        TextBox1.SetFocus
End Sub

Private Sub CommandButton2_Click()
        On Error GoTo noencontro
        Rem Código para buscar, ya lo conocemos
        Cells.Find(What:=TextBox1, After:=ActiveCell, LookIn:=xlFormulas, LookAt _
            :=xlPart, SearchOrder:=xlByRows, SearchDirection:=xlNext, MatchCase:= _
            False).Activate
        ActiveCell.Offset(0, 1).Select
        TextBox2 = ActiveCell
        ActiveCell.Offset(0, 1).Select
        TextBox3 = ActiveCell
        Rem la etiqueta 4 toma el valor del renglón activo y permite modificar la
            información que encontró, ya que modifiques la información presionas el
            botón actualizar.
        Label4 = ActiveCell.Row
        noencontro:
End Sub
```

```
Private Sub CommandButton3_Click()
        Rem Vuelve a indicar el renglón 9 para escribir en los Textboxs
        Label4 = "9"
        Range("a9").Select
        TextBox1 = Empty
        TextBox2 = Empty
        TextBox3 = Empty
        TextBox1.SetFocus
End Sub

Private Sub TextBox1_Change()
        Rem si nos damos cuenta la etiqueta 4 sirve para llevar el renglón donde
            introducimos los datos  o los modificamos, asi que cada textbox que
            programemos debe llevar estas lineas.
        Range("A" + Label4).FormulaR1C1 = TextBox1
End Sub

Private Sub TextBox2_Change()
        Range("B" + Label4).FormulaR1C1 = TextBox2
End Sub

Private Sub TextBox3_Change()
        Range("C" + Label4).FormulaR1C1 = TextBox3
End Sub
```

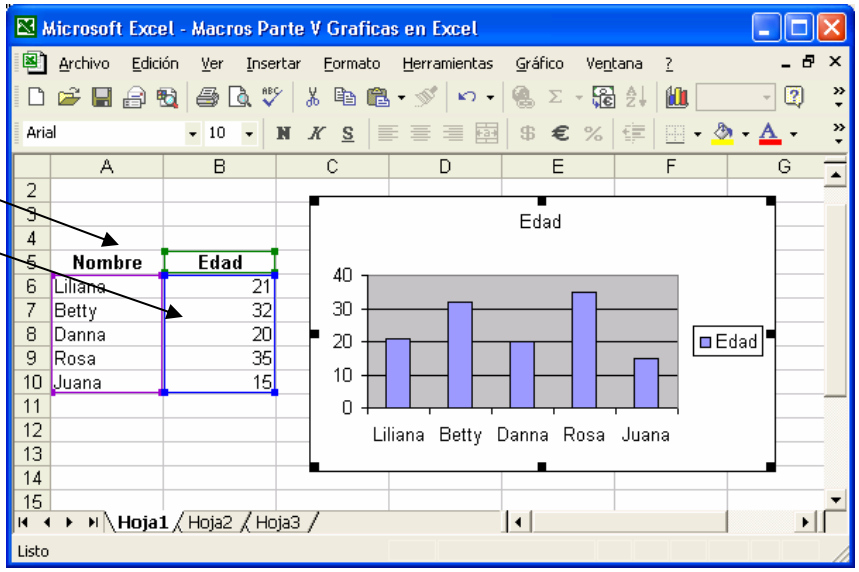
MACROS EN EXCEL (Parte V)



Hoy aprenderemos a trabajar con gráficos en Excel, veremos como se puede generar una grafica desde un código generado y alterado por nosotros mismos.

Si observamos los datos que vamos a graficar nos damos cuenta que en la columna **A** se encuentran los **valores ejes (X)** y en la columna **B** los **valores series (Y)**, estos datos son necesario para efectuar una grafica que podría quedar así

Esta grafica muestra las edades de 5 personas, los nombres son **los valores ejes** y la edad **los valores series**, ahora veremos como se puede detectar estos datos por medio de una Macro



Al graficar estos datos se genero el siguiente código:

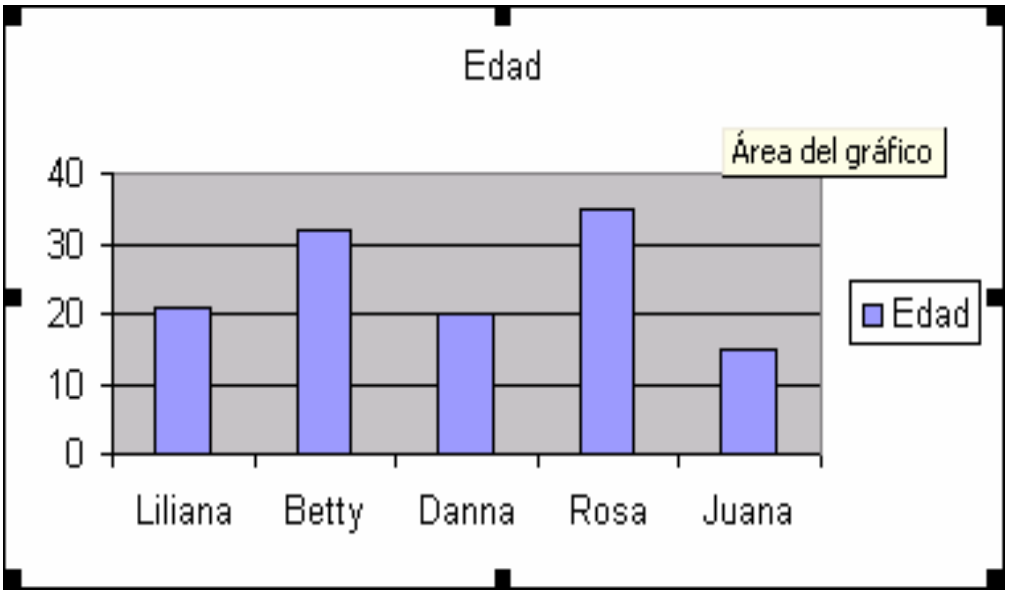
```
Sub Macro1()
    Range("A5:B10").Select
    Charts.Add
    ActiveChart.ChartType = xlColumnClustered
    ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _xlColumns
    ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
End Sub
```

1. La primer línea indica el rango donde están los datos, **valores ejes y valores series**,

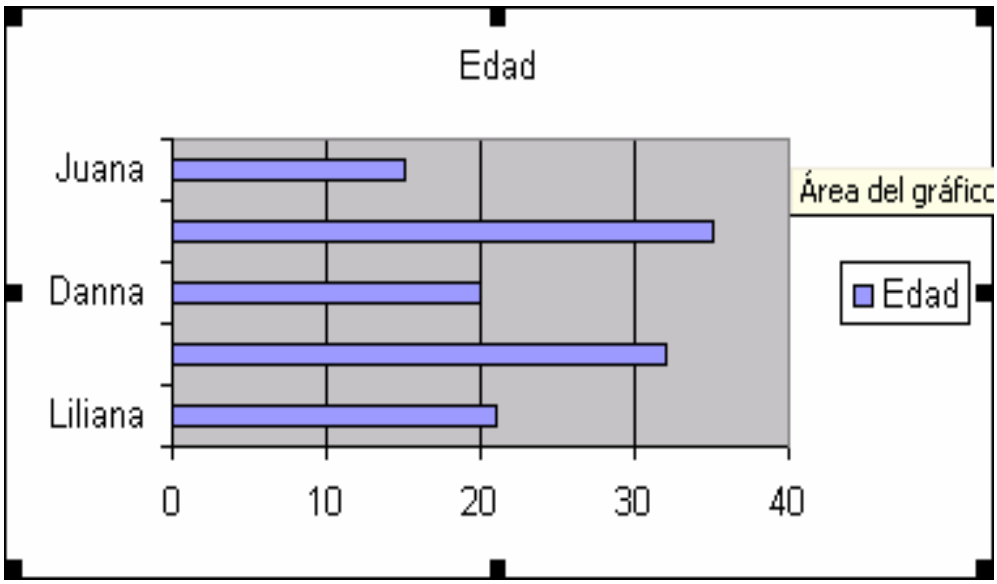
2. La segunda línea indica que se agrega una grafica
3. La tercera línea indica el tipo de grafica que se desea
4. La cuarta línea indica como se acomodan los datos en la grafica
5. La quinta línea indica donde se muestra la grafica, si en la misma hoja o en una sola hoja.

Nota. Los números de 1 al 5 no van en el código, solo los puse para poder explicar las líneas

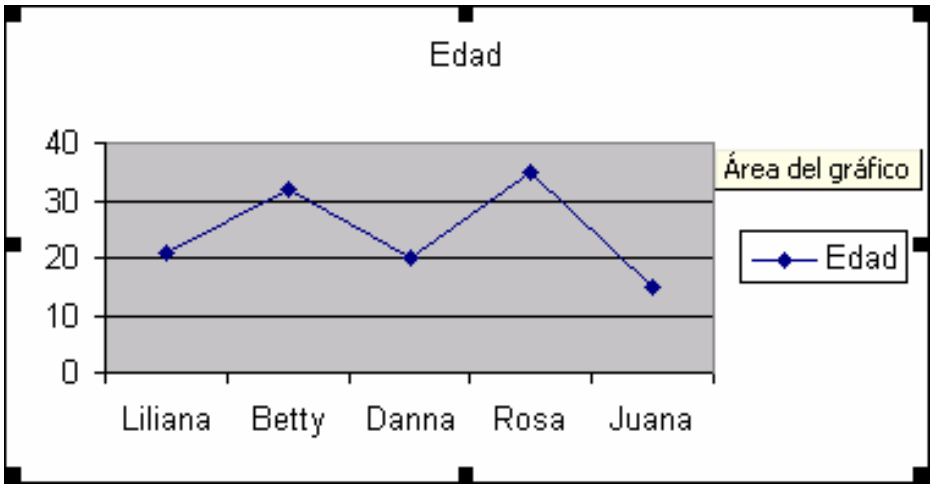
A continuación se muestran algunos de los diferentes tipos de graficas **Línea 3**:



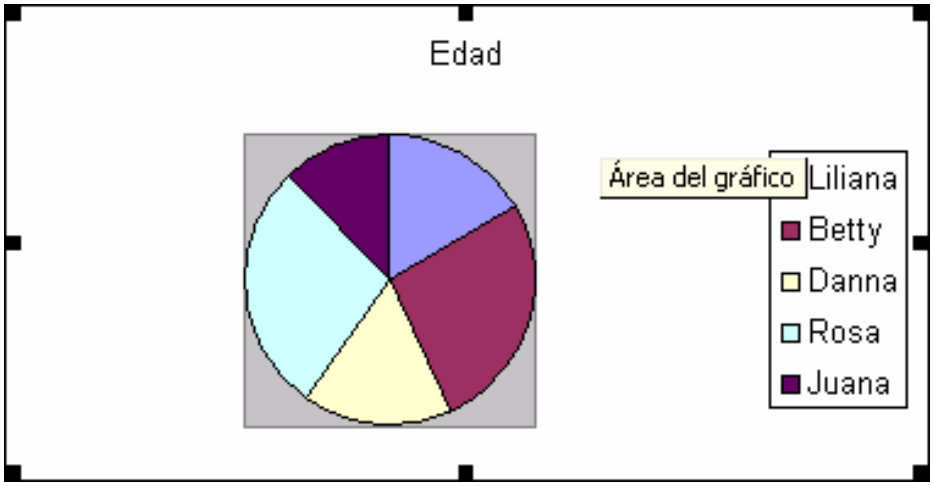
ActiveChart.ChartType = xlColumnClustered



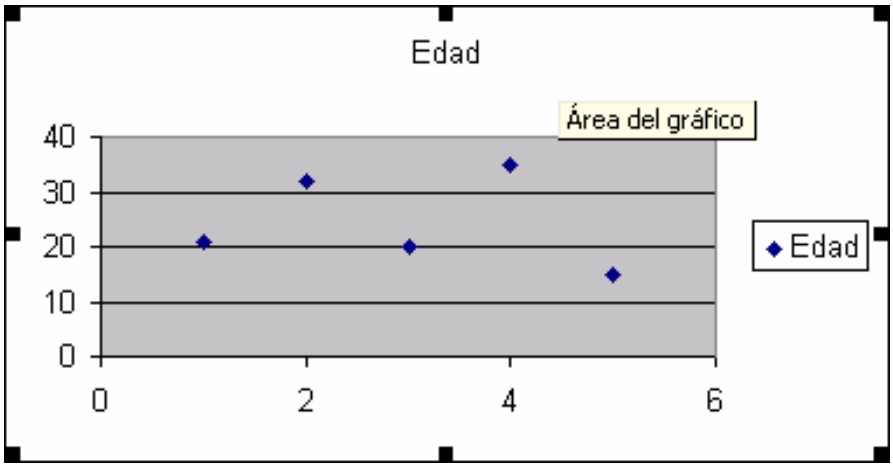
ActiveChart.ChartType = xlBarClustered



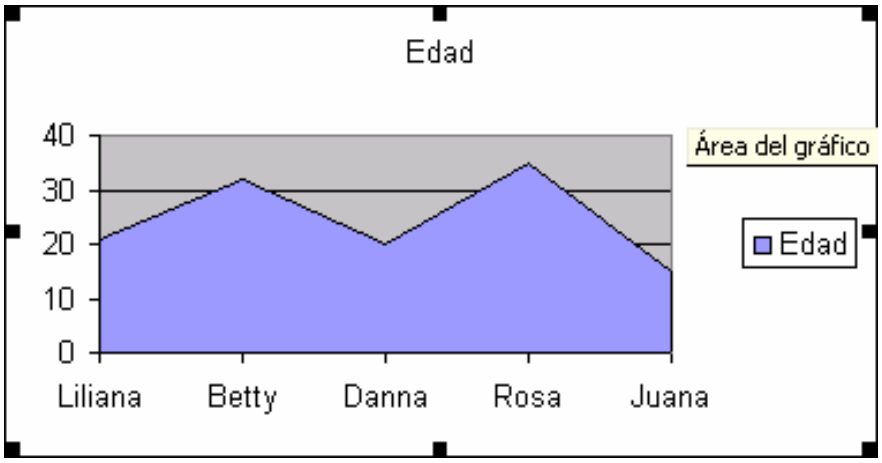
ActiveChart.ChartType = xlLineMarkers



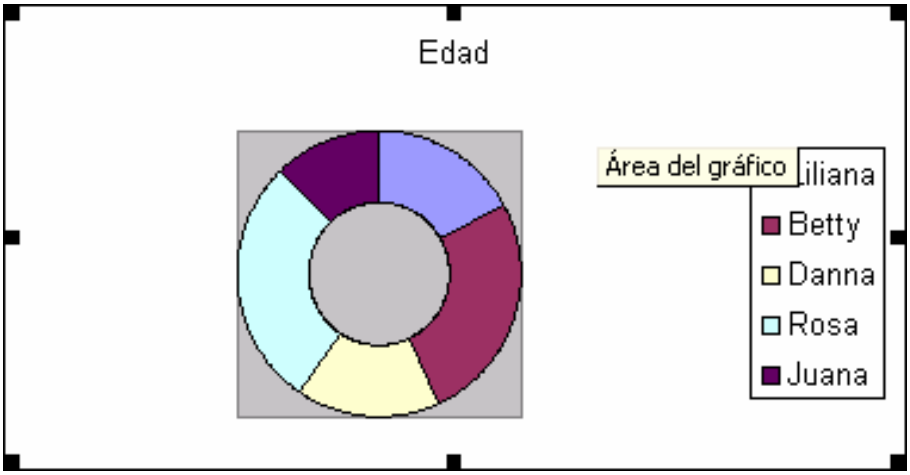
ActiveChart.ChartType = xlPie



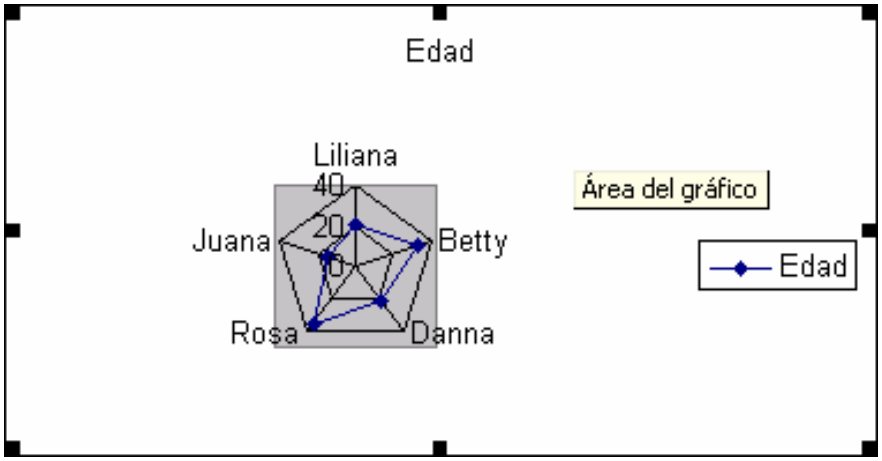
ActiveChart.ChartType = xlXYScatter



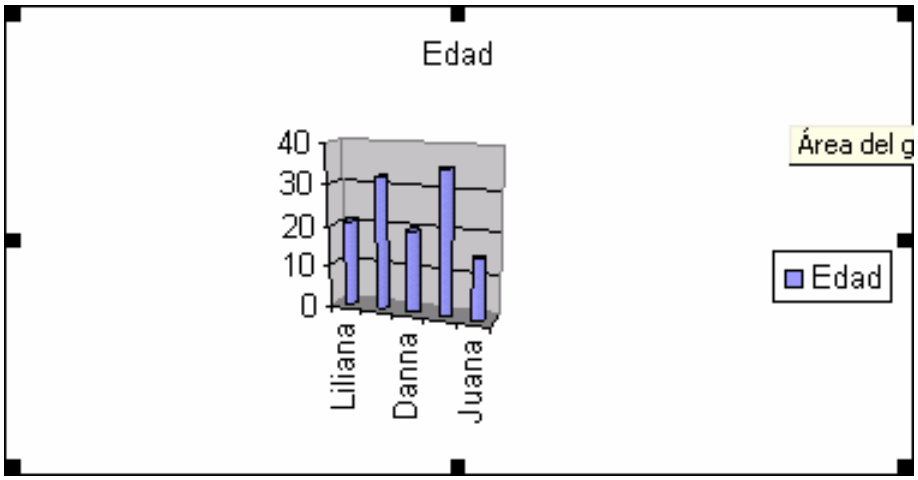
ActiveChart.ChartType = xlAreaStacked



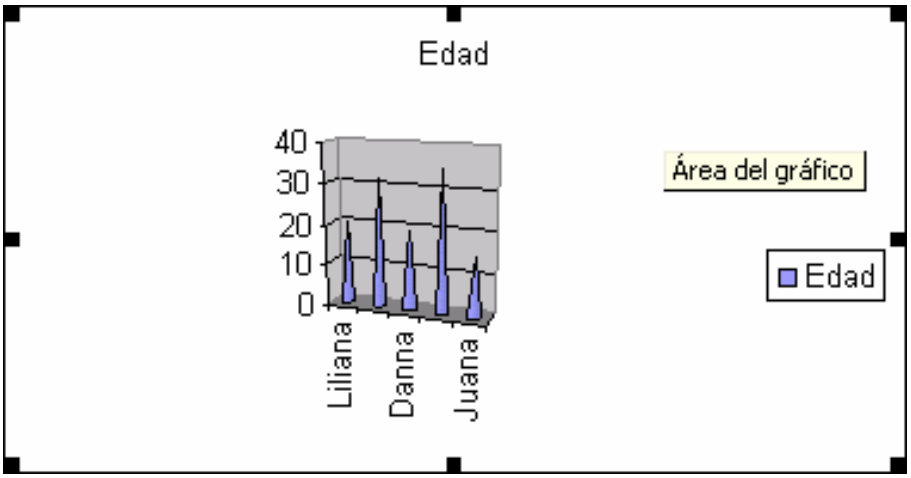
ActiveChart.ChartType = xlDoughnut



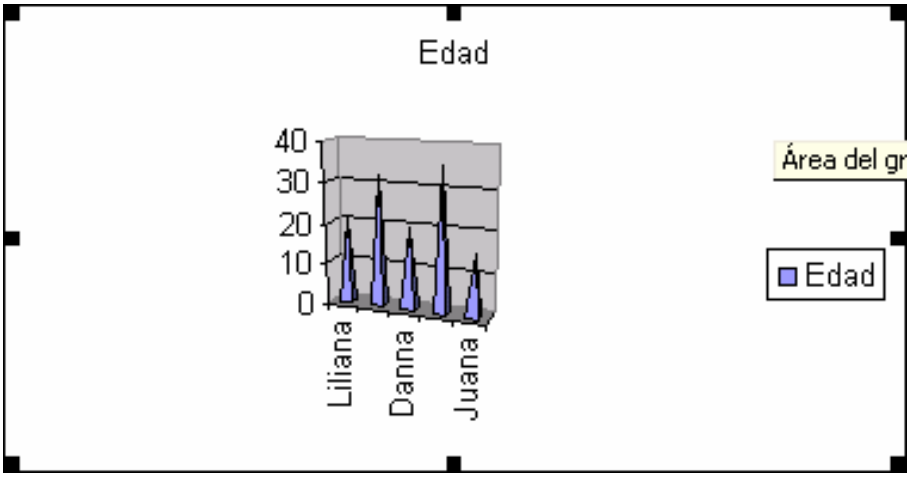
ActiveChart.ChartType = xlRadarMarkers



ActiveChart.ChartType = xlCylinderColClustered



ActiveChart.ChartType = xlConeColClustered



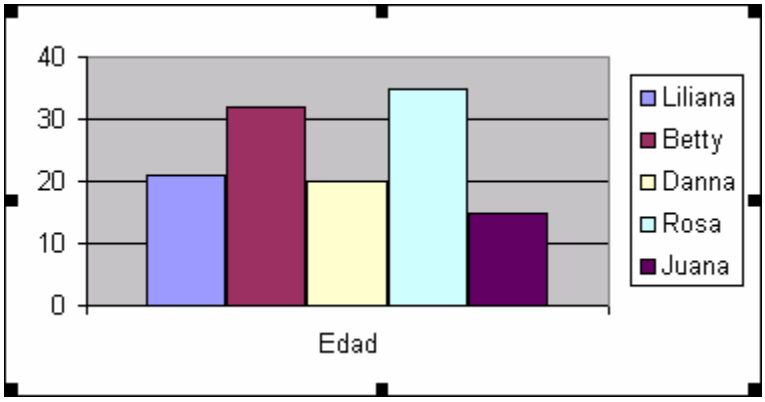
ActiveChart.ChartType = xlPyramidColClustered

Si tu agregas al final del código principal alguna línea del tipo de grafico que te gusto, ese se activara, por ejemplo:

```
Sub Macro1()
        Range("A5:B10").Select
        Charts.Add
        ActiveChart.ChartType = xlColumnClustered
        ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _xlColumns
        ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
        ActiveChart.ChartType = xlPyramidColClustered.
End Sub
```

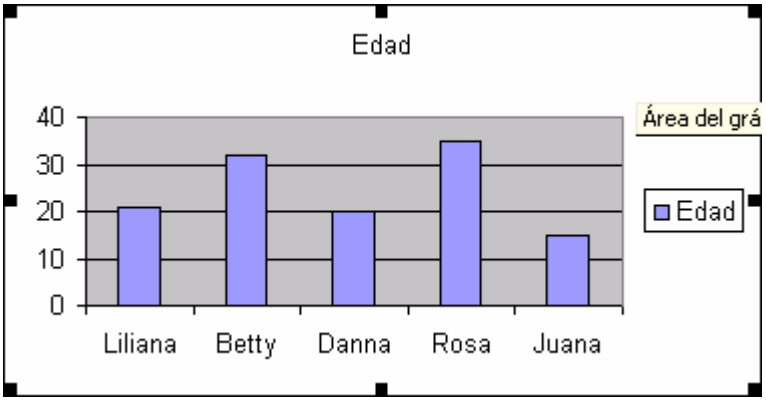
Este código se puede programar en un botón o cualquier otro control de **Visual Basic**.

A continuación se muestra como se acomodan los datos **Línea 4:**



ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _ xlRows

En esta línea se muestra la grafica por **Renglón**

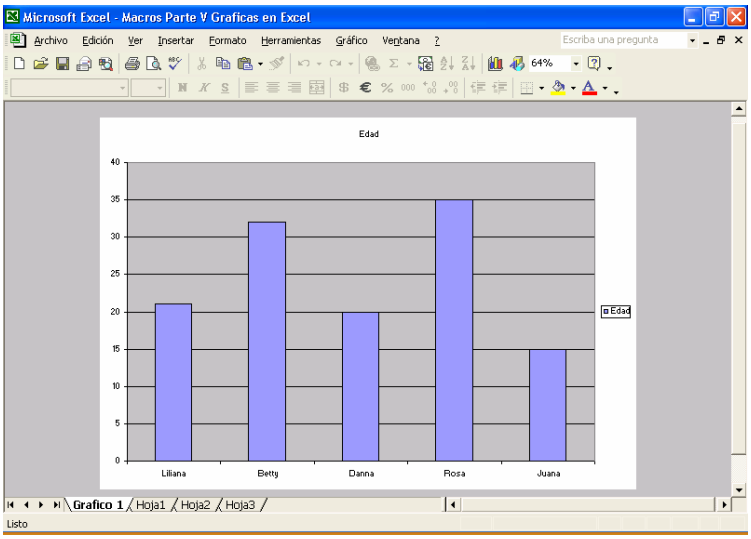


ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _ xlColumns

En esta línea se muestra la grafica por **Columna**

Esta es la forma en que se muestran los datos de lo que habla la **línea 4.**

La **línea 5** habla de que si la grafica queda en la misma hoja o simplemente toma una hoja para ella, por ejemplo:



ActiveChart.Location Where:=xlLocationAsNewSheet, Name:="Grafico 1"

Esta línea indica que la grafica tenga su propia hoja y que su nombre sea **Grafico 1.**

En este ejemplo ejecuto un código con cada una de las características explicadas en las **5 líneas.**

```
Range("A5:B10").Select
Charts.Add
ActiveChart.ChartType = xlColumnClustered
ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _xlColumns
ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
```

```
ActiveChart.ChartType = xlPyramidColClustered
ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:=
xlColumns
ActiveChart.Location Where:=xlLocationAsNewSheet, Name:="Grafico 1"
```

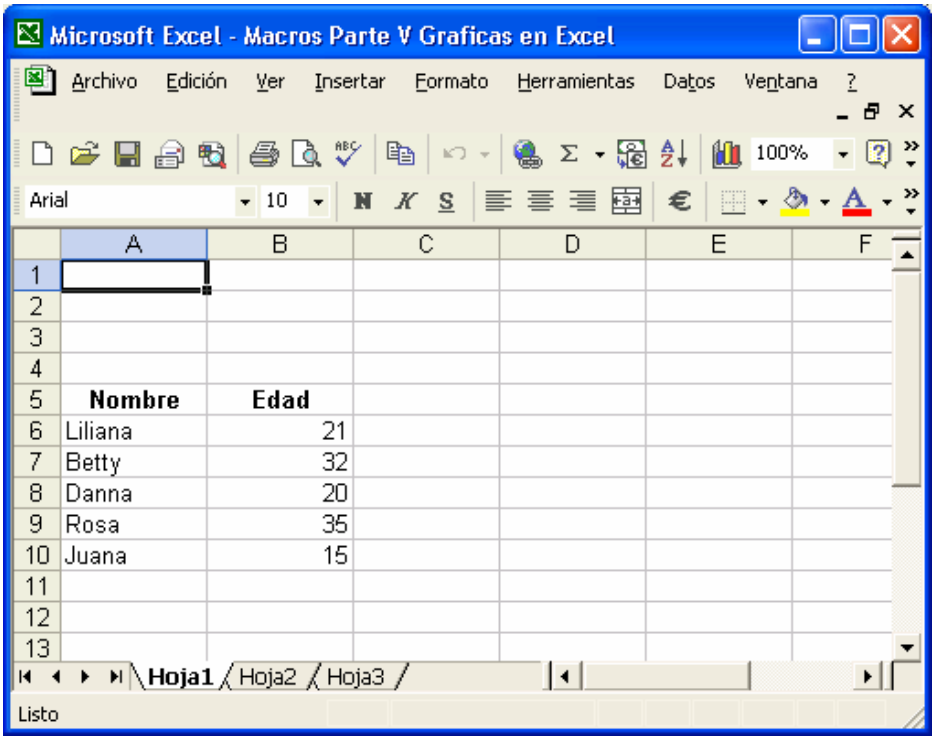
- **Tipo de Grafico**
- **Como se acomodan los datos**
- **Como se muestra la grafica, en este caso en una sola hoja**

Elabora el siguiente formulario con el siguiente código, para observar los diferentes tipos de gráficos y la forma en que se acomodan los datos:

Dibuja dos Listbox y un Botón y pega el código dentro del formulario.

```
Private Sub CommandButton1_Click()
        Rem este código genera la Grafica en la hoja1
        Range("A5:B10").Select
           Charts.Add
           ActiveChart.ChartType = xlColumnClustered
           ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _
               xlColumns
           ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
        Rem agrega los diferentes tipos de grafica al Listbox1
        ListBox1.AddItem "xlColumnClustered"
        ListBox1.AddItem "xlBarClustered"
        ListBox1.AddItem "xlLineMarkers"
        ListBox1.AddItem "xlPie"
        ListBox1.AddItem "xlXYScatter"
        ListBox1.AddItem "xlAreaStacked"
        ListBox1.AddItem "xlDoughnut"
        ListBox1.AddItem "xlRadarMarkers"
        ListBox1.AddItem "xlCylinderColClustered"
        ListBox1.AddItem "xlConeColClustered"
        ListBox1.AddItem "xlPyramidColClustered"
        Rem agrega las diferentes formas de acomodar los datos al Listbox2
        ListBox2.AddItem "Renglon"
        ListBox2.AddItem "Columna"
End Sub

Private Sub ListBox1_Click()
        Rem este código da el tipo de grafica al dar clic en el Listbox1
        If ListBox1 = "xlColumnClustered" Then ActiveChart.ChartType =
            xlColumnClustered
        If ListBox1 = "xlBarClustered" Then ActiveChart.ChartType = xlBarClustered
        If ListBox1 = "xlLineMarkers" Then ActiveChart.ChartType = xlLineMarkers
```

```
        If ListBox1 = "xlPie" Then ActiveChart.ChartType = xlPie
        If ListBox1 = "xlXYScatter" Then ActiveChart.ChartType = xlXYScatter
        If ListBox1 = "xlAreaStacked" Then ActiveChart.ChartType = xlAreaStacked
        If ListBox1 = "xlDoughnut" Then ActiveChart.ChartType = xlDoughnut
        If ListBox1 = "xlRadarMarkers" Then ActiveChart.ChartType = xlRadarMarkers
        If ListBox1 = "xlCylinderColClustered" Then ActiveChart.ChartType =
            xlCylinderColClustered
        If ListBox1 = "xlConeColClustered" Then ActiveChart.ChartType =
            xlConeColClustered
        If ListBox1 = "xlPyramidColClustered" Then ActiveChart.ChartType =
            xlPyramidColClustered
End Sub

Private Sub ListBox2_Click()
        If ListBox2 = "Renglon" Then
           ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _
              xlRows
        End If

        If ListBox2 = "Columna" Then
           ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A5:B10"), PlotBy:= _
              xlColumns
        End If
End Sub
```

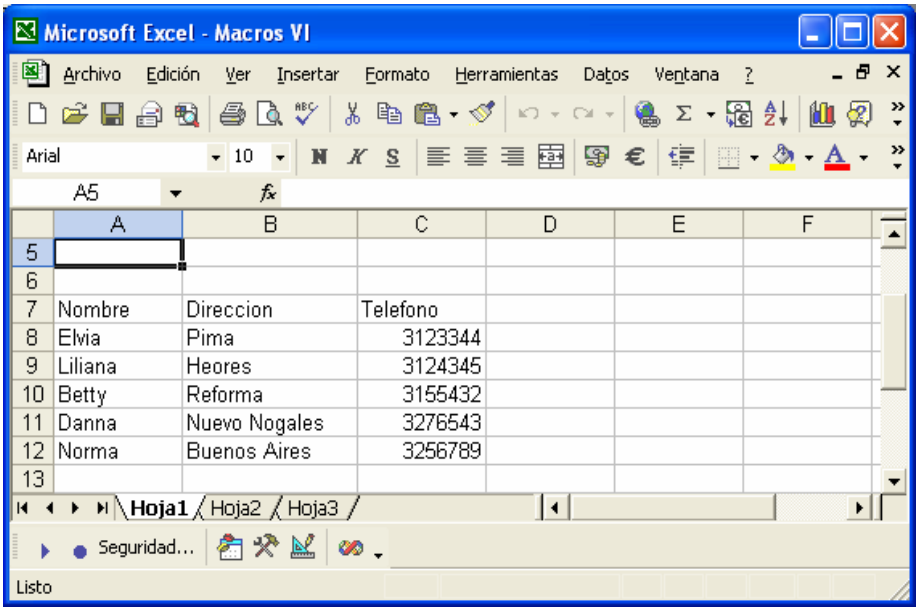


Antes de ejecutar esta **Macro** llenas los datos anteriores en la hoja1 de **Excel**

MACROS EN EXCEL (Parte VI)

Bienvenidos amigos a la sexta parte de **Macros en Excel y Visual Basic**, estamos listos para ver mas sobre este interesante curso, en este caso veremos como se pueden archivar los datos de una hoja en un archivo aparte. Aprenderemos a trabajar con **archivos secuénciales** en **Visual Basic**. Los **archivos secuénciales** son aquellos que al registrar sus datos llevan una secuencia, por ejemplo si registro 5 nombres llevaran un orden del 1 al 5, en cambio existen también los **archivos aleatorios**, pero ellos no respetan la secuencia, por ejemplo los 5 nombres podrían quedar en cualquier posición del 100 en adelante, del 300 en adelante, del 10 en adelante, de donde quieras ponerlos, tu indicas en donde quieres que queden los 5 nombres, pueden quedar hasta separados y no respetar una secuencia.

El problema de los **archivos secuénciales**, es que si introduces algunos símbolos en la captura pueden alterar el archivo y no funcionar correctamente, por eso se recomienda filtrar los datos con algún código o simplemente no capturar símbolos.



	A	B	C
7	Nombre	Direccion	Telefono
8	Elvia	Pima	3123344
9	Liliana	Heores	3124345
10	Betty	Reforma	3155432
11	Danna	Nuevo Nogales	3276543
12	Norma	Buenos Aires	3256789

En esta hoja podemos observar 5 nombres, la intención será archivarlos aparte y hacerlos desparecer de la hoja, para después volverlos aparecer en la hoja. A esto se le llamara Registro de datos y Consulta de datos.

Iremos a **Visual Basic** con **Alt+F11** y Insertaremos un **UserForm,** en el cual dibujaremos dos botones, uno con el nombre de **Registro** y Otro con el Nombre de **Consulta**.

Ahora a programar el botón Registro, para poder archivar los nombres.

```
Private Sub CommandButton1_Click()
    Rem se translada a la celda a8
    Range("a8").Select
    Rem si no hay ningún dato en a8 que no archive de nuevo
    If ActiveCell = Empty Then GoTo salte
```

```
        Rem abre un archivo en la unidad c con el nombre de datos.txt
        Rem en forma de añadir Temporal (Output) en el área de almacenamiento #1
        Open "c:\datos.txt" For Output As 1
        Rem activa una etiqueta para poder regresar
        regresa:
        Rem escribe el dato de la celda activa en el archivo
        Write #1, ActiveCell
        Rem borra el dato de la celda
        ActiveCell = Empty
        Rem baja un renglón para el siguiente nombre
        ActiveCell.Offset(1, 0).Select
        Rem si la celda esta vacía que no regrese ya
        If ActiveCell = Empty Then GoTo salte
        Rem regresa a escribir el siguiente nombre en el archivo
        GoTo regresa:

        salte:
        Rem se acabo
        Rem cierra el archivo
        Close #1
End Sub
```

Los datos quedaran archivados en la unidad y serán devueltos cuando presiones el botón consulta. Que a continuación se muestra:

Ahora a programar el botón consulta

```
Private Sub CommandButton2_Click()
        Rem se translada a la celda a8
        Range("a8").Select
        Rem abre un archivo en la unidad c con el nombre de datos.txt
        Rem en forma de Leer (input) en el área de almacenamiento #1
        Open "c:\datos.txt" For Input As 1
        Rem esto significa hazlo mientras no sea fin del archivo
        Rem esto quiere decir que no deje de leer los datos
        Rem hasta que no se llegue al ultimo de ellos
          Do While Not EOF(1)
            Rem lee un dato
            Input #1, nombre
            Rem lo escribe en la celda
            ActiveCell.FormulaR1C1 = nombre
            Rem baja un renglón para el siguiente nombre
            ActiveCell.Offset(1, 0).Select
        Rem activa el ciclo Do While-que regrese hasta
        Rem que se cumpla la condición
        Loop
```

```
    Rem cierra el archivo
    Close #1
End Sub
```

Que te parece archivar los datos aparte sin que nadie pueda observarlos, esta es la magia de los archivos secuénciales. Este ejemplo viene indexado en un archivo con el nombre de **Macros VI**.

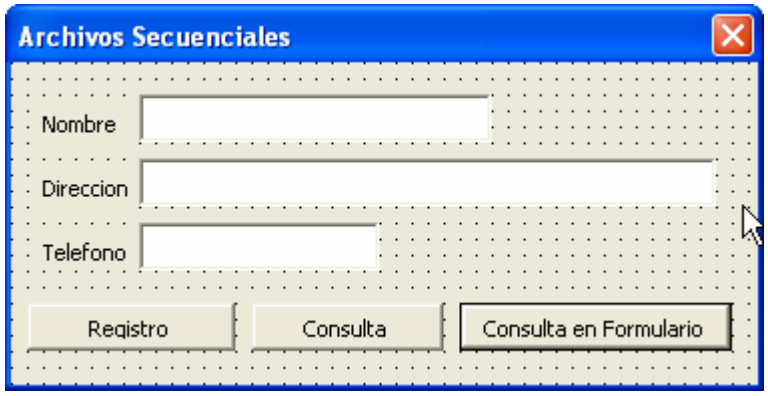
El siguiente código archiva el nombre, la dirección y el teléfono en el archivo, crea un formulario igual con dos botones.

```
Private Sub CommandButton1_Click()
    Rem se traslada a la celda a8
    Range("a8").Select
    Rem si no hay ningún dato en a8 que no archive de nuevo
    If ActiveCell = Empty Then GoTo salte
    Rem abre un archivo en la unidad c con el nombre de datos.txt
    Rem en forma de añadir Temporal(output) en el área de almacenamiento #1
    Open "c:\datos.txt" For Output As 1
    Rem activa una etiqueta para poder regresar
    regresa:
    Rem captura el nombre en una variable
    nombre = ActiveCell
    Rem borra el dato de la celda
    ActiveCell = Empty
    Rem se mueve una columna a la derecha
    ActiveCell.Offset(0, 1).Select
    Rem captura la direccion en una variable
    direccion = ActiveCell
    Rem borra el dato de la celda
    ActiveCell = Empty
    Rem se mueve una columna a la derecha
    ActiveCell.Offset(0, 1).Select
    Rem captura el telefono en una variable
    telefono = ActiveCell
    Rem borra el dato de la celda
    ActiveCell = Empty
    Rem escribe los datos nombre, direccion y telefono en el archivo
    Write #1, nombre, direccion, telefono
    Rem baja un renglón para el siguiente nombre
    ActiveCell.Offset(1, 0).Select
    Rem retrocede dos columnas
    ActiveCell.Offset(0, -2).Select
    Rem si la celda esta vacía que no regrese ya
    If ActiveCell = Empty Then GoTo salte
    Rem regresa a escribir el siguiente nombre en el archivo
```

```
        GoTo regresa:

        salte:
        Rem se acabo
        Rem cierra el archivo
        Close #1
End Sub

Private Sub CommandButton2_Click()
        Rem se translada a la celda a8
        Range("a8").Select
        Rem abre un archivo en la unidad c con el nombre de datos.txt
        Rem en forma de Leer (input) en el área de almacenamiento #1
        Open "c:\datos.txt" For Input As 1
        Rem esto significa hazlo mientras no sea fin del archivo
        Rem esto quiere decir que no deje de leer los datos
        Rem hasta que no se llegue al ultimo de ellos
           Do While Not EOF(1)
              Rem lee los datos
              Input #1, nombre, direccion, telefono
              Rem escribe en la celda el nombre
              ActiveCell.FormulaR1C1 = nombre
              Rem se mueve una columna a la derecha
              ActiveCell.Offset(0, 1).Select
              Rem escribe en la celda la direccion
              ActiveCell.FormulaR1C1 = direccion
              Rem se mueve una columna a la derecha
              ActiveCell.Offset(0, 1).Select
              Rem escribe en la celda el telefono
              ActiveCell.FormulaR1C1 = telefono
              Rem baja un renglón para el siguiente nombre
              ActiveCell.Offset(1, 0).Select
              Rem retrocede dos columnas
              ActiveCell.Offset(0, -2).Select
           Rem activa el ciclo Do While-que regrese hasta
           Rem que se cumpla la condición
           Loop
        Rem cierra el archivo
        Close #1
End Sub
```

Este ejemplo viene en el archivo **Macros VI-2.**

También se puede consultar sin necesidad de leer los datos en la hoja, esto quiere decir leyendo directo del archivo y trayendo los datos al formulario, en el siguiente ejemplo, se programa el botón consulta en formulario.



Dibuja el siguiente formulario, los dos primeros botones es el mismo código anterior, pero el tercer botón incluye el siguiente código:

```
Private Sub CommandButton3_Click()
        Open "c:\datos.txt" For Input As 1
           Do While Not EOF(1)
              Input #1, nombre, direccion, telefono
                 If nombre = TextBox1 Then
                    TextBox2 = direccion
                    TextBox3 = telefono
                 End If
           Loop

        Close #1
End Sub
```

Este ejemplo viene en el archivo **Macros VI-3**

Solo corra el formulario y escriba el nombre que desea consultar y presione el tercer botón.

Usted podrá consultar cualquiera de los nombres que se encuentren dentro del archivo, sin necesidad de que existan en la hoja, claro esta que primero es necesario presionar el botón registro para archivarlos, pero después se pueden manipular.

Bueno espero que sea de su agrado esta parte y que practiquen mucho los archivos secuenciales.

ALGUNAS ACLARACIONES, EJERCICIOS Y PRÁCTICAS SOBRE EL LENGUAJE VBA (Parte VII)

PROCEDIMIENTOS O SUBRUTINAS

Un procedimiento **Sub** es una serie de instrucciones Visual Basic, encerradas entre un par de instrucciones **Sub** y **End Sub,** que realizan acciones específicas pero no devuelven ningún valor. Un procedimiento **Sub** puede aceptar argumentos, como constantes, variables o expresiones que le pasa el procedimiento que ha efectuado la llamada. Si un procedimiento **Sub** no tiene argumentos, la instrucción **Sub** debe incluir un par de paréntesis vacío.

El siguiente procedimiento **Sub** dispone de comentarios explicativos en cada línea.

```
'Declara un procedimiento llamado ObtenInformacion
'Este procedimiento Sub no acepta argumentos
Sub ObtenInformacion()
'Declara una variable de cadena llamada respuesta
Dim respuesta As String
' Asigna el valor que devuelve la funcion InputBox a la variable respuesta
respuesta = InputBox(Prompt:="¿Cómo se llama?")
      ' Instrucción condicional If...Then...Else
   If respuesta = Empty Then
      ' Llama a la función MsgBox
      MsgBox Prompt:="No ha escrito su nombre."
   Else
      ' Función MsgBox concatenada con la variable respuesta
      MsgBox Prompt:="Su nombre es " & respuesta
      ' Fin de la instrucción If...Then...Else
   End If
      ' Fin del procedimiento Sub
End Sub
```

REGLAS DE ASIGNACIÓN DE NOMBRES EN VISUAL BASIC

Para dar nombre a procedimientos, constantes, variables y argumentos en un módulo de Visual Basic han de seguirse las siguientes reglas:

- El primer carácter debe ser una letra.
- En el nombre no se pueden utilizar espacios, puntos (**.**), signos de interjección (**!**), ni los caracteres **@**, **&**, **$,** #.
- El nombre no puede tener más de 255 caracteres de longitud.
- Como regla general, no se deben usar nombres iguales a los de los procedimientos Function, instrucciones y métodos de Visual Basic. Al final puede terminar usando las mismas palabras clave que utiliza el lenguaje. Para utilizar una función intrínseca del lenguaje, o una instrucción o método, cuyo nombre coincide con uno de los nombres asignados, es preciso identificarlos explícitamente. Para ello se sitúa delante del nombre de

la función intrínseca, instrucción o método, el nombre de la biblioteca de tipos asociada. Por ejemplo, si utiliza una variable llamada Left, la única forma de utilizar la función **Left** es escribiendo VBA.Left.

- Los nombres no se pueden repetir dentro del mismo nivel de alcance. Por ejemplo, no se pueden declarar dos variables con el nombre edad dentro del mismo procedimiento. Sin embargo, se puede declarar una variable privada edad y una variable de nivel de procedimiento llamada edad dentro del mismo módulo.

Nota: Visual Basic no diferencia entre mayúsculas y minúsculas, pero respeta la forma en que se escriben las instrucciones de declaración de nombres.

OBJETOS, PROPIEDADES, MÉTODOS Y EVENTOS

Un objeto representa un elemento de una aplicación, como una hoja de cálculo, una celda, un diagrama, un formulario o un informe. En código de Visual Basic, un objeto debe identificarse antes de se pueda aplicar uno de los métodos del objeto o cambiar el valor de una de sus propiedades.

Una colección es un objeto que contiene varios objetos que normalmente, pero no siempre, son del mismo tipo. En Microsoft Excel, por ejemplo, el objeto **Workbooks** contiene todos los objetos **Workbook** abiertos. En Visual Basic, la colección **Forms** contiene todos los objetos **Form** existentes en una aplicación.

Los elementos de una colección se pueden identificar mediante su número o su nombre. Por ejemplo, en el siguiente procedimiento, Libro(1) identifica al primer objeto **Workbook** abierto.

```
Sub CierraPrimero()
   Libro(1).Close
End Sub
```

El siguiente procedimiento utiliza un nombre especificado como cadena para identificar un objeto **Form**.

```
Sub CierraForm()
   Forms("MiForm.frm").Close
End Sub
```

También es posible operar al mismo tiempo sobre toda una colección de objetos siempre que los objetos compartan métodos comunes. Por ejemplo, el siguiente procedimiento cierra todos los formularios abiertos.

```
Sub CierraTodos()
   Forms.Close
End Sub
```

Método es toda acción que puede realizar un objeto. Por ejemplo, **Add** es un método del objeto **ComboBox** ya que sirve para añadir un nuevo elemento a un cuadro combinado.

El siguiente procedimiento utiliza el método **Add** para añadir un nuevo elemento a un **ComboBox**.

```
Sub AñadeElemen(nuevoElemento as String)
   Combo1.Add nuevoElemento
End Sub
```

Propiedad es un atributo de un objeto que define una de las características del objeto, tal como su tamaño, color o localización en la pantalla, o un aspecto de su comportamiento, por ejemplo si está visible o activado. Para cambiar las características de un objeto, se cambia el valor de sus propiedades

Para dar valor a una propiedad, hay que colocar un punto detrás de la referencia a un objeto, después el nombre de la propiedad y finalmente el signo igual (=) y el nuevo valor de la propiedad. Por ejemplo, el siguiente procedimiento cambia el título de un formulario de Visual Basic dando un valor a la propiedad **Caption**.

```
Sub CambiaNombre(nuevoTitulo)
   miForm.Caption = nuevoTitulo
End Sub
```

Hay propiedades a las que no se puede dar valor. El tema de ayuda de cada propiedad indica si es posible leer y dar valores a la propiedad (lectura/escritura), leer sólo el valor de la propiedad (sólo lectura) o sólo dar valor a la propiedad (sólo escritura).

Se puede obtener información sobre un objeto devolviendo el valor de una de sus propiedades. El siguiente procedimiento utiliza un cuadro de diálogo para presentar el título que aparece en la parte superior del formulario activo en ese momento.

```
Sub NombreFormEs()
   formNonmbre = Screen.ActiveForm.Caption
   MsgBox formNombre
End Sub
```

Evento es toda acción que puede ser reconocida por un objeto, como puede ser el clic del *mouse* o la pulsación de una tecla y para la que es posible escribir código como respuesta. Los eventos pueden ocurrir como resultado de una acción del usuario o del código de l programa, también pueden ser originados por el sistema.

Devolver objetos

Cada aplicación tiene una forma de devolver los objetos que contiene. Sin embargo estos procedimientos no son siempre iguales, por ello debe consultar el tema de ayuda correspondiente al objeto o colección que está usando en la aplicación para determinar la forma de devolver el objeto.

DECLARACIÓN DE VARIABLES

Para declarar variables se utiliza normalmente una instrucción **Dim**. La instrucción de declaración puede incluirse en un procedimiento para crear una variable de nivel de procedimiento. O puede colocarse al principio de un módulo, en la sección Declarations, para crear una variable de nivel de módulo.

El siguiente ejemplo crea la variable NombreTexto y específicamente le asigna el tipo de datos String.

```
Dim NombreTexto As String
```

Si esta instrucción aparece dentro de un procedimiento, la variable NombreTexto se puede usar sólo en ese procedimiento. Si la instrucción aparece en la sección Declarations del módulo, la variable NombreTexto estará disponible en todos los procedimientos dentro del módulo, pero para los restantes módulos del proyecto. Para hacer que esta variable esté disponible para todos los procedimientos de un proyecto, basta con comenzar la declaración con la instrucción **Public**, tal y como muestra el siguiente ejemplo:

```
Public NombreTexto As String
```

Si desea más información sobre cómo dar nombre a sus variables, puede consultar la sección "Visual Basic Naming Rules" en la Ayuda de Visual Basic.

Las variables se pueden declarar como de uno de los siguientes tipos de datos: **Boolean**, **Byte**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String** (para cadenas de longitud variable), **String** * *longitud* (para cadenas de longitud fija), **Object**, o **Variant**. Si no se especifica el tipo de datos, el tipo de datos **Variant** es el predefinido. También es posible crear un tipo definido por el usuario empleando la instrucción **Type**. Si desea más información sobre tipos de datos puede consultar la sección "Tipo de datos Summary" en la Ayuda de Visual Basic.

Se pueden declarar varias variables en una instrucción. Para especificar el tipo de datos se debe incluir un tipo de datos para cada variable. En la siguiente instrucción se declaran las variables intX, intY, e intZ como del tipo **Integer**.

```
Dim intX As Integer, intY As Integer, intZ As Integer
```

En la siguiente instrucción, intX e intY se declaran como del tipo **Variant**; y sólo intZ se declara como del tipo **Integer**.

```
Dim intX, intY, intZ As Integer
```

No es necesario especificar el tipo de datos en la instrucción de declaración. Si se omite, la variable será del tipo **Variant**.

Utilizar la instrucción Public
La instrucción **Public** se puede utilizar para declarar variables públicas de nivel de módulo.

Public NombreTexto As String

Las variables públicas se pueden usar en cualquier procedimiento del proyecto. Si una variable pública se declara en un módulo estándar o en un módulo de clase, también se podrá usar en los proyectos referenciados por el proyecto en que se declara la variable pública.

Utilizar la instrucción Private
La instrucción **Private** se puede usar para declarar variables privadas de nivel de módulo.

Private MiNombre As String

Las variables Private pueden ser usadas únicamente por procedimientos pertenecientes al mismo módulo.

Nota: Cuando se utiliza a nivel de módulo, la instrucción **Dim** es equivalente a la instrucción **Private**. Sería aconsejable usar la instrucción **Private** para facilitar la lectura y comprensión del código.

Utilizar la instrucción Static
Cuando se utiliza la instrucción **Static** en lugar de la instrucción **Dim**, la variable declarada mantendrá su valor entre llamadas sucesivas.

Utilizar la instrucción Option Explicit
En Visual Basic se puede declarar implícitamente una variable usándola en una instrucción de asignación. Todas las variables que se definen implícitamente son del tipo **Variant**. Las variables del tipo **Variant** consumen más recursos de memoria que la mayor parte de las otros tipos de variables. Su aplicación será más eficiente si se declaran explícitamente las variables y se les asigna un tipo de datos específico. Al declararse explícitamente las variables se reduce la posibilidad de errores de nombres y el uso de nombres erróneos.

Si no desea que Visual Basic realice declaraciones implícitas, puede incluir en un módulo la instrucción **Option Explicit** antes de todos los procedimientos. Esta instrucción exige que todas las variables del módulo se declaren explícitamente. Si un módulo incluye la instrucción **Option Explicit**, se producirá un error en tiempo de compilación cuando Visual Basic encuentre un nombre de variable que no ha sido previamente declarado, o cuyo nombre se ha escrito incorrectamente.

Se puede seleccionar una opción del entorno de programación de Visual Basic para incluir automáticamente la instrucción **Option Explicit** en todos los nuevos módulos. Consulte la documentación de su aplicación para encontrar la forma de modificar las opciones de entorno de Visual Basic. Tenga en cuenta que esta opción no tiene ningún efecto sobre el código que se haya escrito con anterioridad.

Nota: Las matrices fijas y dinámicas siempre se tiene que declarar explícitamente.

Declarar una variable de objeto para automatización

Cuando se utiliza una aplicación para controlar los objetos de otra aplicación, debe establecerse una referencia a la biblioteca de tipos de la otra aplicación. Una vez que se ha establecido la referencia, se pueden declarar variables de objeto conforme a su tipo más específico. Por ejemplo, si desde Microsoft Word se establece una referencia a la biblioteca de tipos de Microsoft Excel, se puede declarar una variable del tipo **Worksheet** desde Microsoft Word para representar un objeto **Worksheet** de Microsoft Excel.

Si se utiliza otra aplicación para controlar objetos de Microsoft Access, es posible, en la mayor parte de los casos, declarar variables objetos del tipo más específico. Se puede usar también la palabra clave **New** para crear automáticamente una nueva definición de un objeto. Sin embargo, puede ser necesario indicar que se trata de un objeto Microsoft Access. Por ejemplo, cuando se declara una variable de objeto para representar un formulario de Microsoft Access desde Microsoft Visual Basic, debe distinguirse entre el objeto **Form** de Microsoft Access y un objeto **Form** de Visual Basic. Para ello se incluye el nombre de la biblioteca de tipos en la declaración de la variable, como muestra el siguiente ejemplo:

```
Dim frmPedidos As New Access.Form
```

Algunas aplicaciones no reconocen algunos de los tipos de objetos de Microsoft Access. En ese caso, incluso después de establecer una referencia a la biblioteca de tipos de Microsoft Access, será necesario declarar todas las variables objeto de Microsoft Access como del tipo **Object**. Tampoco puede usarse la palabra clave **New** para crear una nueva definición del objeto. El siguiente ejemplo muestra cómo declarar una variable que represente una nueva definición del objeto **Application** de Microsoft Access desde una aplicación que no reconoce los tipos de objeto de Microsoft Access. La aplicación crea entonces una nueva definición del objeto **Application**.

```
Dim appAccess As Object
Set appAccess = CreateObject("Access.Application")
```

Para determinar la sintaxis a utilizar con una aplicación determinada debe consultarse la documentación de la aplicación.

DECLARACIÓN DE MATRICES Y ARREGLOS DE VARIABLES ESTÁTICOS Y DINÁMICOS

Las matrices se declaran igual que las restantes variables, utilizando instrucciones **Dim**, **Static**, **Private**, o **Public**. La diferencia entre las variables escalares (aquellas que no son matrices) y las variables matriz es que normalmente se debe especificar el tamaño de la matriz. Una matriz con un tamaño especificado es una matriz de tamaño fijo. Una matriz cuyo tamaño puede cambiar mientras el programa se está ejecutando es una matriz dinámica.

Si una matriz se indexa desde 0 ó desde 1 depende del valor de la instrucción **Option Base**. Si **Option Base 1** no se especifica, todos los índices de matrices comienzan en cero.

Declarar una matriz fija
En la siguiente línea de código se declara como matriz **Integer** una matriz de tamaño fijo con 11 filas y 11 columnas:

```
Dim MiMatriz(10, 10) As Integer
```

El primer argumento corresponde al número de filas y el segundo al número de columnas.

Como sucede en cualquier otra declaración de variable, a menos que se especifique para la matriz un tipo de datos, los elementos de ésta serán del tipo **Variant**. Cada elemento numérico **Variant** de la matriz utiliza 16 bytes. Cada elemento de cadena **Variant** utiliza 22 bytes. Para escribir código de la forma más compacta posible, debe declarar explícitamente sus matrices con un tipo de datos distinto a **Variant**. Las siguientes líneas de código comparan el tamaño de varias matrices:

```
' Una matriz Integer utiliza 22 bytes (11 elementos * 2 bytes).
ReDim MiMatrizInteger(10) As Integer

' Una matriz Double-precision utiliza 88 bytes (11 elementos * 8 bytes).
ReDim MiMatrizDoble(10) As Double

' Una matriz Variant utiliza al menos 176 bytes (11 elementos * 16 bytes).
ReDim MiMatrizVariant(10)

' La matriz Integer utiliza 100 * 100 * 2 bytes (20.000 bytes).
ReDim MiMatrizInteger(99, 99) As Integer

' La matriz Double-precision utiliza 100 * 100 * 8 bytes (80.000 bytes).
ReDim MiMatrizDoble (99, 99) As Double

' La matriz Variant utiliza al menos 160.000 bytes (100 * 100 * 16 bytes).
ReDim MiMatrizVariant(99, 99)
```

El tamaño máximo de una matriz depende del sistema operativo y de la cantidad de memoria disponible. Es más lento utilizar una matriz que sobrepasa la cantidad de memoria RAM disponible en el sistema ya que los datos tienen que ser leídos y escritos del disco.

Declarar una matriz dinámica
Al declarar una matriz dinámica se puede cambiar el tamaño de una matriz mientras que el código se está ejecutando. Para declarar una matriz dinámica se usan las instrucciones **Static**, **Dim**, **Private**, o **Public**, dejando los paréntesis vacíos, tal y como se muestra en el siguiente ejemplo.

```
Dim MatrizSingle() As Single
```

Nota: Se puede usar la instrucción **ReDim** para declarar implícitamente una matriz dentro de un procedimiento. Tenga cuidado para no cambiar el nombre de la matriz cuando use la instrucción **ReDim**, ya que se creará una segunda matriz incluso en el caso de que se haya incluido la instrucción **Option Explicit** en el módulo.

La instrucción **ReDim** se puede utilizar en un procedimiento, dentro del alcance de la matriz, para cambiar el número de dimensiones, definir el número de elementos y para definir los límites superior e inferior para cada dimensión. Se puede usar la instrucción **ReDim** para modificar la matriz dinámica cuantas veces sea necesario. Sin embargo, cada vez que se hace, se pierden los valores almacenados en la matriz. Se puede usar la instrucción **ReDim Preserve** para ampliar una matriz conservando los valores que contiene. Por ejemplo, la siguiente instrucción añade 10 nuevos elementos a la matriz MatrizVar sin perder los valores almacenados en los elementos originales.

```
ReDim Preserve MatrizVar(UBound(MatrizVar) + 10)
```

Nota Cuando se utiliza la palabra clave **Preserve** con una matriz dinámica, sólo se puede cambiar el límite superior de la última dimensión, no pudiendo modificarse el número de dimensiones.

DECLARACIÓN DE CONSTANTES

Al declarar una constante, se puede asignar a un valor un nombre que tenga algún significado apropiado. La instrucción **Const** se utiliza para declarar una constante y darle valor. Una constante no puede modificarse o cambiar de valor una vez que ha sido declarada.

Se puede declarar una constante dentro de un procedimiento o al principio de un módulo, en la sección de Declarations. Las constantes a nivel de módulo son privadas, a menos que se especifique lo contrario. Para declarar una constante pública a nivel de módulo, la instrucción **Const** debe ir precedida por la palabra clave **Public**. Se puede declarar explícitamente una constante como privada colocando la palabra clave **Private** antes de la instrucción **Const** para facilitar la lectura y comprensión del código. Si desea más información, consulte la sección "Comprender el alcance y la visibilidad" en la Ayuda de Visual Basic.

El siguiente ejemplo declara la constante **Public** EdadCon como un **Integer** y le asigna el valor 34.

```
Public Const EdadCon As Integer = 34
```

Las constantes se pueden declarar de uno de los siguientes tipos de datos: **Boolean**, **Byte**, **Integer**, **Long**, **Currency**, **Single**, **Double**, **Date**, **String**, o **Variant**. Dado que ya se conoce el valor de una constante, es muy fácil elegir el tipo de datos en la instrucción **Const**. Si desea más información sobre tipos de datos, consulte la sección "Tipo de datos Summary" en la Ayuda de Visual Basic.

En una sola instrucción se pueden declarar varias constantes. Para especificar un tipo de datos, debe incluirse el tipo de datos para cada constante. En la siguiente instrucción se declaran como **Integer** las constantes EdadCon y SalarioCon.

Const EdadCon As Integer = 34, SalarioCon As Currency = 35000

Crear variables de objeto
Se puede crear una variable de objeto de la misma forma que el objeto al que hace referencia. Se pueden activar o devolver las propiedades del objeto o utilizar cualquiera de sus métodos.

Para crear una variable de objeto:
1. Declare la variable de objeto.
2. Asigne la variable de objeto a un objeto.

DECLARACIÓN DE VARIABLES DE OBJETO

Para declarar una variable de objeto se ha de usar la instrucción **Dim** o una de las restantes instrucciones de declaración (**Public**, **Private**, o **Static**). Una variable que se refiere a un objeto debe ser una **Variant**, un **Object**, o un tipo específico de objeto. Por ejemplo, son válidas las siguientes declaraciones:

```
' Declara MiObjeto como tipo de datos Variant.
Dim MiObjeto
' Declara MiObjeto como un tipo de datos Object.
Dim MiObjeto As Object
' Declara MiObjeto como un tipo Font.
Dim MiObjeto As Font
```

Nota Si utiliza una variable de objeto sin haberla declarado previamente, el tipo de datos predefinido de la variable de objeto es **Variant**.

Se puede declarar una variable de objeto con el tipo de datos **Object** cuando el tipo de objeto específico no se conoce hasta que se ejecuta el procedimiento. Utilice el tipo de datos **Object** para crear una referencia genérica a cualquier objeto.

Si conoce el tipo específico de objeto, debe declarar así la variable de objeto. Por ejemplo, si la aplicación contiene un tipo de objeto Ejemplo, se puede declarar una variable de objeto para ese objeto empleando una cualquiera de las dos instrucciones siguientes:

```
Dim MiObjeto As Object    ' Se declara como objeto genérico.
Dim MiObjeto As Ejemplo   ' Se declara sólo como un objeto Ejemplo.
```

Al declarar objetos específicos es posible comprobar automáticamente los tipos, el código es más rápido de ejecución y mejora su legibilidad.

Asignar una variable de objeto a un objeto

Para asignar una variable de objeto a un objeto se utiliza la instrucción **Set**. Es posible asignar una expresión de objeto o **Nothing**. Por ejemplo, son válidas las siguientes asignaciones a una variable de objeto:

```
Set MiObjeto = SuObjeto    ' Asigna referencia a objeto.
Set MiObjeto = Nothing    ' Deshace la relación.
```

Se puede efectuar al mismo tiempo la declaración de la variable de objeto con la asignación de un objeto a la misma, para ello se utiliza la palabra clave **New** en la instrucción **Set**. Por ejemplo:

```
Set MiObjeto = Nuevo Objeto    ' Crea y asigna
```

Al asignar a una variable de objeto el valor **Nothing** se deshace la relación que pudiera existir entre la variable de objeto y cualquier objeto específico. Así se evita que, accidentalmente, se pueda cambiar el objeto al cambiar la variable. Una variable de objeto queda definida siempre como **Nothing** al cerrar el objeto asociado, así es posible comprobar si la variable de objeto está asociada a un objeto válido. Por ejemplo:

```
If Not MiObjeto Is Nothing Then
   ' La variable hace referencia a un objeto válido.
   . . .
End If
```

Por supuesto, esta prueba nunca podrá determinar con absoluta certeza si un usuario ha cerrado o no la aplicación que contiene al objeto al que se hace referencia por la variable de objeto.

Hacer referencia a la definición actual de un objeto

Utilice la palabra clave **Me** para hacer referencia a la definición actual del objeto donde se está ejecutando el código. Todos los procedimientos asociados con el objeto actual tienen acceso al objeto al que se hace referencia como **Me**. La utilización de **Me** es especialmente útil para pasar información sobre la definición actual de un objeto a un procedimiento de otro módulo. Por ejemplo, suponga que existe el siguiente procedimiento en un módulo:

```
Sub CambiaColorObjeto(MiObjetoNombre As Object)
   MiObjetoNombre.ColorFondo = RGB(Rnd * 256, Rnd * 256, Rnd * 256)
End Sub
```

Se puede hacer una llamada al procedimiento y pasarle, como argumento, la definición actual del objeto empleando la siguiente instrucción:

```
CambiaColorObjeto Me
```

CREAR SECUENCIAS ITERATIVAS MEDIANTE CÓDIGO VISUAL BASIC

Mediante el uso de instrucciones condicionales y instrucciones de bucle (también conocidas como estructuras de control) es posible escribir código de Visual Basic que tome decisiones y repita determinadas acciones. Otra estructura de control útil, la instrucción **With**, permite ejecutar una serie de instrucciones sin necesidad de recalificar un objeto.

Utilizar instrucciones condicionales para tomar decisiones
Las instrucciones condicionales evalúan si una condición es **True** o **False** y a continuación especifican las instrucciones a ejecutar en función del resultado. Normalmente, una condición es una expresión que utiliza un operador de comparación para comparar un valor o variable con otro.

Elegir la instrucción condicional a utilizar

- If...Then...Else: Salto a una instrucción cuando una condición es **True** o **False**
- Select Case: Selección de la instrucción a ejecutar en función de un conjunto de condiciones

Utilizar bucles para repetir código
Empleando bucles es posible ejecutar un grupo de instrucciones de forma repetida. Algunos bucles repiten las instrucciones hasta que una condición es **False**, otros las repiten hasta que la condición es **True**. Hay también bucles que repiten un conjunto de instrucciones un número determinado de veces o una vez para cada objeto de una colección.

Elegir el bucle a utilizar

- Do...Loop: Seguir en el bucle mientras o hasta una condición sea **True**.
- For...Next: Utilizar un contador para ejecutar las instrucciones un número determinado de veces.
- For Each...Next: Repetición del grupo de instrucciones para cada uno de los objetos de una colección.

Ejecutar varias instrucciones sobre el mismo objeto
Normalmente, en Visual Basic, debe especificarse un objeto antes de poder ejecutar uno de sus métodos o cambiar una de sus propiedades. Se puede usar la instrucción **With** para especificar un objeto una sola vez para una serie completa de instrucciones.

- With: Ejecutar una serie de instrucciones sobre el mismo objeto

INSTRUCCIÓN If...Then...Else

Se puede usar la instrucción **If...Then...Else** para ejecutar una instrucción o bloque de instrucciones determinadas, dependiendo del valor de una condición. Las instrucciones **If...Then...Else** se pueden anidar en tantos niveles como sea necesario. Sin embargo, para hacer más legible el código es aconsejable utilizar una instrucción **Select Case** en vez de recurrir a múltiples niveles de instrucciones **If...Then...Else** anidadas.

Ejecutar una sola instrucción cuando una condición es True

Para ejecutar una sola instrucción cuando una condición es **True**, se puede usar la sintaxis de línea única de la instrucción **If...Then...Else**. El siguiente ejemplo muestra la sintaxis de línea única, en la que se omite el uso de la palabra clave **Else**:

```
Sub FijarFecha()
   miFecha = #13/2/95#
   If miFecha < Now Then miFecha = Now
End Sub
```

Para ejecutar más de una línea de código, es preciso utilizar la sintaxis de múltiples líneas. Esta sintaxis incluye la instrucción **End If**, tal y como muestra el siguiente ejemplo:

```
Sub AvisoUsuario(valor as Long)
   If valor = 0 Then
      Aviso.ForeColor = "Red"
      Aviso.Font.Bold = True
      Aviso.Font.Italic = True
   End If
End Sub
```

Ejecutar unas instrucciones determinadas si una condición es True y ejecutar otras si es False

Use una instrucción **If...Then...Else** para definir dos bloques de instrucciones ejecutables: un bloque que se ejecutará cuando la condición es **True** y el otro que se ejecutará si la condición es **False**.

```
Sub AvisoUsuario(valor as Long)
   If valor = 0 Then
      Aviso.ForeColor = vbRed
      Aviso.Font.Bold = True
      Aviso.Font.Italic = True
   Else
      Aviso.Forecolor = vbBlack
      Aviso.Font.Bold = False
      Aviso.Font.Italic = False
   End If
End Sub
```

Comprobar una segunda condición si la primera condición es False

Se pueden añadir instrucciones **ElseIf** a una instrucción **If...Then...Else** para comprobar una segunda condición si la primera es **False**. Por ejemplo, el siguiente procedimiento función calcula una bonificación salarial dependiendo de la clasificación del trabajador. La instrucción que sigue a la instrucción **Else** sólo se ejecuta cuando las condiciones de todas las restantes instrucciones **If** y **ElseIf** son **False**.

```
Function Bonificación(rendimiento, salario)
   If rendimiento = 1 Then
      Bonificación = salario * 0.1
   ElseIf rendimiento = 2 Then
      Bonificación= salario * 0.09
   ElseIf rendimiento = 3 Then
      Bonificación = salario * 0.07
   Else
      Bonificación = 0
   End If
End Function
```

INSTRUCCIÓN Do...Loop

Se pueden usar instrucciones **Do...Loop** para ejecutar un bloque de instrucciones un número indefinido de veces. Las instrucciones se repiten mientras una condición sea **True** o hasta que llegue a ser **True**.

Repetir instrucciones mientras una condición es True
Hay dos formas de utilizar la palabra clave **While** para comprobar el estado de una condición en una instrucción **Do...Loop**. Se puede comprobar la condición antes de entrar en el bucle, o después de que el bucle se haya ejecutado al menos una vez.

En el siguiente procedimiento ComPrimeroWhile, la condición se comprueba antes de entrar en el bucle. Si miNum vale 9 en vez de 20, las instrucciones contenidas en el bucle no se ejecutarán nunca. En el procedimiento ComFinalWhile, las instrucciones contenidas en el bucle sólo se ejecutarán una vez antes de que la condición llegue a ser **False**.

```
Sub ComPrimeroWhile()
   contador = 0
   miNum = 20
   Do While miNum > 10
      miNum = miNum - 1
      contador = contador + 1
   Loop
   MsgBox "El bucle se ha repetido " & contador & " veces."
End Sub

Sub ComFinalWhile()
   contador = 0
   miNum = 9
   Do
      miNum = miNum - 1
      contador = contador + 1
   Loop While miNum > 10
   MsgBox "El bucle se ha repetido " & contador & " veces."
```

```
End Sub
```

Repetir instrucciones hasta que una condición llegue a ser True

Hay dos formas de utilizar la palabra clave **Until** para comprobar el estado de una condición en una instrucción **Do...Loop**. Se puede comprobar la condición antes de entrar en el bucle (como muestra el procedimiento ComPrimeroUntil) o se pueden comprobar después de que el bucle se haya ejecutado al menos una vez (como muestra el procedimiento ComFinalUntil). El bucle sigue ejecutándose mientras la condición siga siendo **False**.

```
Sub ComPrimeroUntil()
   contador = 0
   miNum = 20
   Do Until miNum = 10
      miNum = miNum - 1
      contador = contador + 1
   Loop
   MsgBox "El bucle se ha repetido " & contador & " veces."
End Sub

Sub ComFinalUntil()
   contador = 0
   miNum = 1
   Do
      miNum = miNum + 1
      contador = contador + 1
   Loop Until miNum = 10
   MsgBox "El bucle se ha repetido " & counter & " veces."
End Sub
```

Instrucción de salida de Do...Loop desde dentro del bucle

Es posible salir de **Do...Loop** usando la instrucción **Exit Do**. Por ejemplo, para salir de un bucle sin fin, se puede usar la instrucción **Exit Do** en el bloque de instrucciones **True** de una instrucción **If...Then...Else** o **Select Case**. Si la condición es **False**, el bucle seguirá ejecutándose normalmente.

En el siguiente ejemplo, se asigna a miNum un valor que crea un bucle sin fin. La instrucción **If...Then...Else** comprueba esa condición y ejecuta entonces la salida, evitando así el bucle sin fin.

```
Sub EjemploSalida()
   contador = 0
   miNum = 9
   Do Until miNum = 10
      miNum = miNum - 1
      contador = contador + 1
      If miNum < 10 Then Exit Do
   Loop
   MsgBox "El bucle se ha repetido " & contador & " veces."
End Sub
```

Nota Para detener la ejecución de un bucle sin fin, presione la tecla ESC o CTRL+PAUSE.

INSTRUCCIÓN For...Next

Las instrucciones **For...Next** se pueden utilizar para repetir un bloque de instrucciones un número determinado de veces. Los bucles **For** usan una variable contador cuyo valor se aumenta o disminuye cada vez que se ejecuta el bucle.

El siguiente procedimiento hace que el equipo emita un sonido 50 veces. La instrucción **For** determina la variable contador x y sus valores inicial y final. La instrucción **Next** incrementa el valor de la variable contador en 1.

```
Sub Bips()
   For x = 1 To 50
      Beep
   Next x
End Sub
```

Mediante la palabra clave **Step**, se puede aumentar o disminuir la variable contador en el valor que se desee. En el siguiente ejemplo, la variable contador j se incrementa en 2 cada vez que se repite la ejecución del bucle. Cuando el bucle deja de ejecutarse, total representa la suma de 2, 4, 6, 8 y 10.

```
Sub DosTotal()
   For j = 2 To 10 Step 2
      total = total + j
   Next j
   MsgBox "El total es " & total
End Sub
```

Para disminuir la variable contador utilice un valor negativo en **Step**. Para disminuir la variable contador es preciso especificar un valor final que sea menor que el valor inicial. En el siguiente ejemplo, la variable contador miNum se disminuye en 2 cada vez que se repite el bucle. Cuando termina la ejecución del bucle, total representa la suma de 16, 14, 12, 10, 8, 6, 4 y 2.

```
Sub NuevoTotal()
   For miNum = 16 To 2 Step -2
      total = total + miNum
   Next miNum
   MsgBox "El total es " & total
End Sub
```

Nota No es necesario incluir el nombre de la variable contador después de la instrucción **Next**. En los ejemplos anteriores, el nombre de la variable contador se ha incluido para facilitar la lectura del código.

Se puede abandonar una instrucción **For...Next** antes de que el contador alcance su valor final, para ello se utiliza la instrucción **Exit For**. Por ejemplo, si se produce un error se puede usar la instrucción **Exit For** en el bloque de instrucciones **True** de una instrucción **If...Then...Else** o **Select Case** que detecte específicamente ese error. Si el error no se produce, la instrucción **If…Then…Else** es **False** y el bucle continuará ejecutándose normalmente.

INSTRUCCIONES DE ASIGNACIÓN

Las instrucciones de asignación asignan un valor o expresión a una variable o constante. Las instrucciones de asignación incluyen siempre un signo igual (=). El siguiente ejemplo asigna el valor que devuelve la función **InputBox** a la variable suNombre.

```
Sub Pregunta()
   Dim suNombre As String
   suNombre = InputBox("¿Cómo se llama?")
   MsgBox "Su nombre es " & suNombre
End Sub
```

La instrucción **Let** es opcional y normalmente se omite. Por ejemplo, la instrucción de asignación anterior podría haberse escrito así:

```
Let suNombre = InputBox("¿Cómo se llama?").
```

La instrucción **Set** se utiliza para asignar un objeto a una variable que ha sido declarada como objeto. La palabra clave **Set** es necesaria. En el siguiente ejemplo, la instrucción **Set** asigna un rango de Hoja1 a la variable de objeto miCelda:

```
Sub DarFormato()
Dim miCelda As Range
Set miCelda = Worksheets("Hoja1").Range("A1")
   With miCelda.Font
      .Bold = True
      .Italic = True
   End With
End Sub
```

Las instrucciones que establecen valores propiedad son también instrucciones de asignación. El siguiente ejemplo asigna la propiedad **Bold** del objeto **Font** para la celda activa:
ActiveCell.Font.Bold = Trae

USO DE FUNCIONES PERSONALIZADAS - Function -

Un procedimiento **Function** es una serie de instrucciones de Visual Basic encerradas entre dos instrucciones **Function** y **End Function**. Un procedimiento **Function** es similar a un procedimiento **Sub**, aunque una función puede devolver además un valor. Un procedimiento **Function** acepta argumentos, como pueden ser constantes, variables o expresiones que le pasa el procedimiento que efectúa la llamada. Si un procedimiento **Function** no tiene argumentos, la instrucción **Function** debe incluir un par de paréntesis vacíos. Una función devuelve un valor asignándolo a su nombre en una o más instrucciones del procedimiento.

En el siguiente ejemplo, la función **Celsius** calcula grados centígrados a partir de grados Fahrenheit. Cuando se llama a la función desde el procedimiento **Principal**, se le pasa una variable que contiene el valor del argumento. El resultado de los cálculos se devuelve al procedimiento que efectúo la llamada y se presenta en un cuadro de mensaje.

```
Sub Principal()
   temp = Application.InputBox(Texto:= _
      "Por favor, introduzca la temperatura en grados F.", Tipo:=1)
   MsgBox "La temperatura es " & Celsius(temp) & " grados C."
End Sub

Function Celsius(GradosF)
   Celsius = (GradosF - 32) * 5 / 9
End Function
```

USO DE FUNCIONES DE HOJA DE CÁLCULO DE MICROSOFT EXCEL EN VISUAL BASIC

Puede usar la mayoría de las funciones de hoja de cálculo de Microsoft Excel en los enunciados de Visual Basic. Si desea ver una lista de las funciones de hoja de cálculo que puede usar, vea Lista de funciones para hojas de cálculo en Visual Basic.

Nota Algunas funciones de hoja de cálculo no tienen utilidad en Visual Basic. Por ejemplo, la función **Concatenar** no se necesita, ya que en Visual Basic puede usar el operador **&** para unir varios valores de texto.

Llamar a una función de hoja de cálculo desde Visual Basic
En Visual Basic, las funciones de hoja de calculo de Microsoft Excel pueden ejecutarse mediante el objeto **WorksheetFunction**.

El siguiente procedimiento **Sub** usa la función **Mín** para obtener el valor más pequeño de un rango de celdas. En primer lugar, se declara la variable miRango como un objeto **Range** y, a continuación, se establece como el rango A1:C10 de la Hoja1. Otra variable, respuesta, se asigna al resultado de aplicar la función **Mín** a miRango. Por último, el valor de respuesta se muestra en un cuadro de mensaje.

```
Sub UseFunction()
   Dim myRange As Range
   Set myRange = Worksheets("Sheet1").Range("A1:C10")
   answer = Application.WorksheetFunction.Min(myRange)
   MsgBox answer
End Sub
```

Si usa una función de hoja de cálculo que requiere como argumento una referencia de rango, deberá especificar un objeto **Range**. Por ejemplo, puede usar la función de hoja de cálculo **Coincidir** para efectuar una búsqueda en un rango de celdas. En una celda de hoja de cálculo, podría introducir una fórmula como =COINCIDIR(9;A1:A10;0). No obstante, en un procedimiento de Visual Basic, para obtener el mismo resultado debe especificar un objeto **Range**.

```
Sub FindFirst()
   myVar = Application.WorksheetFunction _
         .Match(9, Worksheets(1).Range("A1:A10"), 0)
   MsgBox myVar
End Sub
```

Nota Las funciones de Visual Basic no usan el calificador **WorksheetFunction**. Una función puede tener el mismo nombre que una función de Microsoft Excel y, sin embargo, dar otros resultados. Por ejemplo, Application.WorksheetFunction.Log y Log dan resultados diferentes.

Insertar una función de hoja de cálculo en una celda

Para insertar una función de hoja de cálculo en una celda, especifique la función como el valor de la propiedad **Formula** del objeto **Range** correspondiente. En el siguiente ejemplo, la función ALEATORIO (que genera un número aleatorio) se asigna a la propiedad **Formula** del rango A1:B3 de la Hoja1 del libro activo.

```
Sub InsertFormula()
      Worksheets("Sheet1").Range("A1:B3").Formula = "=RAND()"
End Sub
```