Excel con Visual Basic Para Aplicaciones (VBA)

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

Parte 1:

Características Generales

INTRODUCCIÓN

- Esencialmente la programación de Excel se reduce a la manipulación de objetos (mediante la escritura de instrucciones en un lenguaje que Excel puede entender), mediante **el lenguaje VBA**.
- VBA está pensado como un lenguaje de secuencia de comandos de aplicación de Microsoft común y ahora está incluido en las aplicaciones de Office 2000 (e incluso en aplicaciones de otros fabricantes). Por consiguiente, al dominar VBA usando Excel se podrá entrar directamente en la escritura de macros para otros productos de Microsoft. Mejor aún, se pueden generar soluciones completas que usan opciones a través de varias aplicaciones.

3

INTRODUCCIÓN

- El secreto de usar VBA con otras aplicaciones reside en entender el **modelos de objetos** para cada aplicación.
- VBA simplemente manipula objetos, y cada producto (Excel, Word, Access, PowerPoint y demás) posee un modelo de objeto único propio.
- Por ejemplo Excel expone varios objetos de análisis de datos muy potentes como las hojas de cálculo, gráficos, tablas dinámicas, escenarios y numerosas funciones matemáticas, financieras y temas generales. Con VBA se puede trabajar con estos objetos y diseñar procedimientos automatizados.

4

EL NÚCLEO DE VBA

- Las acciones de VBA se realizan mediante la ejecución del código VBA.El código VBA se escribe (o se graba) y se guarda en un módulo VBA.
- Los módulos se guardan en un libro de trabajo de Excel pero se editan o visualizan en el editor de Visual Basic.
- Un módulo VBA se compone de procedimientos. Un procedimiento es básicamente un código de ordenador que realiza alguna acción sobre los objetos o con ellos.

```
Sub Prueba()
Sum= 1+1
MSGBox “La respuesta es” & Sum
End Sub
```

5

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es
Excel con VBA

EL NÚCLEO DE VBA

- Un módulo VBA puede tener también procedimientos Function. Un procedimiento Function devuelve un solo valor. Se puede llamar una función desde otro procedimiento VBA o usar una fórmula de hoja de cálculo.

```
Function AñadirDos(arg1, arg2)
    AñadirDos= arg1 + arg2
End Function
```

- VBA manipula los objetos contenidos en su aplicación anfitriona (en este caso Excel).

6

EL NÚCLEO DE VBA

- Excel proporciona más de 100 clases de objetos para manipular. Ejemplos de objetos son un libro de trabajo, una hoja de cálculo, un rango de una hoja de cálculo, un gráfico y un rectángulo dibujado. Existen muchos más objetos a nuestra disposición y se pueden manipular mediante el uso de código VBA.
- Las clases de objetos están ordenados jerárquicamente. Los objetos pueden actuar como contenedores de otros objetos. Por ejemplo, Excel es un objeto llamado Applcation y contiene otros objetos como WorkBook y CommandBar. El objeto WoorkBook puede contener otros objetos como WorkSheet y Chart. Un objeto WoorkSheet puede contener objetos como Range, PivotTable y demás. Nos referimos al orden de estos objetos como **modelo de objeto de Excel.**

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es
Excel con VBA

EL NÚCLEO DE VBA

Objetos similares forman una colección. Por ejemplo, la colección WorkSheets está compuesta por todas las hojas de cálculo de un libro concreto. La colección CommandBars está compuesta por todos los objetos CommandBar. Las colecciones son objetos en sí mismas.

Cuando nos referimos a un objeto contenido o miembro, estamos especificando su posición en la jerarquía del objeto, usando un punto como seprador entre el contenedor y el miembro.

Application.WorkBooks(“Libro1.xls”)

Esto se refiere al libro de trabajo Libro1.xls en la colección WorkBooks.

EL NÚCLEO DE VBA

- Nos podemos referir a la Hoja1 del Libro1 como

  Application.WorkBooks("Libro1.xls").WorkSheets("Hoja1")

- Si se quiere omitir una referencia específica a un objeto, Excel usa los objetos activos.Si el libro1 es el libro de trabajo activo, la referencia anterior se puede simplificar a

  WorkSheets("Hoja1").Range("A1")

- Si se sabe que la Hoja1 es la hoja activa, se puede incluso simplificar más

  Range("A1")

9

EL NÚCLEO DE VBA

- Los objetos tienen propiedades. Se puede pensar en una propiedad como en una configuración para un objeto.
- Por ejemplo, un objeto Range tiene propiedades como Value y Name. Un objeto Chart tiene propiedades como HasTitle y Type. Se puede usar VBA para determinar las propiedades del objeto y también para cambiarlas.
- Nos podemos referir a propiedades para combinar el objeto con su propiedad, separados por punto.
- Por ejemplo nos podemos referir al valor de la celda A1 de la Hoja1 como

  WorkSheets("Hoja1").Range("A1").Value

10

EL NÚCLEO DE VBA

- Se pueden asignar valores a variables de VBA.Para asignar un valor en la celda A1 de la Hoja1 a una variables llamada Interés, use la siguiente instrucción VBA:

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

- Los objetos disponen de métodos.Un método es una acción que se realiza con el objeto. Por ejemplo, uno de los métodos para el objeto Range es ClearContents. Este método borra el contenido del método.
- Se pueden especificar métodos mediante la combinación del objeto con el método, separados por punto.

  Range (“A1”).ClearContents

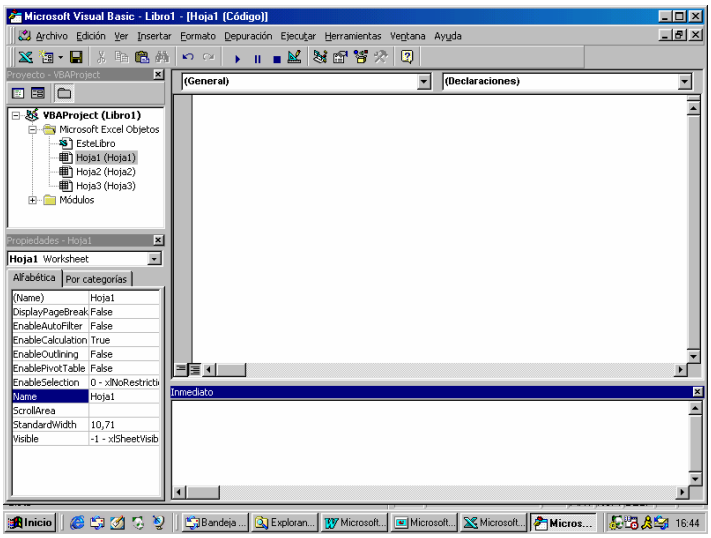
- VBA también incluye todas las estructuras de lenguajes de programación modernos, como matrices, bucles y de más.

EL EDITOR DE VISUAL BASIC

Para activar el editor de Visual Basic puede usar alguna delas siguientes técnicas:

- Presionar ALT+F11
- Seleccionar HERRAMIENTAS MACRO EDITOR DE VISUAL BASIC
- Hace clic sobre el botón Editor de Visual Basic, sitiado en la barra de herramientas de Visual Basic

VENTANAS DEL EDITOR DE VISUAL BASIC



VENTANAS DEL EDITOR DE VISUAL BASIC

El Editor de Visual Basic está compuesto de varias ventanas partes :

Barra de menús

Barras de herramientas

Ventana Explorador de proyectos. Presenta un diagrama de árbol que contiene cada libro de trabajo que está actualmente abierto en Excel. Cada libro de trabajo es un **proyecto**. Si esta ventana no esta abierta hay que presionar Control+R. Para esconder la ventana, hay que hacer clic en el botón Cerrar de su narra de título.

Ventana código. Una ventana de código, o ventana de módulo, contiene un código VBA. Cada elemento de un proyecto tiene asociada una ventana de código. Para visualizar una ventana de código para un objeto, hay que hacer doble clic sobre el objeto en la ventana del Explorador de proyectos.

Ventana inmediato. Esta ventana es útil para ejecutar instrucciones de VBA directamente, probar las instrucciones y limpiar el código. Para abrir esta ventana presione Control+G, para ocultarla basta hacer clic sobre el botón Cerrar de su barra de título.

AÑADIR UN NUEVO MÓDULO DE VBA

Para añadir un nuevo módulo de VBA a n proyecto, hay que seleccionar el nombre del proyecto en la ventana Explorador de proyectos y seleccionar INSERTAR MODULO. Cuando se graba una macro, Excel inserta automáticamente un módulo VBA para contener el código grabado.

QUITAR UN MÓDULO VBA

Hay que seleccionar el nombre del módulo en la ventana del Explorador de proyectos y elegir ARCHIVO QUITAR.

GUARDAR UN CÓDIGO VBA

En general una ventana de código puede soportar cuatro tipos de código:

Procedimientos Sub. Conjuntos de instrucciones que ejecutan alguna acción.

Procedimientos Function. Es un conjunto de instrucciones que devuelven un solo valor.

Procedimientos Property. Son procedimientos especiales que se usan en módulos de clase.

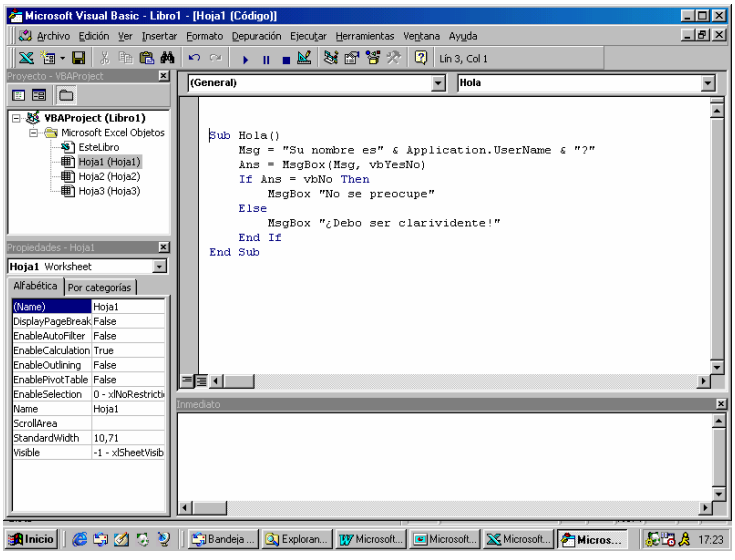
Declaraciones. Es información acerca de una variable que se le proporciona aVBA.

Un solo módulo de VBA puede guardar cualquier cantidad de procedimientos Sub, procedimientos Function y declaraciones.

INTRODUCIR UN CÓDIGO VBA

```
Sub Hola()
        Msg = "Su nombre es " &Application.UserName & "?"
        Ans = MsgBox(Msg, vbYesNo)
        If Ans = VbNo Then
                MsgBox "No se preocupe"
        Else
                MsgBox  "¿Debo ser clarividente!"
        End If
End Sub
```


INTRODUCIR UN CÓDIGO VBA



EJECUTAR UN CÓDIGO VBA

Para ejecutar el procedimiento Hola es necesario asegurarse que el cursor está situado en cualquier parte del texto introducido. Después se puede optar por uno de los siguientes métodos:

- Presionar F5
- Seleccionar Ejecutar, Ejecutar Sub/UserForm
- Hacer clic sobre el botón Ejecutar, Ejecutar Sub/UserForm

USAR EL GRABADOR DE MACROS

El siguiente ejemplo muestra la forma de grabar una macro que simplemente cambia la configuración de página de orientación vertical a horizontal.

1) Activar una hoja de cálculo del libro activo
2) Seleccionar el comando HERRAMIENTAS MACRO GRABAR NUEVA MACRO
3) Hacer cli en Aceptar para aceptar las opciones predeterminadas
4) Seleccionar el comando ARCHIVO CONFIGURAR PÁGINA
5) Seleccione la opción Horizontal y haga clic en Aceptar
6) Haga clic en el botón detener grabación

(Ejemplo parte 1)

```
Sub Macro1()
' Macro1 Macro
' Macro grabada el 15/11/00 por PRI00
'
    With ActiveSheet.PageSetup
        .PrintTitleRows = ""
        .PrintTitleColumns = ""
    End With
    ActiveSheet.PageSetup.PrintArea = ""
    With ActiveSheet.PageSetup
        .LeftHeader = ""
        .CenterHeader = ""
        .RightHeader = ""
        .LeftFooter = ""
        .CenterFooter = ""
        .RightFooter = ""
        .LeftMargin = Application.InchesToPoints(0)
         .RightMargin = Application.InchesToPoints(0)
         .TopMargin = Application.InchesToPoints(0)
```


(Ejemplo parte 2)

```
        .BottomMargin = Application.InchesToPoints(0)
        .HeaderMargin = Application.InchesToPoints(0)
        .FooterMargin = Application.InchesToPoints(0)
        .PrintHeadings = False
        .PrintGridlines = False
        .PrintComments = xlPrintNoComments
        .PrintQuality = -4
        .CenterHorizontally = False
        .CenterVertically = False
        .Orientation = xlLandscape
        .Draft = False
        .PaperSize = xlPaperA4
        .FirstPageNumber = xlAutomatic
        .Order = xlDownThenOver
        .BlackAndWhite = False
        .Zoom = 100
    End With
End Sub
```

A menudo el código producido cuando se graba una macro es excesivo.

La macro anterior se puede simplificar a :

```
Sub Macro1()
' Macro1 Macro
' Macro grabada el 15/11/00 por PRI00
   With ActiveSheet.PageSetup
        .Orientation = xlLandscape

   End With
End Sub
```

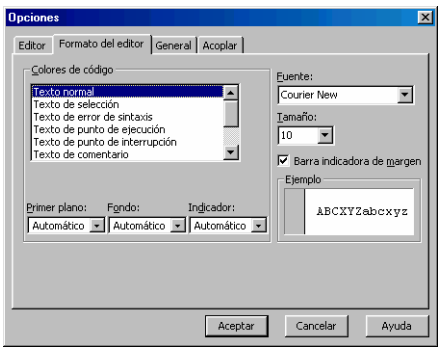

PERSONALIZAR EL ENTORNO DEL EDITOR DE VB

La ficha Editor

- Si está configurada la opción sugerencias de datos automáticas, el editor de VB presenta el valor de la variable sobre la que esta situada el cursor cuando se está depurando un código.
- Seleccionar la opción de sangría automática determina si el editor de VB realiza automáticamente una sangría en cada línea de código nueva igual que la de la línea anterior. También puede colocar sangrías con la tecla tab, o desde la barra de herramientas edición.
- Cuando se encuentra activa la opción modificar texto mediante “arrastrar y colocar” permite copiar y mover texto mediante dicha técnica.
- La opción vista completa predeterminada del módulo configura el estado predeterminado de los módulos nuevos (no afecta a los existentes). Si se selecciona, los procedimientos de la ventana código aparecerán como una sola ventana con barras de desplazamiento. Si se desactiva esta opción sólo se podrá ver un solo procedimiento a la vez.
- Cuando se selecciona la opción separador de procedimientos, presenta unas líneas separadoras al inicio y al final del procedimiento de una ventana de código.

PERSONALIZAR EL ENTORNO DEL EDITOR DE VB

La ficha Formato del editor



- La opción colores del código permite establecer el color del texto (de primer plano o de fondo), así como el color del indicador desplegado para varios elementos de código VBA.
- La opción fuente permite seleccionar la fuente que se va usar en los módulos de VBA

PERSONALIZAR EL ENTORNO DEL EDITOR DE VB

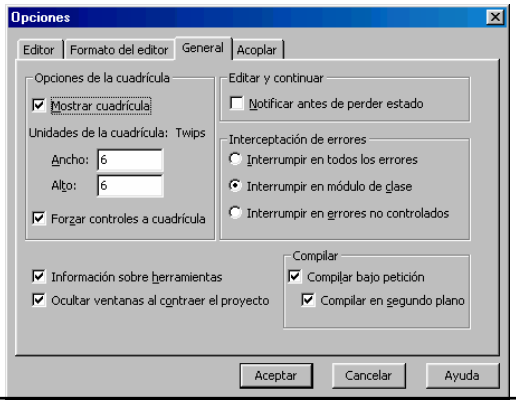
La ficha Formato del editor

La configuración del tamaño especifica el tamaño de fuente de los módulos de VBA. El tamaño predeterminado es 10.

La opción barra indicadora al margen controla el despliegue de la barra indicadora del margen vertical de los módulos. Es útil cuando se está depurando el código.

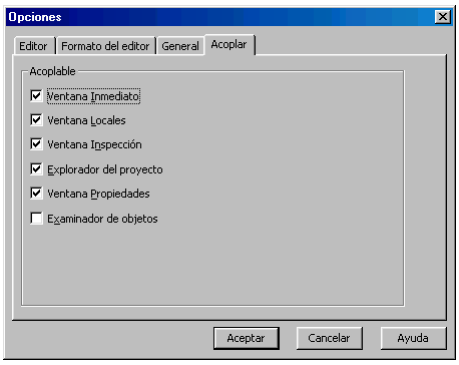
La ficha General

En general las opciones predeterminadas suelen ser muy útiles.



PERSONALIZAR EL ENTORNO DEL EDITOR DE VB

La ficha Acoplar



Las configuraciones de esta ficha determinan el comportamiento de las diversas ventanas en el Editor de VB. Cuando una ventana está acoplada, se fija en un lugar a lo largo de los bordes de la ventana del VBE. Esto facilita la identificación y localización de una ventana en particular. Generalmente las configuraciones predeterminadas funcionan bien.

27

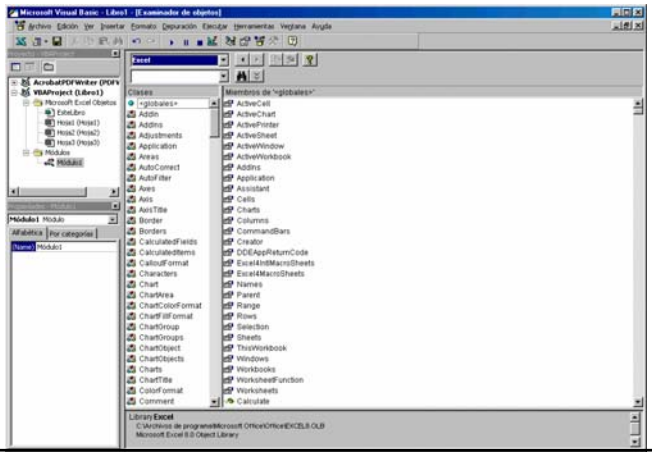
Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

PERSONALIZAR EL ENTORNO DEL EDITOR DE VB

El Examinador de Objetos

El Examinador de objetos es una herramienta muy práctica que presenta cada propiedad y método para todos los objetos disponibles. Cuando se activa el VBE, se puede llamar al Examinador de objetos presionando F2, o seleccionando la opción Ver Examinador de Objetos.



28

Parte 2:

Variables, Constantes, Expresiones y Tipos de Datos

VARIABLES Y TIPOS DE DATOS

- El propósito principal de VBA es manipular datos. Algunos residen en objetos tales como rangos de hojas de cálculo. Otros se guardan en las variables que se crean.
- Una variable es una localización de almacenamiento con nombre, dentro de la memoria del ordenador.
- VBA tiene algunas reglas relacionadas con los nombres de las variables:
  - Se pueden usar caracteres alfabéticos, números y algún carácter de puntuación, pero el primero de los caracteres debe ser alfabético
  - VBA no distingue entre mayúsculas y minúsculas
  - No se pueden usar espacios ni puntos
  - No se pueden incrustar en el nombre de una variable los siguientes símbolos: #, $, %,!
- Los nombres de las variables pueden tener hasta 254 caracteres

TIPOS DE DATOS INTEGRADOS EN VBA

Tipo de dato	Bytes usados	Rango de valores
Byte	1	0 a 255
Boolean	2	Verdadero o Falso
Integer	2	-32.768 hasta 32.767
Long	4	-2.147.483.648 hasta 2.147.483.647
Single	4	-3.402823E38 a -1.401298E-4545 (para valores negativos) y 1.4012298E-45 a 3.42823E38 (para valores positivos)
Double	8	
Currency	8	
Date	8	Del 1 Enero, 0100 hasta 31 de diciembre, 9999
Object	4	
String (de longitud variable)		
String (de longitud fija)		

ÁMBITO DE LAS VARIABLES

El ámbito de una variable determina el módulo y el procedimiento en el que se puede usar una variable.

Ámbito	Cómo se declara una variable en este ámbito
Un procedimiento	Incluye instrucciones Dim, Static o Private dentro del procedimiento.
Al nivel de módulo	Incluye una instrucción Dim antes del primer procedimiento de un módulo.
Todos los módulos	Incluye una instrucción Public antes del primer procedimiento de un módulo.

Variables locales

Una variable local es una variable declarada dentro de un procedimiento. Estas variables se pueden usar sólo en el procedimiento en que se declararon. Cuando el procedimiento finliza, la variable deja de existir y Excel libera su memoria.

Variables locales

La manera más común de declarar una variable local es situarla dentro de una instrucción DIM.

```
Sub MiSub()
        DIM x As Integer
        DIM First As Long
        DIM InerestRate As Single
        DIM TodaysDate As Date
        DIM UserName As String * 20
    ´Aquí se sitúa el código del procedimiento
    End Sub
```

Esto también es válido:

```
DIM x As Integer, y As Integer, z As Integer
```

Esto no es válido:

```
DIM x, y, z As Integer
```


Variables a nivel de módulo

Algunas veces se deseará que una variable esté disponible para todos los procedimientos de un módulo. Para ello, se declara la variable antes del primer procedimiento del módulo (fuera de cualquier procedimiento o función).

```
DIM fecha As Date
Sub Dias()
        ´Aquí va el código
End Sub
Sub Meses()
        ´Aquí va el código
End Sub
```

Variables Public

Para que una variable esté disponible para todos los procedimientos de un proyecto de Vba, se declara la variable a nivel de módulo con el uso de la palabra public.

```
        Public InterésAnual as Long
```

Variables Static

Las variables estáticas son un caso especial. Se declaran a nivel de procedimiento y retienen su valor después de que el procedimiento finaliza.

```
Sub Ejemplo()
        Static Counter As Integer
        ´Aquí va el código
End Sub
```


CONSTANTES

Declarar Constantes

Las constantes se declaran usando Const.

Const NumTrim As Integer = 4

Const Interés = 0.05, Periodo = 12

Const Nombre Mod as String = “Macros Presupuestos”

Public Const NombreApp As String = “Aplicación Presupuestos”

Al igual que las variables, las constantes también poseen un ámbito.

- Para que una constante esté disponible sólo dentro de in determinado procedimiento, hay que declararla después de Sub o Function para convertirla en una constante local.
- Para que una constante esté disponible para todos los procedimientos de un módulo, se tiene que declarar antes del primer procedimiento de un módulo.
- Para que una constante esté disponible para todos los módulos de un libro de trabajo, hay que usar la palabra Public, y declarar la constante antes del primer procedimiento de un módulo.

CONSTANTES

Declarar Constantes

Excel y VBA contienen muchas constantes predeterminadas, que se pueden usar sin necesidad de declararlas; de hecho no es necesario conocer el valor de estas constantes para usarlas.

```
Sub CalcManual()
  Application.Calculation = xlManual
End Sub
```

TRABAJAR CON CADENAS

Existen dos tipos de cadenas en VBA:

- Cadenas de longitud fija, que se declaran con un número específico de caracteres. La máxima longitud es de 65.536 caracteres.
- Cadenas de lingitud variable, que teóricamente pueden tener hasta 2.000 milones de caracteres.

```
Dim MiCadena As String * 50
Dim SuCadena As String
```


TRABAJAR CON FECHAS

```
Dim Hoy As Date
Dim HoraInicio As Date
Const PrimerDía As Date = #1/1/2001#
Const MedioDía As date = #12:00:00#
```

EXPRESIONES DE ASIGNACIÓN

Una expresión de asignación es una instrucción de VBA que realiza evaluaciones matemáticas y asigna el resultado a una variable o aun objeto.

VBA usa el signo igual "=" como operador de asignación.

```
x = 1
x = x + 1
x = (y * 2) / (z * 2)
FileOpen =  true
Range("Año"). Value = 1995
```

Se puede usar una secuencia de continuación (espacio seguido de un signo de subrayado) para facilitar la lectura de expresiones muy largas.

OPERADORES ARITMÉTICOS

+ Suma, - Resta, * Multiplicación, / División, \ División entera, Mod Resto, ^ exponencial, & Concatenación

OPERADORES COMPARATIVOS

= Igual, < Menor, > Mayor, <= Menor o igual, >= Mayor o igual, <> Distinto

OPERADORES LÓGICOS

Not (negación lógica, And (conjunción lógica), Or (disyunción lógica), XoR (exclusión lógica), Eqv (equivalencia en dos expresiones), Imp (implicación lógica)

MATRICES

- **Una matriz es un grupo de elementos del mismo tipo que tienen un nombre común; para referirse aun elemento específico de una matriz se usa el nombre de la matriz y un número de índice.**
- **Se puede declarar una matriz con DIM o Public como cualquier variable.**

MATRICES DE UNA DIMENSIÓN

```
DIM MiMatriz(1 To 100) As Integer
DIM MiMatriz (100)
MiMatriz(1) = 20
```

MATRICES MULTIDIMESIONALES

```
DIM MiMatriz(1 To 100, 1 to 10) As Integer
MiMatriz(1,2) = 20
```

VARIABLES DE OBJETO

Una variable de objeto es una variable que representa un objeto entero, como puede ser un rango o una hoja de cálculo.

Las variables de objeto son importante por dos razones:

•Pueden simplificar el código

•Pueden hacer que el código se ejecute más de prisa.

Se declaran con DIM o Public.

Por ejemplo, la expresión siguiente declara AreaEntradaDatos como un objeto Range.

Para ver como simplifican el código, veamos un ejemplo sin usar variables de objeto:

```
Sub VarSinObj();
        Worksheets("Hoja1").Range("A1").Value = 124
        Worksheets("Hoja1").Range("A1").Font.Bold = True
        Worksheets("Hoja1").Range("A1").Font. Italic = True
End Sub
```


Esta rutina introduce un valor en la celda A1 de la HOJA1, del libro de trabajo activo, y después aplica Negrita y cursiva a su contenido. Para reducir el código se puede condensar la rutina con una variable de objeto.

```
Sub VarObj();
        Dim MiCelda As Range
        Set Micelda = Worksheets("Hoja1").Range("A1")
        MiCelda.Value = 124
        MiCelda.Font.Bold = True
        MiCelda.Font. Italic = True
End Sub
```

TIPOS DE DATOS DEFINIDOS POR EL USUARIO

VBA permite crear tipos de datos personalizados definidos por el usuario (un concepto más parecido a los registros de Pascal o las estructuras de C).

```
Type InfoClientes
        Empresa As String * 25
        Ventas As Long
End Type
```

Los tipos de datos personalizados se definen fuera de los procedimientos, en la parte superior del módulo

TIPOS DE DATOS DEFINIDOS POR EL USUARIO

DIM Clientes(1 To 100) As InfoClientes

Se puede hacer referencia a una componente particular, de la siguiente manera:

Clientes(1).Empresa “Diseño Gráfico”

Clientes(1).Ventas= 187000

Para copiar la información de Clientes(1) en Clientes(2) se puede hacer lo siguiente:

Clientes(2) = Clientes(1)

FUNCIONES INTEGRADAS

Las funciones integradas de VBA no son las misma que las de Excel. La función UCASE de VBA, que convierte una cadena a mayúsculas es equivalente a la función MAYUSC de Excel.

```
Sub MostrarRaiz()
        MiValor = 25
        RaizCuadrada = Sqr(MiValor)
        MsgBox RaizCuadrada
End Sub
```

Para obtener la lista de funciones de VBA, se teclea en el código VBA seguido de un punto. El Editor de VB despliega una lista con todas las funciones.

FUNCIONES INTEGRADAS

El objeto WorsheetFunction, que está contenido en el objeto Application, contiene todas las funciones de hoja de cálculo que se pueden llamar desde los procedimientos VBA. Veamos un ejemplo para convertir un número decimal en número romano.

```
Sub MostrarRomano
        ValorDecimal = 1999
        ValorRomano = Application.WorsheetFunction.Romano(ValorDecimal)
        MsgBox ValorRomano
End Sub
```

Es importante saber que no se puede usar una función de hoja de cálculo que tenga una función de VBA equivalente.

Por ejemplo no se puede usar la función de hoja de cálculo RAIZ, porque VBA tiene la función SQR. La siguiente sentencia da error:

```
Application.WorksheetFunction.RAIZ(144)
```

Parte 3:

Manipulación objetos y colecciones

VBA ofrece dos importantes dos importantes estructuras que pueden simplificar el trabajo con objetos y colecciones:

–Estructuras With ...End With

–Estructuras For Each...next

ESTRUCTURAS WITH...END WITH

Permite realizar múltiples operaciones en un solo objeto.

```
Sub CambiarFuente()
    With Selection.Font
        .Name = “Times New Roman”
        .FontStyle = “Bold Italic”
        .Size = 12
        .Underline = xlSingle
        .ColorIndex = 5
    End With
End Sub
```

ESTRUCTURAS FOR EACH...NEXT

Recordemos que una colección es un grupo de objetos relacionado. Por ejemplo, la colección WorkBooks es una colección de todos los objetos WorkBook abiertos.

No es necesario saber la cantidad de elementos que existen en una colección para usar la estructura For Each...Next.

```
Sub ContarHojas()
        Dim Item As Worksheet
        For Each Item In ActiveWorkbook.Sheets
                MsgBox Item.Name
        Next Item
End Sub
Sub VentanasAbiertas()
        Suma = 0
        For Each Item In Windows
                Suma = Suma + 1
        Next Item
        MsgBox "Total de ventanas abiertas", & Suma
End Sub
```

Muestra el nombres de las hojas del libro de trabajo activo

Cuenta el número de ventanas abiertas

ESTRUCTURAS FOR EACH...NEXT

Cierra todos los libros de trabajo, excepto el activo

```
Sub CerrarActivo()
        For Each Book In Workbooks
                If Book.Name <> ActiveWorkbook.Name Then Book.Close
        Next Book
End Sub
```

Convierte a mayúsculas un rango previamente seleccionado

```
Sub ConvertirMayus()
        For Each Cell In Selection
                Cell.Value = UCASE(Cell.Value)
        Next Cell
End Sub
```

ESTRUCTURAS IF...THEN

Se usa para ejecutar una o más instrucciones de forma condicional. La sintaxis general es:

If *condición* **Then** *inst_verdaderas* **[Else** *inst_falsas***]**

```
Sub Positivos()
   a = InputBox("Ingrese un número")
   If a > 0 Then
      MsgBox "Número Positivo"
   End If
End Sub
```

Ingresa un número, y si es mayor que cero muestra el mensaje Número Positivo

```
Sub Positivos_Negativos()
   a = InputBox("Ingrese un número")
   If a > 0 Then MsgBox "Número Positivo" Else _
      MsgBox "Número negativo"
End Sub
```

Ingresa un número, y si es mayor que cero muestra el mensaje Número Positivo, y si no Número Negativo

ESTRUCTURAS IF...THEN

```
Sub Positivos_Negativos_Cero()
   a = InputBox("Ingrese un número")
   If a > 0 Then
      MsgBox "Número Positivo"
   ElseIf a < 0 Then
      MsgBox "Número negativo "
   ElseIf a = 0 Then
      MsgBox " Cero "
   End If
End Sub
```

Ingresa un número, y si es mayor que cero muestra el mensaje Número Positivo, si es menor que cero muestra el mensaje Número Negativo, y si es cero muestra el mensaje Cero

La función IIF de VBA es parecida a la función IF (SI) de Excel

IIF(*expresión, parte verdadera, parte falsa*)

```
Sub Descuento()
   MsgBox IIf(Range("A1") = 0, "Cero", "Distinto de Cero")
End Sub
```

ESTRUCTURAS SELECT CASE

La estructura Select Case es útil para elegir entre tres o más opciones

```
Sub Positivos_Negativos_Cero()
   a = InputBox("Ingrese un número")
   Select Case a
   Case Is > 0
      Msg = "Número Positivo"
   Case Is < 0
      Msg = "Número negativo"
   Case Else
      Msg = "Cero"
   End Select
   MsgBox Msg
End Sub
```


ESTRUCTURAS SELECT CASE

```
Sub Descuento1()
   Cantidad = InputBox("Introducir cantidad: ")
   Select Case Cantidad
   Case "": Exit Sub
    Case 0 To 24: Descuento = 0.1
    Case 25 To 49: Descuento = 0.15
    Case 50 To 74: Descuento = 0.2
    Case Is >= 75: Descuento = 0.25
End Select
   MsgBox "Descuento:  " & Descuento
End Sub
```

ESTRUCTURAS SELECT CASE

También se pueden anidar estructuras Select Case

El siguiente procedimiento, verifica el estado de la ventana de Excel (maximizada, minimizada o normal) y después presenta un mensaje describiendo dicho estado. Si el estado de la ventana de Excel es norma, el procedimiento verifica el estado de la ventana activa y después presenta otro mensaje.

```
Sub AppWindow()
   Select Case Application.WindowState
      Case xlMaximized: MsgBox "App Maximizada"
      Case xlMinimized: MsgBox "App Minimizada""
      Case xlNormal: MsgBox "App Normal"
                     Select Case ActiveWindow.WindowState
                              Case xlMaximized: MsgBox "Libro Maximizado"
                              Case xlMinimized: MsgBox "Libro Minimizado"
                              Case xlNormal: MsgBox "Libro Normal"
                     End Select
   End Select
End Sub
```


BUCLES FOR...NEXT

Esta sentencia de iteración se ejecuta un número determinado de veces

Su sintaxis es:

For contador = empezar **To** finalizar **[Step** valorincremento**]**
    [Instrucciones**]**
    [Exit For]
    [instrucciones**]**
Next [contador**]**

Suma los diez primeros números naturales

```
Sub SumaNúmeros
  Sum = 0
  For Count = 0 To 10
        Sum = Sum + Count
  Next Count
   MsgBox Sum
End Sub
```

BUCLES FOR...NEXT

```
Sub SumaNúmerosPares
  Sum = 0
  For Count = 0 To 10 Step 2
        Sum = Sum + Count
  Next Count
   MsgBox Sum
End Sub
```

Suma los números pares entre 0 y 10

```
Sub BuclesAnidados
  Dim MiMatriz(1 To 3, 1 To 3)
  For i = 1 To 3
        For j = 1 to 3
              MiMatriz(i,j) = 2
        Next j
  Next i
End Sub
```

Asigna el valor 2 a todos las casillas de una matriz de 3 x 3

BUCLES DO...WHILE

Do While se ejecuta **mientras** se verifica una condición especificada. Do While puede tener cualquiera de estas dos sintaxis.

```
Do [While condicion]
   [instrucciones]
   [Exit Do]
   [instrucciones]
Loop
```

```
Do
   [instrucciones]
   [Exit Do]
   [instrucciones]
Loop [While condicion]
```

```
Sub DoWhileDemo()
   Do While IsEmpty(ActiveCell)
      ActiveCell.Value = 0
      ActiveCell.Offset(1, 0).Select
   Loop
End Sub
```

Mientras la celda activa este vacía se desplaza hacia abajo asignándole a cada celda el valor 0

BUCLES DO...UNTIL

El bucle se ejecuta **hasta** que la condición llegue a ser verdadera. Do Until puede tener cualquiera de estas dos sintaxis.

Do Until condicion**]**
  [instrucciones**]**
  [Exit Do]
  [instrucciones**]**
Loop

Do
  [instrucciones**]**
  [Exit Do]
  [instrucciones**]**
Loop [Until condicion**]**

```
Sub DoUntilDemo()
   Do
      ActiveCell.Value = 0
      ActiveCell.Offset(1, 0).Select
   Loop Until Not IsEmpty(ActiveCell)
End Sub
```


Parte 4:

Procedimientos SUB con VBA

PROCEDIMIENTOS SUB DE VBA

EJEMPLO: Mover una hoja y grabarlo con el grabador de macros

```
Sub MoverHoja()
   Sheets("Hoja3").Select
   Sheets("Hoja3").Move Before:=Sheets(1)
End Sub
```

Move es un método que desplaza una hoja a otro lugar de un libro. Puede moverse hacia atrás (Before) o hacia delante (After).

¿Cómo contar las hojas?

```
Sub ContarHojas()
For Each hoja In Worksheets
  contar = contar + 1
Next
MsgBox "Cantidad de hojas " & contar
End Sub
```

También se puede probar en la ventana de Inmediato:

?ActiveWorkBook.Sheets.Count

PROCEDIMIENTOS SUB DE VBA

¿Cómo saber el nombre de una hoja?

Probar en la ventana de Inmediato: ?ActiveWorkBook.Sheets(1).Name, aparecerá el nombre de la primera hoja HOJA1.

Configuración general

1) Crear un libro de trabajo vacío con cinco hojas de cálculo, llamadas Hoja1 ... Hoja5.

2) mover las hojas al azar para que no tengan un orden particular.

3) Guardar el libro de trabajo como Test.xls.

4) Activar el editor de VB y seleccionar el proyecto Personal.xls del Explorador de proyectos. Si Personal no aparece, grabar una macro (cualquiera) y como destino para la macro seleccionar libro de macros personal.

5) Insertar un nuevo módulo de VBA (Insertar Módulo).

6) Crear un procedimiento vacío llamado SortSheets.

PROCEDIMIENTOS SUB DE VBA

7) Activar Excel. Usar el comando Herramientas, Macro, Macros (botón Opciones) para asignar una tecla de método abreviado a la macro. La combinación Control-Mayús-S puede ser una buena elección.

Escribir el código

```
Sub SortSheets()
   Dim SheetNames()
   SheetCount = ActiveWorkbook.Sheets.Count
   ReDim SheetNames(1 To SheetCount)
   For i = 1 To SheetCount
      SheetNames(i) = ActiveWorkbook.Sheets(i).Name
      MsgBox SheetNames(i)
   Next i
End Sub
```


PROCEDIMIENTOS SUB DE VBA

Para probar el código se activa el libro Test.xls, y se prueba la combinación Contriol-Mayús-S. Luego debería borrase la instrucción MsgBox.

Escribir el procedimiento de clasificación

```
Sub BubbleSort(List())
   Dim First As Integer, Last As Integer
   Dim i As Integer, j As Integer, Temp As String
   First = LBound(List)
   Last = UBound(List)
   For i = First To Last - 1
    For j = i + 1 To Last
      If List(i) > List(j) Then
         Temp = List(j)
         List(j) = List(i)
         List(i) = Temp
      End If
    Next j
   Next i
End Sub
```

PROCEDIMIENTOS SUB DE VBA

```
Sub SortSheets()
   Dim SheetNames() As String
   Dim i As Integer
   Dim SheetCount As Integer
   SheetCount = ActiveWorkbook.Sheets.Count
   ReDim SheetNames(1 To SheetCount)
   For i = 1 To SheetCount
      SheetNames(i) = ActiveWorkbook.Sheets(i).Name
   Next i
   Call BubbleSort(SheetNames)
   For i = 1 To SheetCount
      ActiveWorkbook.Sheets(SheetNames(i)).Move ActiveWorkbook.Sheets(i)
   Next i
End Sub
```


Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es
Excel con VBA

Parte 5:

Procedimientos FUNCTION con VBA

PROCEDIMIENTOS FUNCTION DE VBA

Los procedimientos **Function** devuelven un solo valor (al igual que las funciones de hoja de cálculo de Excel y las funciones incorporadas en VBA).

Los procedimientos Function se pueden usar en dos situaciones:

- Como parte de una expresión en un procedimiento VBA
- En fórmulas que se crean en una hoja de cálculo

Lo que no pueden realizar las funciones de hoja de cálculo personalizadas

- Al diseñar funciones personalizadas, es importante entender una distinción clave entre funciones a las que se puede llamar desde un procedimiento VBA y funciones que se usan en fórmulas de hoja de cálculo.
- Los procedimientos de función usados en fórmulas de hoja de cálculo deben ser "pasivos". Por ejemplo, un código dentro de un procedimiento de función no puede manipular un rango. Es necesario recordar que una función devuelve un valor, no ejecuta acciones con objetos.

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

PROCEDIMIENTOS FUNCTION DE VBA

DECLARAR UNA FUNCIÓN

[Private | **Public] [Static] Function** nombre **[**(lista_argumentos)**] [As** tipo**]**

[instrucciones**]**

[Exit Sub]

[instrucciones**]**

[nombre = expresión**]**

End Function

Private. Opcional. Indica que el procedimiento Functiones accesible sólo para otros procedimientos del mismo módulo.

Public. Opcional. Indica que el procedimiento es accesible para todos los procedimientos, de todos los proyectos activos de VBA.

Static. Opcional. Indica que las variables del procedimiento Function se conservan entre llamadas.

Sub. Requerido. Palabra clave que indica el principio de un procedimiento que devuelve un valor u otro dato.

PROCEDIMIENTOS FUNCTION DE VBA

DECLARAR UNA FUNCIÓN

nombre. Requerido. Cualquier nombre de procedimiento Function válido (Igual que las variables, pero no se pueden poner nombres de celdas, i.e., J34). Cuando finaliza la función, el resultado de un solo valor se asigna a su propio nombre.

lista_argumentos. Opcional. Representa una lista de variables, encerradas entre paréntesis, que reciben argumentos pasados al procedimiento. Para separar los argumentos se usa una coma.

tipo. Opcional. Es el tipo de dato devuelto por la función.

instrucciones. Opcional. Cualquier número de instrucciones de VBA válidas.

Exit Function. Opcional. Una instrucción que fuerza una salida inmediata del procedimiento Function antes de su conclusión formal.

End Function. Requerido. Indica el final del procedimiento function.

EJECUTAR PROCEDIMIENTOS FUNCTION

Desde un procedimiento

```
Function SumaMatriz(matriz() As Integer)
Dim i As Integer, suma As Integer
suma = 0
For i = LBound(matriz) To UBound(matriz)
    suma = suma + matriz(i)
Next
SumaMatriz = suma
End Function
Sub LlamarSumaMatriz()
  Dim mat(5) As Integer
  Dim SumaTotal As Integer
  mat(1) = 4: mat(2) = 5: mat(3) = 8: mat(4) = 1: mat(5) = 2
  SumaTotal = SumaMatriz(mat)
MsgBox "La suma de la matriz es: " & SumaTotal
End Sub
```

EJECUTAR PROCEDIMIENTOS FUNCTION

En una fórmula de hoja de cálculo

usar funciones personalizadas en fórmulas de hoja de cálculo es como usar funciones integradas, excepto porque es necesario asegurarse de que Excel puede localizar el procedimiento Function. Si dicho procedimiento está localizado en el mismo lobro de trabajo, no es necesario hacer nada especial. Si está localizado en un libro de trabajo diferente, se puede decir a Excel que lo encuentre. Para ello existen dos maneras:

Preceder el nobre de la función con una referencia al archivo

= MisFunciones.xls!ContarNombres(A1:A1000)

Establecer una referencia al libro de trabajo. Para ello se emplea el comando del Editor de VBA, Herramientas, Referencias. Si la función está definida en un libro de trabajo referenciado, no será necesario usar el nombre de la hoja de cálculo.

EJECUTAR PROCEDIMIENTOS FUNCTION

Los procedimientos Function no aparecen en el cuadro de diálogo de las macros. Además no se puede ejecutar directamente desde el Editor de VBA.

Para ejecutar una función es necesario definir un procedimiento Sub que la llame. Si la función esta diseñada para ser usada en fórmulas de hojas de cálculo, se puede introducir una simple fórmula para probarla.

ARGUMENTOS DE FUNCIÓN

- Los argumentos pueden ser variable (incluyendo matrices), constantes, literales o expresiones
- Algunas funciones no tienen argumentos
- Algunas funciones tienen un número fijo de argumentos (desde 1 a 60)
- Algunas funciones tienen una combinación de argumentos opcionales y requeridos

EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función sin argumentos

La siguiente función devuelve la propiedad UserName del objeto Application.

```
Function Usuario()
  Usuario = Application.UserName
End Function
```

Cuando se introduce la siguiente fórmula, la celda devuelve el nombre del usuario actual:

```
=Usuario()
```

Para usar esta función en otro procedimiento, se debe asignar a una variable, usarla en una expresión o emplearla como argumento para otra función.

```
Sub MostrarUsuario()
  MsgBox "El usuario es: " & Usuario()
End Sub
```


EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función con un argumento

El siguiente ejemplo permite calcular las comisiones que debe cobrar cada vendedor de acuerdo a sus ventas mensuales.

VENTAS MENSUALES	% DE COMISIÓN
0-9.999	8,00%
10.000-19999	10,50%
20.000-39.999	12,00%
40.000+	14,00%

Una forma de calcular las comisiones es mediante el uso e la fucnión SI:

```
=si(Y(A1>00,A1<=9999,99),A1*0,08, si(Y(A1>=10000,A1>=19999,99),A1*0,105,
si(Y(A1>=20000,A1<=39999,99),A1*0,12, si(A1>=40000,A1*0,14,0))))
```

Este es un mal planteamiento por varias razones. Primera, la fórmula es demasiado compleja y muy difícil de entender. Segunda, los valores son códigos cerrados dentro de la fórmula haciendo la modificación de la fórmula muy difícil.

EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función con un argumento

Un planteamiento mejor es crear una función personalizada como la siguiente:

```
Function Comision(Ventas)
  Const  porcentaje1 = 0.08
  Const porcentaje2 = 0.105
  Const porcentaje3 = 0.12
  Const porcentaje4 = 0.14
  Select Case Ventas
        Case 0 To 9999.9: Comision = Ventas * porcentaje1
        Case 1000 To 19999.9: Comision = Ventas * porcentaje2
        Case 20000 To 39999.9: Comision = Ventas * porcentaje3
        Case Is >= 40000: Comision = Ventas * porcentaje4
 End Select
End Function
```


EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función con un argumento

```
Sub CalcComm()
   sales = InputBox("Introducir Ventas: ")
   MsgBox "La comisión es: " & Comision(sales)
End Sub
```

EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función con dos argumentos

La comisión total pagada se incrementa en un 1 por ciento por cada año que el vendedor ha estado en la compañía.

```
Function Comision2(Ventas, Años)
   Const porcentaje1 = 0.08
   Const porcentaje2 = 0.105
   Const porcentaje3 = 0.12
   Const porcentaje4 = 0.14
   Select Case Ventas
      Case 0 To 9999.9: Comision2 = Ventas * porcentaje1
      Case 1000 To 19999.9: Comision2 = Ventas * porcentaje2
      Case 20000 To 39999.9: Comision2 = Ventas * porcentaje3
      Case Is >= 40000: Comision2 = Ventas * porcentaje4
    End Select
    Comision2 = Comision2 + (Comision2 * Años / 100)
End Function
```


EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función con un argumento de matriz

La siguiente función acepta una matriz como argumento y devuelve la suma de sus elementos

```
Function SumaMatriz(List)
   SumaMatriz = 0
   For i = LBound(List) To UBound(List)
         SumaMatriz = SumaMatriz + List(i)
   Next i
End Function

Sub HacerList()
  Dim Num(1 To 100) As Integer
  For i = 1 To 100
      Num(i) = i
  Next i
  MsgBox SumaMatriz(Num)
End Sub
```

EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Una función con argumentos opcionales

```
Function Calculo(A As Integer, B As Integer, Optional operación As String)

    If IsMissing(operación) Then operación = "suma"
        Select Case operación
            Case Is = "suma": Calculo = A + B
            Case Is = "resta": Calculo = A - B
            Case Is = "multiplicacion": Calculo = A * B
            Case Is = "division": Calculo = A / B
        End Select
End Function
```

```
Sub probar()
  valor = Calculo(2, 4, "resta")
  MsgBox valor
End Sub
```

```
Sub probar()
  valor = Calculo(2, 4)
  MsgBox valor
End Sub
```


EJEMPLOS DE PROCEDIMIENTOS FUNCTION

Especificar la categoría de Función

Es siguiente código se ejecuta siempre que el libro de trabajo está abierto. Este procedimiento asigna la función Comision a la categoría financieras.

```
Private Sub Workbook_Open()
  Application.MacroOptions Macro:="Comision", Category:=3
End Sub
```

NÚMERO DE CATEGORÍA	NOMBRE DE CATEGORIA
0	Todas
1	Financieras
2	Fecha y hora
3	Matemáticas y trigonométricas
4	Estadísticas
5	Búsqueda y referencia
6	Base de datos
7	Texto
8	Lógicas
9	Información
10	Comandos
11	Personalizado
12	Control de macros
13	DDE/Externas
14	Definidas por el usuario
15	Ingeniería

Parte 6:

Formularios

UserForms

Un cuadro de diálogo personalizado se genera en un UserForm y se puede acceder a él con ayuda del Editor de VB.

A continuación se expone la secuencia típica de fases a realizar para la creación de un UserForm:

1) Insertar un UserForm en el libro de trabajo

2) Escribir un procedimiento que despliegue el UserForm Este procedimiento está localizado en un módulo de VBA (no en el módulo de código para el UserForm)

3) Añadir controles al UserForm

4) Ajustar algunas propiedades a los controles añadidos

5) Escribir procedimientos de controlador de evento para los controles. Estos procedimientos que están situados en la ventana de código para el UserForm, se ejecutan cuando ocurren varios eventos (como hacer clic con el ratón).

UserForms

INSERTAR UN NUEVO UserForm

Activar el Editor de VB y seleccionar el Libro de trabajo correspondiente y elegir el comando Insertar, UserForm.

Un libro de trabajo puede tener cualquier número de UserForms, y cada uno de ellos contiene un solo UserForms. Los UserForms tienen nombres como **UserForm1**, **UserForm2**, y así sucesivamente.

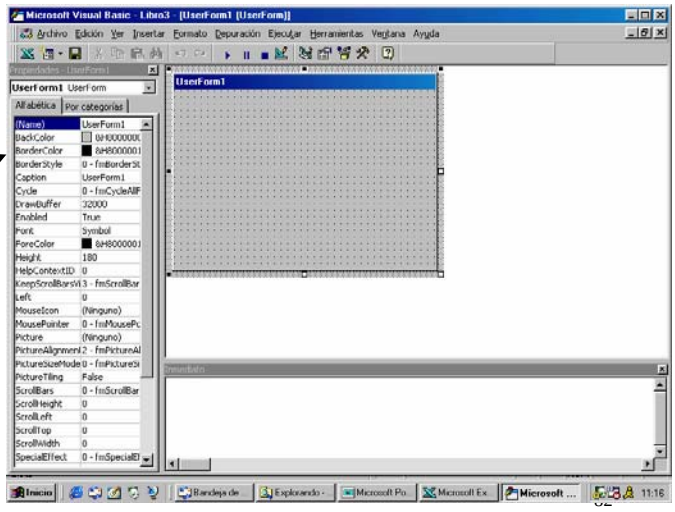
Se puede cambiar el nombre del UserForm para que su identificación sea más fácil. Para ello se selecciona el UserForm y se usa la ventana propiedades, desde donde se puede cambiar la propiedad Name (presionar F4 si la ventana de propiedades no está desplegada).

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

UserForms

Ventana Propiedades



UserForms

DESPLEGAR UN UserForm

Se usa el método **Show** del objeto UserForm. El siguiente procedimiento, que se encuentra dentro de un módulo de VBA normal, despliega UserForm1:

```
Private Sub UserForm1.Clik()
End Sub
```

Cuando se despliega el UserForm, permanece visible en la pantalla hasta que se oculta. El procedimiento puede tanto cargar el UserForm (con una instrucción Unload) como ocultarlo (con el método Hide del objeto UserForm)

AÑADIR CONTROLES A UN UserForm

Se usa el Cuadro de herramientas (el Editor de VB no tiene comandos de menú para añadir controles). Este cuadro se puede desplegar con Ver, Cuadro de Herramientas.

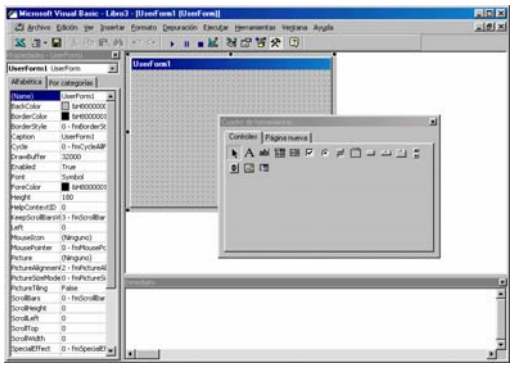
Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

UserForms

AÑADIR CONTROLES A UN UserForm

Se usa el Cuadro de herramientas (el Editor de VB no tiene comandos de menú para añadir controles). Este cuadro se puede desplegar con Ver, Cuadro de Herramientas.



UserForms

AÑADIR CONTROLES A UN UserForm

Se hace clic sobre el botón del Cuadro de Herramientas que corresponde al control que se quiere añadir, y después se hace clic dentro del cuadro de diálogo.

Cuando se añade un control nuevo, se le asigna un nombre que combina el tipo de control con la secuencia numérica para ese tipo de control. Por ejemplo si se añade el primer botón de comando se le asignará el nombre ComandButton1, y al segundo que se añada CommandButton2. Siempre conviene cambiarle el nombre a los controles para que sean más representativos. Los nombres de lo controles se cambian desde la ventana de Propiedades.

CONTROLES DISPONIBLES PARA EL USUARIO

Casilla de verificación (CheckBox). Es útil para ofrecer al usuario una opción binaria: sí o no, verdadero o falso, activar o desactivar, y demás. Cuando se selecciona una Casilla de verificación posee un valor Verdadero; en caso contrario es Falso.

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

UserForms

CONTROLES DISPONIBLES PARA EL USUARIO

Cuadro combinado (ComboBox). Es similar al cuadro de lista. Sin embargo, un Cuadro Combinado es un cuadro de lista desplegable que presenta un solo elemento por vez. Otra diferencia con respecto al cuadro de lista es que el usuario puede introducir un valor que no aparece en la lista dada de elementos.

Botón de comando (CommandButton). Todo cuadro de diálogo que se genere probablemente tenga, al menos, un Botón de comando. Normalmente se tendrá un Botón de comando etiquetado como Aceptar y otro etiquetado como Cancelar.

Botón Macro (Frame). Se usa para agrupar otros controles. Se puede hacer bien por motivos estéticos o por agrupar lógicamente un conjunto de controles. Un Marco es particularmente útil cuando el cuadro de diálogo contiene más de un grupo de controles de Botón de opción.

UserForms

CONTROLES DISPONIBLES PARA EL USUARIO

Imagen (Image). Se usa para desplegar una imagen gráfica, que pude provenir de un archivo o se pude pegar desde el Portapapeles. La imagen gráfica se guarda en el libro de trabajo. De esta forma, se puede distribuir el libro a cualquier persona y no es necesario incluir una copia del archivo gráfico.

Etiqueta (Label). Simplemente presenta texto en el cuadro de diálogo.

Cuadro de lista (Listbox). Presenta una lista de elementos donde el usuario puede seleccionar uno (o múltiples elementos). Estos controles son muy flexibles. Por ejemplo, se puede especificar un rango de hoja de cálculo que contenga elementos de un Cuadro de lista, y este rango puede constar de múltiples columnas

Página múltiple (Multipage). Permite crear cuadros de diálogo con fichas, como el que aparece cuando se selecciona el comando herramientas, opciones. De forma predeterminada una página múltiple consta de dos páginas. Para añadir páginas, se hace clic con el botón derecho del ratón sobre una ficha y se selecciona Nueva página desde el menú contextual.

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

UserForms

CONTROLES DISPONIBLES PARA EL USUARIO

Botón de opción (OptionButtons). Son muy útiles cuando el usuario necesita seleccionar entre un pequeño número de elementos. Estos botones se usan siempre en grupos de al menos dos elementos. Cuando se selecciona uno de los botones, los otros botones del grupo no están seleccionados. Si el cuadro de diálogo contiene más de un grupo de Botones de opción, cada grupo de éstos debe tener el mismo valor de la propiedad group name. De lo contrario, todos los Botones de opción formarán parte del mismo grupo. De forma alternativa, se pueden agrupar los Botones de opción en un control Marco, que agrupa automáticamente los Botones de opción contenidos dentro del marco.

RefEdit. Se usa cuando es necesario permitir que el usuario seleccione un rango de una hoja de cálculo.

Barra de desplazamiento (ScrollBar). Es similar a un control Botón de número. La diferencia estriba en que el usuario puede desplazarse con el botón Barra de desplazamiento para cambiar el valor del contrl en incrementos más amplios. Dicho control es más útil para seleccionar un valor que se extiende a través de un rango muy amplio de posibles valores.

ISUAL BASIC PARA APLICACIONES

UserForms

CONTROLES DISPONIBLES PARA EL USUARIO

Botón de número (SpinButton). Permite al usuario seleccionar un valor haciendo clic sobre una de las dos flechas que contiene. Este control se usa a menudo en conjunción con el control Cuadro de texto o el control Etiqueta, que presentan el valor actual de un Control de número.

Barra de tabulaciones (TabStrip). Es similar a un control Página múltiple, pero no es tan fácil de usar.

Botón de alternar (ToggleButton). Posee dos estados: activado y desactivado. Al hacer clic sobre el mencionado botón, se alternan estos dos estados y el botón cambia de apariencia. Su valor puede ser o bien Verdadero (presionado) p bien Falso (no presionado).

UserForms

AJUSTAR LOS CONTROLES DEL CUADRO DE DIÁLOGO

Después de situar un control en un cuadro de diálogo, se puede mover y modificar su tamaño usando las técnicas del ratón estándar.

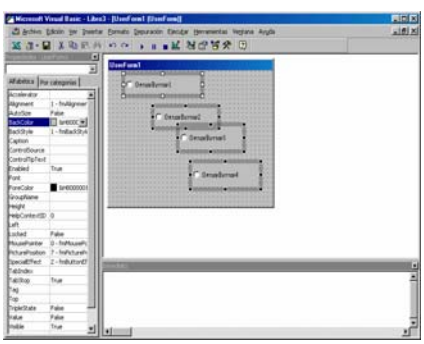
Un UserForm puede contener líneas de división horizontales y verticales que ayudan a alinear los controles que se añaden. Cuando se añade o se mueve un control, se ajusta a la cuadrícula. Si no se quieren ver estas líneas se pueden desactivar seleccionando Herramientas Opciones en el Editor de VB.

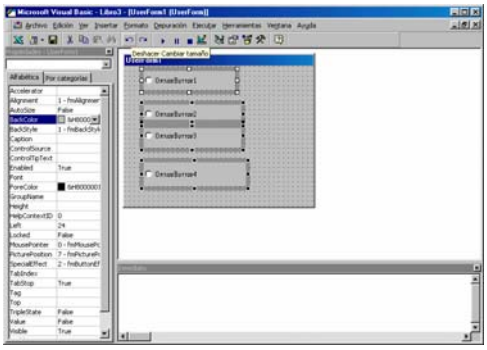
En el cuadro de diálogo Opciones se selecciona la ficha general y se establecen las opciones deseadas en la sección Opciones de la cuadricula.

El menú Formato de la ventana del Editor de VB proporciona varios comandos para ayudar a precisar la alineación y el espacio de los controles en un cuadro de diálogo. Antes de usar estos comandos hay que seleccionar los controles con los que se quiere trabajar.

UserForms

AJUSTAR LOS CONTROLES DEL CUADRO DE DIÁLOGO





UserForms

AJUSTAR LAS PROPIEDADES DEL CONTROL

Se pueden cambiar las propiedades del control en el tiempo de diseño con la ventana de Propiedades, mientras se está configurando el cuadro de diálogo, o durante el tiempo de ejecución, cuando el cuadro de diálogo se presenta al usuario. Se pueden usar instrucciones VBA para cambiar las propiedades del control en el tiempo de ejecución.

USAR LA VENTANA DE PROPIEDADES

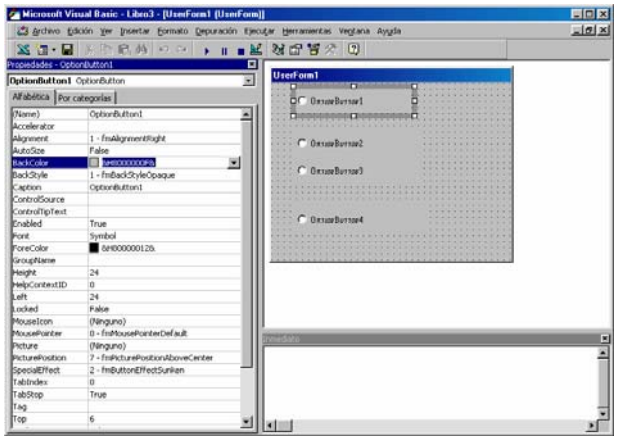
La ventana propiedades tiene dos fichas:

- La ficha alfabética presenta las propiedades del objeto seleccionado en orden alfabético
- La ficha Por categorías las presenta agrupadas en categorías lógicas

Si se seleccionan dos o más controles a la vez, la ventana Propiedades despliega sólo las propiedades comunes a los controles seleccionados.

UserForms

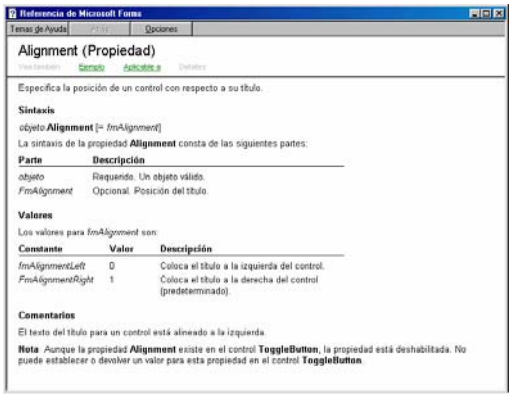
USAR LA VENTANA DE PROPIEDADES



UserForms

USAR LA VENTANA DE PROPIEDADES

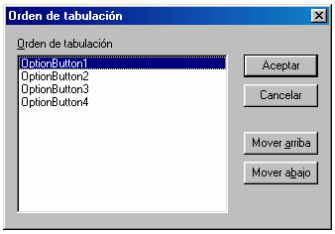
La mejor manera de aprender sobre diversas propiedades de un control es usar la Ayuda en línea. Simplemente se hace clic sobre una propiedad de la ventana de Propiedades y se presiona F1. Por ejemplo, la propiedad Alignment de un Botón de Opción.



UserForms

CAMBIAR EL ORDEN DE TABULACION

El orden de tabulación determina la secuencia en la que los controles se activan cuando el usuario presiona Tab o Mayús-tab.Para establecer el orden de tabulación de los controles se selecciona Ver Orden de tabulación en el Editor de VB



De forma alternativa, se puede establecer una posición de control individual en el orden de tabulación, usando la ventana Propiedades. El primer control en el orden de tabulación tiene una Propiedad **TabIndex** de 0. Cambiar esta propiedad puede afectar a otros controles. Si se quiere eliminar un control del orden de tabulación, se establece su propiedad TabStop como False.

95

UserForms

ESTABLECER TECLAS DE ACCESO DIRECTO

Se puede asignar una tecla de aceleración o tecla de acceso directo a la mayoría de los controles de un cuadro de diálogo. Esto permite al usuario acceder al control presionando Alt-tecla de acceso directo. Para ello se usa la propiedad **Accelerator** de la ventana propiedades.

PROBAR UN UserForm

Existen tres maneras de probar un UserForm sin tener que llamarlo desde un procedimiento de VBA.

- Elegir el comando Ejecutar, Ejecutar Sub/UserForm
- Presionar F5
- Hacer clic sobre el botón Ejecutar Sub/UserForm en la barra de herramientas Estándar

96

UserForms

DESPLEGAR UN UserForm

```
Sub MostrarDiálogo()
        UserForm1().show
End Sub
```

Este procedimiento debe estar en un módulo de VBA, no en el módulo del código del UserForm.

CERRAR UN UserForm

```
Unload UserForm1
```

PROCEDIMINETOS DE CONTROLADOR DE EVENTO

Cuando el usuario interactúa con el cuadro de diálogo, mediante la selección de un elemento de un cuadro de lista, haciendo clic sobre un botón de comando y demás, se produce un **evento** a ocurrir. Por ejemplo, hacer clic sobre el Botón de comando promueve el evento **Click** para dicho botón. La aplicación necesita procedimientos que se ejecuten cuando estos eventos ocurran. Estos procedimiento se llaman **controlador de evento**.

UserForms

PROCEDIMINETOS DE CONTROLADOR DE EVENTO

Los procedimientos de controlador de evento deben estar localizados en la ventana de código del UserForm. Sin embargo, el procedimiento de controlador de evento puede llamar a cualquier procedimiento que esté localizado en un módulo VBA estándar.

CREAR UN UserForm: Un ejemplo

- El ejemplo usa un UserForm para obtener dos tipo de información: el nombre y el sexo de una persona.
- Usa el control Cuadro de texto (TextBox) para obtener el nombre.
- Usa tres botones de opción (OptionsButtons) para obtener el sexo (masculino, femenino o desconocido).
- La información se recoge en el cuadro de diálogo y luego se envía a la siguiente fila en blanco de la hoja de cálculo.

UserForms

CREACIÓN DEL CUADRO DE DIÁLOGO

1) Abra un libro de trabajo nuevo

2) Presionar Alt-F11 para activar el Editor de VB

3) En la ventana Proyecto, seleccionar el proyecto del libro de trabajo y elegir insertar, UserForm para añadir un formulario vacío.

4) Si la ventana propiedades no está visible, presionar F4.

5) cambiar la propiedad Caption del UserForm (usando la ventana propiedades) a Obtener nombre y sexo.

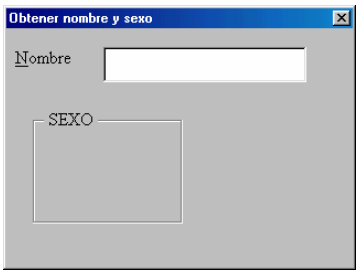
6) Añadir un control Etiqueta (Label) y ajustar sus propiedades: Accelerator N, caption Nombre, TabIndex 0

7) Añadir un control Cuadro de texto (TextBox) y ajustar sus propiedades: Name NombreTexto, TabIndex 1

8) Añadir un control Marco (frame) y ajustar sus propiedades: Caption Sexo, TabIndex 2

UserForms

CREACIÓN DEL CUADRO DE DIÁLOGO



9) Añadir un control Botón de opción (OptionButtons) dentro del Marco y ajustar sus propiedades: Accelerator M, Caption Masculino, Name OpciónMasculino, TabIndex 0

10) Añadir otro control Botón de opción (OptionButtons) dentro del Marco y ajustar sus propiedades: Accelerator F, Caption Femenino, Name OpciónFemenino, TabIndex 1

UserForms

CREACIÓN DEL CUADRO DE DIÁLOGO

11) Añadir otro control Botón de opción (OptionButtons) dentro del Marco y ajustar sus propiedades: Accelerator D, Caption Desconocido, Name Opcióndesconocido, TabIndex 2, Value True

12) Añadir un control Botón de Comando (CommandButton) dentro del Marco y ajustar sus propiedades como sigue: Caption Aceptar, Default True, Name BotónAceptar, TabIndex 3

13) Añadir otro control Botón de Comando (CommandButton) dentro del Marco y ajustar sus propiedades como sigue: Caption Cancelar, Cancel True, Name BotónCancelar, TabIndex 4

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

UserForms

CREACIÓN DEL CUADRO DE DIÁLOGO

ESCRIBIR UN CÓDIGO PARA DESPLEGAR EL CUADRO DE DIÁLOGO

Ahora se debe añadir un Botón de Comando a la hoja de cálculo (Cuadro de controles). Este botón ejecuta un procedimiento que despliega el UserForm.

1) Activar Excel

2) activar la barra Cuadro de Controles

3) Añadir un Botón de comando

UserForms

ESCRIBIR UN CÓDIGO PARA DESPLEGAR EL CUADRO DE DIÁLOGO

4) Hacer doble clic sobre el botón, esto activa el Editor de VB (específicamente, el módulo de código para la hoja de calculo se despliega, con un procedimiento controlador de evento vacío para el Botón de Comando (CommandButton)de la hoja de Cálculo)

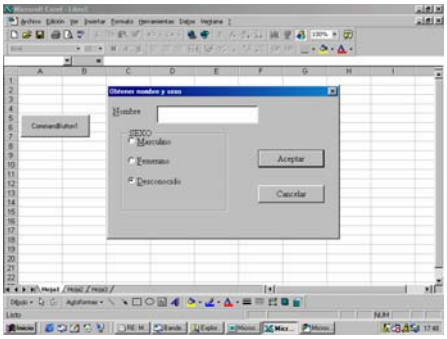
5) Añadir la instrucción UserForm1.Show al procedimiento

```
Private Sub CommandButton1-Click()
   UserForm1.Show
End Sub
```


UserForms

PROBAR EL UserForm

Al salir del modo de diseño (desactivarlo de la barra Cuadro de Controles) y hacer clic sobre el botón, se despliega el cuadro de diálogo:



Como aún no hemos creado ningún controlador de evento, sólo podemos cerrar el cuadro de diálogo.

UserForms

AÑADIR PROCEDIMIENTOS DE CONTROLADOR DE EVENTO

En esta sección se explica cómo escribir procedimientos que controlan los eventos que ocurren cuando el cuadro de diálogo se ha desplegado.

1) Activar el Editor de VB

2) Hacer doble clic sobre el botón Cancelar. El Editor de VB activa la ventana de Código del UserForm y proporciona un procedimiento vacío llamado BotónCancelar_Click

3) Modificar el procedimiento como sigue:

```
Private Sub Botóncancelar_Click()
  Unload UserForm
End Sub
```

4) Presionar Mayús-F7 para volver a desplegar el UserForm1

5) Hacer doble click sobre el botón Aceptar e introducir el siguiente procedimiento (éste es el controlador de evento para el evento Click del Botón Aceptar)

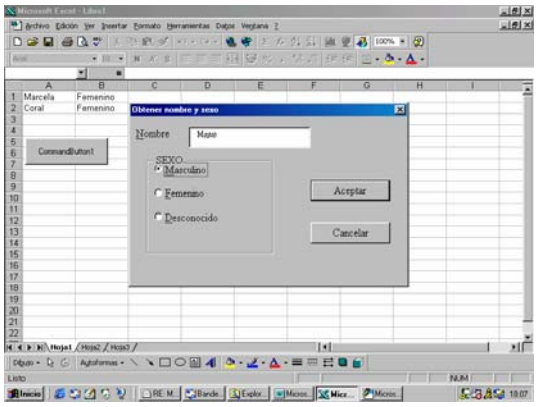
UserForms

AÑADIR PROCEDIMIENTOS DE CONTROLADOR DE EVENTO

```
Private Sub BotónAceptar_Click()
  Sheets("Hoja1").Activate
  Next = Row = application.WorkSheetFunction.CountA(Range("A:A")) + 1
  Cells(NextRow, 1) = NombreTexto.text
  If OpciónMasculino Then Cells(NextRow, 2) = "Masculino"
  If OpciónFemenino Then Cells(NextRow, 2) = "Femenino"
  If OpciónDesconocido Then Cells(NextRow, 2) = "Desconocido"
  NombreTexto = ""
  OpciónDesconocido  = true
  NombreTexto.SetFocus
End Sub
```

UserForms

AÑADIR PROCEDIMIENTOS DE CONTROLADOR DE EVENTO



UserForms

VALIDAR LOS DATOS

Este ejemplo no asegura que el usuario realmente introduce el nombre en el Cuadro de texto. El siguiente código se inserta en el procedimiento BotónAceptar_Click() antes de que el texto se transfiera a la hoja de cálculo.

```
´Asegurar que se ha introducido un nombre
If NombreTexto.Text = “” Then
        MsgBox “Se debe introducir un nombre”
        Exit Sub
End  If
```

Con esto se asegura que el usuario ingresa un texto. Si este está vacío aparece un mensaje y la rutina termina.

Parte 7:

Controles

PROPIEDADES COMUNES A TODOS LOS CONTROLES

Categoría apariencia

Backcolor y Forecolor: Color del fondo del control y color del texto.

Caption: Informa el texto que aparece en el control.

Picture: Indica el nombre de un fichero gráfico que se mostrará como fondo del control.

Categoría comportamiento

Enabled: Propiedad del tipo True/False que especifica si el control está activo o no en tiempo de ejecución; un control no activo es visible pero el usuario no puede interactuar con él, y se visualiza con un color distinto.

Visible: Otra propiedad True/False, que indica si el control está visible u oculto en tiempo de ejecución.

TabIndex: El orden por el cual nos movemos con la tecla TAB, entre los controles, se establece con esta propiedad. Es un valor numérico, 0, 1 ...

TabStop: Es del tipo True/False y establece si un control puede ser accesible con la teca TAB.

PROPIEDADES COMUNES A TODOS LOS CONTROLES

Categoría fuente

Font: Permite elegir el tipo, estilo, tamaño, etc... de letra del texto mostrado por el control.

Categoría posición

Left y Top: La primera contiene la coordenada columna y la segunda la coordenada línea de pantalla donde se sitúa el control.

Width y Height: Cuando un control es redimensionable, esto es, que sus dimensiones son variables, tendrá estas dos propiedades que nos informan del ancho y la altura del control.

EJEMPLO

1) Inserte un formulario en l proyecto

2) Inserte un botón de comando

3) Haga doble click sobre el botón para acceder a la ventana de código

4) En el procedimiento de evento click añada las siguientes líneas de código:

PROPIEDADES COMUNES A TODOS LOS CONTROLES

EJEMPLO

```
Private Sub CommandButton_Click()
        CommandButton1_Height = UserForm1.Height / 2
        CommandButton1_Width = UserForm1.Width / 2
        CommandButton1_Left = UserForm1.Width / 4
        CommandButton1_Top = UserForm1.Height / 4
        CommandButton1_Caption = “redimensionado”
End Sub
```

5) Ejecute el procedimiento presionando la tecla F5 y observe como cambia el título del control CommandButton y su tamaño y posición al hacer clic sobre él.

EVENTOS COMUNES A TODOS LOS CONTROLES

Para nombrar un evento correspondiente a un control, la nomenclatura es siempre la misma:

nombre_del_control.nombre_del_evento

Command Button.Click

Click: Se activa al hacer clic sobre el control, pero también al pulsar la barra de espacios o la tecla Enter si se trata de un botón CommandButton

MouseMove: Se activa al mover el puntero del ratón por encima del control.

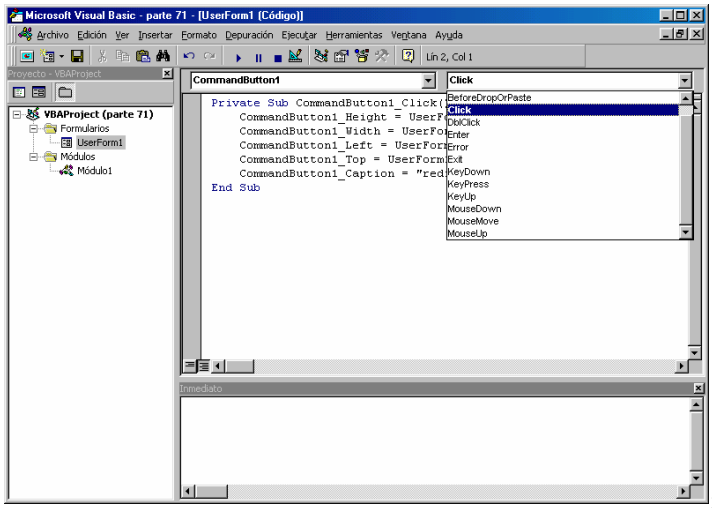
EJEMPLO

1) Acceda a la ventana de código del control command Button del ejemplo anterior.

2) Observe que en la parte superior de la ventana de código hay dos listas desplegables. La de la izquierda muestra el nombre del control CommandButton1 y es una lista de los controles del UserForm. La de la derecha muestra el nombre del evento Click y es una lista de todos los eventos del control que tenemos seleccionado.

En la ventana de elección de eventos, desplegándola, elija el evento MouseMove.

EVENTOS COMUNES A TODOS LOS CONTROLES



EVENTOS COMUNES A TODOS LOS CONTROLES

3) Escriba el código necesario para que el evento quede como sigue:

```
Private Sub CommandButton1_MouseMove(ByVal Button As Integer, ByVal Shift As
Integer, ByVal X As Single, ByVal Y As Single)
        CommandButton1.Caption = "Ahora pasas por encima"
End Sub
```

4) Despliegue ahora la lista de controles y elija el objeto UserForm. En la lista de eventos del UserForm elija de nuevo MouseMove y escriba el código siguiente:

```
Private Sub UserForm_MouseMove(ByVal Button As Integer, ByVal Shift As Integer,
ByVal X As Single, ByVal Y As Single)
        CommandButton1.Caption = "Desactivado"
End Sub
```

5) En un módulo de VBA escriba el siguiente código:

```
Sub mostrar()
        UserForm1.Show
End Sub
```


EVENTOS COMUNES A TODOS LOS CONTROLES

6) Ejecute el procedimiento Mostrar. Mueva el puntero del ratón por encima del control sin hacer clic y observe como cambia el texto que muestra. Cuando el puntero abandona el área del botón y se mueve por encima del UserForm se activa el evento MouseMove de éste y el título del botón muestra el texto “desactivado”.

MouseDown y MouseUp: El primer evento se activa al presionar uno de los botones del ratón y el segundo al liberarlo

KyePress, KeyDown y KeyUp: Son eventos que relacionan un control que acepta entrada de texto por parte del usuario (por ejemplo TextBox) con el teclado del ordenador. Cuando el usuario presiona una tecla de tipo carácter se activa el evento KeyPress. Si presiona una tecla especial, como puede ser la tecla “Inicio” o “Enter”, se activa solo el evento KeyDown. Al soltar la tecla se activa KeyUp. Notemos que el evento KeyDown se activa para cualquier tecla, mientras KeyPress sólo lo hace si la tecla es de tipo carácter.

DblClick: Se activa al hacer doble clic sobre el control.

EJEMPLO DEL BOTÓN CHECKBOX

Diseñemos un UserForm de entrada de datos de los clientes de un hotel. Ha de contener DNI, nombre día de llegada, habitación doble (sí/no), habitación con baño completo (sí/no), pensión completa (sí/no), suplemento cama niño (sí/no).

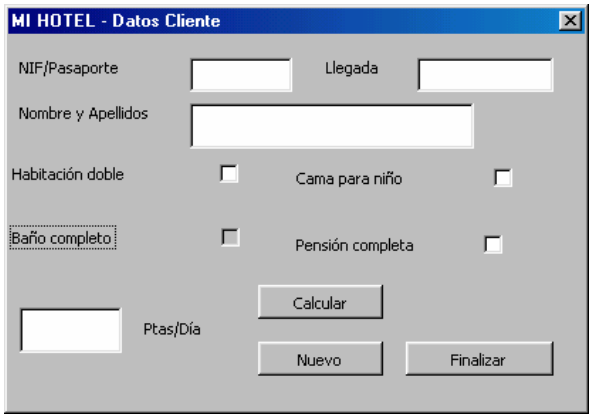
El programa calculará el precio diario de la habitación en base a una suma de conceptos. La habitación tiene un recio de 5000 pts/día, y los precios de los suplementos son:

Habitación doble: 3000 pts/día, Con baño completo: 2000 pts/día, Pensión completa (por persona): 4500 pts/día, Cama suplementaria para niño: 1500 pts/día).

Los pasos a seguir son:

1) Diseñe un formulario com. se muestra a continuación:

EJEMPLO DEL BOTÓN CHECKBOX



2) Acceda a las propiedades Caption de los controles de forma que queden como en la pantalla anterior.

3) Ponga la propiedad Visible del TextBox que muestra el resultado del cálculo a False. Haga lo mismo con el correspondiente control Label.

EJEMPLO DEL BOTÓN CHECKBOX

4) Acceda a la ventana del código del botón “Finalizar”, evento Click, y escriba la instrucción End

```
Private Sub CommandButton3_Click()
  End
End Sub
```

5) Acceda a la ventana de código del botón “Nuevo” y déjelo como sigue:

```
Private Sub CommandButton2_Click()
        TextBox1 = ""
        TextBox2 = ""
        TextBox3 = ""
        HabDoble = false
        PensionCompleta = false
        CamaAdicional = false
        BañoCompleto = false
        TextBox4.Visible = False
        Label4.Visible = False
End Sub
```

Cambie el nombre de los botones CheckBox, HabDoble, PensionCompleta, CamaAdicional, BañoCompleto

EJEMPLO DEL BOTÓN CHECKBOX

6) Acceda a la ventana de código del botón “Calcular” y déjelo como sigue:

```
Private Sub CommandButton1_Click()
Dim Precio As Integer
Precio = 5000
If HabDoble = true Then Precio = Precio + 3000
If PensionCompleta = true Then
  Precio = Precio + 4500
  If HabDoble = true Then
     Precio = Precio + 4500
  End If
  If CamaAdicional = true Then
     Precio = Precio + 4500
  End If
End If
If CamaAdicional = true Then
   Precio = Precio + 1500
End If
If BañoCompleto = true Then
    Precio = Precio + 2000
End If
   TextBox4 = Precio
   TextBox4.Visible = True
    Label4.Visible = True
End Sub
```

EJEMPLO DEL BOTÓN CHECKBOX

6) En un módulo de VBA escriba el siguiente código:

```
Sub Clientes()
  UserForm1.Show
End Sub
```

7) Luego ejecútelo

EL CONTROL LISTBOX

A continuación se presentan algunos puntos a tener en cuenta cuando se trabaja con controles de Cuadro de lista.

- Los elementos de un Cuadro de Lista se pueden recuperar desde un rango de celdas (especificadas por la propiedad RowSource) o pueden ser añadido usando un código de VBA (y usando el método AddItem).
- Un Cuadro de lista se puede configurar para permitir una selección de una celda o una selección múltiple. Esto está determinado por la propiedad MultiSelect.
- No es posible desplegar un Cuadro de lista sin elementos seleccionados (la propiedad ListIndex es -1). Sin embargo, una vez se ha seleccionado un elemento, no es posible no seleccionar ningún elemento.
- Un Cuadro de lista puede contener columnas múltiples (controladas por la propiedad ColumnCount) e incluso un encabezado descriptivo (controlado por la propiedad ColumnHeads).
- Los elementos de un Cuadro de lista se pueden presentar como Casillas de verificación si se permite una selección múltiple, o como Botones de opción si se permite una selección de una sola celda. Esta operación está controlada por la propiedad ListStyle.

EL CONTROL LISTBOX

Añadir elementos al control Cuadro de lista

Antes de desplegar un Userform que use un control Cuadro de lista, probablemente se necesite rellenar el mismo con elementos. Esto se debe realizar en tiempo de diseño, usando elementos guardados en un rango de hoja de cálculo, o en tiempo de ejecución, usando VBA para añadir los elementos.

Los ejemplos que veremos a continuación suponen que:

Se ha generado un cuadro de diálogo en un UserForm llamado UserForm1.

- Este cuadro de diálogo contiene un control de Cuadro de lista llamado ListBox1.
- El libro de trabajo contiene una hoja llamada Hoja1 y un rango A1:A12 que contiene los elementos a desplegar en el Cuadro de lista.

Añadir elementos a un Cuadro de lista en tiempo de diseño

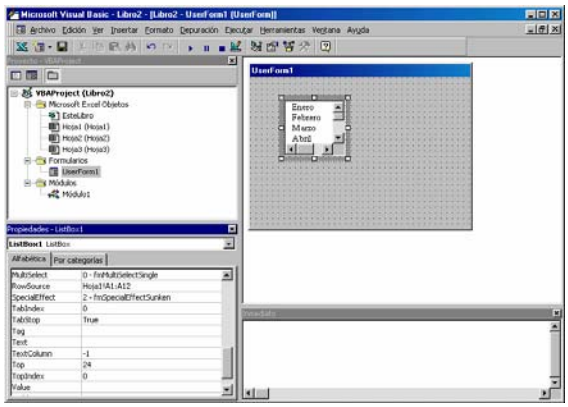
Para añadir elementos en tiempo de diseño los elementos deben estar guardados en en un rango de hoja de cálculo. Se usa la propiedad **RowSource** para especificar el rango que contiene dichos elementos (HOJA1!A1:A2)

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

EL CONTROL LISTBOX

Añadir elementos a un Cuadro de lista en tiempo de diseño



EL CONTROL LISTBOX

Añadir elementos a un Cuadro de lista en tiempo de ejecución

Para añadir elementos a un Cuadro de lista en tiempo de ejecución existen dos formas:

- Configurar la propiedad RowSource para una dirección de rango usando un código.
- Escribir un código que usa el método AddItem para añadir los elementos al Cuadro de lista.

```
UserForm1.ListBox1.RowSource = “Hoja1!A1:A12”
```

Si los elementos no están contenidos en un rango de hoja de cálculo, se puede escribir un código VBA para rellenar el cuadro de lista antes de que aparezca el cuadro de diálogo (con el método AddItem).

EL CONTROL LISTBOX

Añadir elementos a un Cuadro de lista en tiempo de ejecución

```
Sub ShowUserForm1()
  With UserForm1.ListBox1
        .RowSource = “”
        .AddItem “Enero”
        .AddItem “Febrero”
        .AddItem “Marzo”
        .AddItem “Abril”
        .AddItem “Mayo”
        .AddItem “Junio”
        .AddItem “Julio”
        .AddItem “Agosto”
        .AddItem “Septiembre”
  End With
  UserForm1. Show
End Sub
```

EL CONTROL LISTBOX

Añadir elementos a un Cuadro de lista en tiempo de ejecución

También se pude usar el método AddItem para recuperar elementos de un Cuadro de Lista a partir de un rango.

```
For  Row = 1 To 12
    UserForm1.ListBox1.AddItem Sheets("Hoja1").Cells(Row,1)
Next Row
```

Si los datos están contenidos en una matriz de una dimensión **meses**, que contiene 12 elementos, podemos escribir

```
ListBox1.List = meses
```


EL CONTROL LISTBOX

Selección de las opciones en el ListBox

La propiedad MultiSelect que en el momento de crear el control tiene el valor 0-Single, que permite seleccionar sólo una opción, puede tomar los siguientes valores:

- 1-Simple: permite seleccionar más de un elemento simplemente pulsando el botón del ratón sobre cada uno
- 2-Extended: permite además seleccionar un rango de la lista combinando la tecla de mayúsculas con el botón del ratón. En este caso para seleccionar elementos aislados combinaremos la tecla Control con el ratón.

Acceso a la opción en el caso de selección simple

Para determinar el elemento que se ha seleccionado, hay que acceder a la propiedad **Value** del Cuadro de lista.

```
MsgBox ListBox1.Value
```

Si se necesita saber la posición del elemento seleccionado se puede acceder a la propiedad **ListIndex** del Cuadro de lista.

```
MsgBox "Se ha seleccionado el elemento " & ListBox1.ListIndex
```

EL CONTROL LISTBOX

Acceso a las opciones en multiselección

Cuando tengamos la multiselección activada (la propiedad MultiSelect es 1 ó 2) podrá haber más de un elemento seleccionado, y las propiedades anteriores no serán adecuadas. Utilizaremos la propiedad **Selected** que es una matriz unidimensional de elementos tipo Boolean. El número de elementos es el mismo que el de la propiedad List.

Si el tercer elemento está seleccionado, entonces el tercer elemento de Selected tendrá el valor True, en caso contrario valdrá False. También nos puede ser útil la propiedad **ListCount**, que devuelve el número total de elementos de la lista.

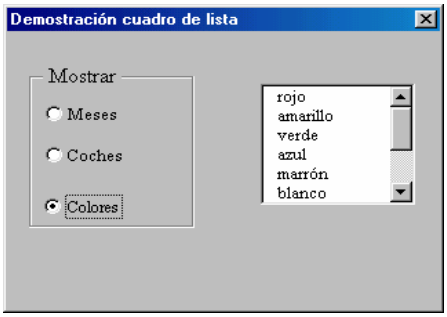
Crear un cuadro de lista con contenido variable

Este ejemplo demuestra cómo crear un Cuadro de lista cuyo contenido cambia, dependiendo de la selección del usuario de un grupo de botones de opción. El cuadro de lista obtiene los elementos de un rango de hoja de cálculo. El procedimiento que controla el evento **Click** de los controles de Botón de opción simplemente establece la propiedad **RowSource** del Cuadro de lista en un rango diferente.

EL CONTROL LISTBOX

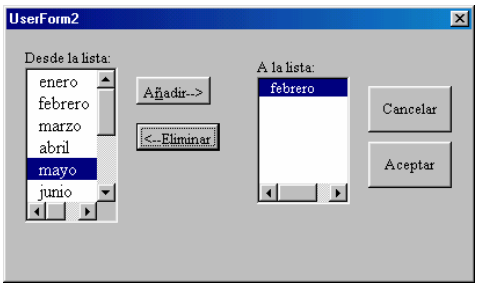
Crear un cuadro de lista con contenido variable

```
Private Sub OpcionCoches_Click()
   ListBox1.RowSource = "Hoja1!Coches"
End Sub

Private Sub OpcionColores_Click()
   ListBox1.RowSource = "Hoja1!Colores"
End Sub

Private Sub OpcionMeses_Click()
   ListBox1.RowSource = "Hoja1!Meses"
End Sub
```



EL CONTROL LISTBOX

Elaborar un Cuadro de lista desde otra lista



A continuación se muestra el procedimiento que se ejecuta cuando el usuario hace clic sobre el botón Añadir

```
Private Sub Añadir_Click()
  If ListBox1.ListIndex = -1 Then Exit Sub
  ListBox2.AddItem ListBox1.Value
End Sub
```


Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

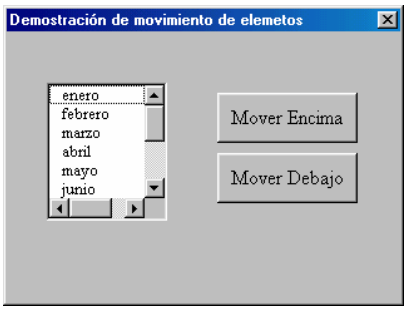
EL CONTROL LISTBOX

Elaborar un Cuadro de lista desde otra lista

A continuación se muestra el procedimiento que se ejecuta cuando el usuario hace clic sobre el botón Eliminar

```
Private Sub Eliminar_Click()
  If ListBox2.ListIndex = -1 Then Exit Sub
  ListBox2.RemoveItem ListBox2.ListIndex
End Sub
```

Mover elementos de un cuadro de lista



EL CONTROL LISTBOX

Mover elementos de un cuadro de lista

```
Private Sub MoverEncima_Click()
  With ListBox1
    ItemNum = .ListIndex
    If ItemNum > 0 Then
       TempItem = .List(ItemNum - 1)
       .List(ItemNum - 1) = .List(ItemNum)
       .List(ItemNum) = TempItem
       .ListIndex = .ListIndex - 1
    End If
  End With
End Sub
```


EL CONTROL LISTBOX

Mover elementos de un cuadro de lista

```
Private Sub MoverDebajo_Click()
  With ListBox1
    ItemNum = .ListIndex
    If ItemNum < .ListCount - 1 And ItemNum <> -1 Then 0 Then
       TempItem = .List(ItemNum +1)
       .List(ItemNum +1 1) = .List(ItemNum)
       .List(ItemNum) = TempItem
       .ListIndex = .ListIndex + 1
    End If
  End With
End Sub
```

EL CONTROL MULTIPAGE

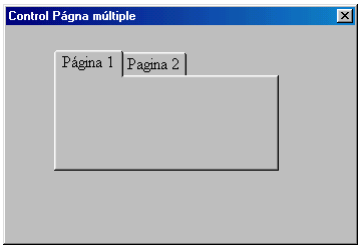
El control Página Múltiple es muy útil para cuadros de diálogo personalizados que deben presentar muchos controles. El mencionado control permite agrupar las opciones y colocar cada grupo en una “ficha” aparte.

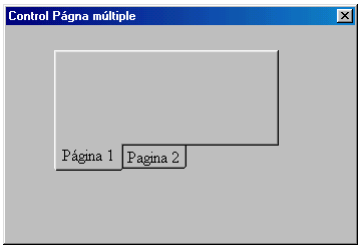
Hay que tener en cuenta lo siguiente cuando se use este control:

- La ficha (o página) que se despliega al frente está determinada por la función Value el control.El valor 0 despliega la primera ficha, el 1 la segunda y así sucesivamente.
- De forma predeterminada, un control de Página Múltiple tiene dos páginas. Para añadir una nueva, se hace clic con el botón derecho del ratón sobre una ficha y se selecciona Nueva Página desde el menú contextual.
- Cuando se está trabajando con un control de Página Múltiple, basta con hacer clic sobre una ficha para establecer las propiedades de esa página en concreto. La ventana Propiedades pesenta las propiedades que se pueden ajustar.
- Puede ser difícil seleccionar el control de Página múltiple, porque al hacer clic sobre el mismo, se selecciona toda la página. Para seleccionar el control propiamente dicho se puede usar la tecla Tab para realizar un recorrido en círculo por todos los controles. También se puede seleccionar el control Página múltiple desde la lista desplegable de la ventana Propiedades.

EL CONTROL MULTIPAGE

- Si el control página Múltiple consta de muchas fichas, se puede establecer su propiedad multiRow en True para desplegar las fichas en más de una fila.
- Si se prefiere se pueden desplegar botones en lugar de fichas. Para ello se cambia la propiedad Style a 1.
- La propiedad TabOrientation determina la localización de las fichas en el control Página Múltiple





CONTROL DE ERRORES DE EJECUCIÓN

- Los errores de ejecución de tipo externo suelen interrumpir de form súbita la ejecución normal del programa. Dependiendo del tipo de error, o bien VBA o bien Windows mostrarán un mensaje de error que no siempre será comprensible para el usuario.Veremos ejemplos de cómo hacer que nuestro programa controle estos errores sin terminar de forma anormal y generando nuestros propios mensajes de rror para el usuario.

EJEMPLO

```
Sub grabar()
   On Error goTo Problemas
   Application.SaveWorkSpace
   On Error GoTo 0
   Exit Sub
Problemas:
   MsgBox “operación cancelada”
   Resume Next
EndSub
```


CONTROL DE ERRORES DE EJECUCIÓN

- El método SaveWorkspace del objeto Application muestra el cuadro de diálogo estándar “Guardar Como”
- La instrucción On error GoTo activa el control de los errores de ejecución. Todos los errores ocurridos en la líneas uqe siguen a esta instrucción desviarán el flujo de ejecución a un bloque de código de tratamiento de errores. El control de errores se interrumpe con la instrucción On Error GoTo 0.

El esquema a seguir es:

```
Sub nombre_del_procedimiento()
   On Error GoTo Etiqueta ´Inicio del control
            (Instrucciones con control de errores)
   On errorGoTo 0 ´Fin del control
   Exit Sub ´Fuerza el final del procedimiento
Etiqueta:  Ínicio del bloque de tratamiento de errores
   (Acciones a seguir en caso de error)
Resume Next  ´Fin el bloque
End Sub
```

CONTROL DE ERRORES DE EJECUCIÓN

- También podemos ignorar los errores producidos en un cierto bloque de código; dicho en otras palabras, las acciones a realizar en caso de error no existen, pero no queremos que el programa se detenga de forma anormal. En este caso el esquema a seguir es:

```
On error Resume  next  ´Inicio del control
   (Instrcciones con control de errores)
On Error GoTo 0     ´Fin del control
```

- Cualquier error en las instrucciones que tienen el control de errores activado será ignorado, y la ejecución seguirá por la siguiente instrucción. Obviamente este esquema es más cómo pero más peligroso.

EL CONTROL RefEdit

A continuación se presentan algunos temas a tener en cuenta cuando se use el control REfEdit:

- El control REfEdit devuelve una cadena de texto que representa una dirección de rango. Se puede convertir esta cadena en un objeto Range mediante el uso de una instrucción como la siguiente:

```
Set UserRange = Range(RefEdit1.Text)
```

- Desplegar la selección de rango actual es una buena práctica para inicializar el control RefEdit. Esto se puede hacer con la ayuda del procedimiento UserForm_Initialize usando una instrucción como la siguiente.

```
RefEdit1.Text = ActiveWindow.RangeSelection.Address
```

- No hay que dar por supuesto que el control RefEdit siempre va a devolver siempre la dirección de rango válido, por lo tanto debemos verificar que sea realmente válido.

EL CONTROL RefEdit

```
On Error Resume Next
  Set UserRange = Range(RefEdit1.Text)
  If Err <> 0 Then
        MsgBox "El rango seleccionado no es válido"
        RefEdit1.SetFocus
        On error GoTo 0
        Exit Sub
  End If
```

- Si el rango seleccionado correponde aun hoja diferente de la activa deberá indicarlo de la siguiente forma:

Hoja2!A1:A20

EL CONTROL RefEdit

EJEMPLO

EL CONTROL RefEdit

EJEMPLO

```
Sub Cálculo()
   UserForm1.Show
End Sub

Private Sub CommandButton1_Click()
   RefEdit1.Text = ""
   TextBox1 = ""
End Sub

Private Sub CommandButton2_Click()
  End
End Sub
```


EL CONTROL RefEdit

EJEMPLO

```
Private Sub Opción_Producto_Click()
   Set UserRange = Range(RefEdit1.Text)
   prod = 1
   For Each cell In UserRange
     prod = prod * cell.Value
   Next cell
   TextBox1 = prod
End Sub
```

```
Private Sub Opción_Suma_Click()
    Set UserRange = Range(RefEdit1.Text)
    Sum = 0
    For Each cell In UserRange
       Sum = Sum + cell.Value
     Next cell
     TextBox1 = Sum
End Sub
```

Parte 8:

Rangos

TRABAJAR CON RANGOS

Copiar un rango (macro)

```
Sub Copiar_Rango()
   Range("A4:A8").Select
   Selection.Copy
   Range("C4").Select
   ActiveSheet.Paste
End Sub
```

Se puede simplificar de la siguiente manera:

```
Sub Copiar_Rango()
   Range("A4:A8").Copy  Range("C4")
End Sub
```

Ambas macros suponen questá activa una hoja de cálculo y que la operación tiene lugar en la hoja de cálculo activa.

TRABAJAR CON RANGOS

Copiar un rango

Para copiar un rango a una hoja de cálculo o libro diferente , simplemente se cualifica la referencia del rango para el destino. El siguiente ejemplo copia un rango desde la Hoja1 del libro Archivo1.xls, a la Hoja2 de libro Archivo2.xls.

```
Sub Copiar_Rango()
    Set Rango1 =WorkBooks(Archivo1.xls).Sheets(“Hoja1”).Range(“A1:A8”)
    Set Rango2 =WorkBooks(Archivo2.xls).Sheets(“Hoja2”).Range(C4“”)
    Rango1.Copy Rango2
End Sub
```

Mover un rango

```
Sub Mover_Rango()
   Range(“A1:A8”).Cut Range(“C4”)
End Sub
```


TRABAJAR CON RANGOS

Copiar un rango de tamaño variable

En muchos casos, es necesario copiar un rango de celdas, pero no se conocen exactamente las dimensiones de la fila y de la columna. Por ejemplo, se puede tener libro de trabajo que realiza el seguimiento de ventas semanales. El número de filas cambia semanalmente a medida que se introducen nuevos datos.

La siguiente macro demuestra cómo copiar un rango desde la Hoja! A la Hoja2 (comenzando por la celda A1). La macro usa la propiedad CurrentRegion que devuelve un objeto Range que corresponde al bloque de celdas alrededor de una celda concreta.

```
Sub Copiar_Rango_Variable()
    Range(“A1”).CurrentRegion.Copy Sheets(“Hoja2”).Range(“A1”)
End Sub
```

TRABAJAR CON RANGOS

Seleccionar rangos múltiples

```
Sub Selección_Múltiple()
   Range("A4:A8,C4:C9,B10:B13").Select
End Sub
```

Ejemplo de selcción de rangos

La siguiente instrucción selecciona un rango desde la celda activa hasta la última celda no vacía:

```
Range(ActiveCell, Activecell.End(xlDown)).select
```

Tres constantes simulan las combinaciones de teclas en las otras direcciones: xlUp, xlToLeft y xlToRight.

TRABAJAR CON RANGOS

Ejemplo de selcción de rangos

```
Sub Selección_Región_Actual()
    ActiveCell.CurrentRegion.Select
End Sub
```

El siguiente procedimiento crea un objeto Range y después aplica el formato al rango.

```
Sub Formato_Región_Actual()
    Set WorkRange = ActiveCell.CurrentRegion.Select
    WorkRange.Font.Bold = True
End Sub
```

TRABAJAR CON RANGOS

Solicitar el valor de la celda

El siguiente procedimiento demuestra cómo preguntar al usuario por un valor y después insertarlo en la celda A1 de libro de trabajo activo:

```
Sub OtenerValor1()
    Range("A1").Value = InputBox("Introducir el valor ")
End Sub
```

Validar la entrada

```
SunObtenerValor3()
    MinVal = 1
    MaxVal = 12
    Msg = "Introducir un valor entre 1 y 12"
    ValidEntry = False
    Do
        UserEntry =InnputBox(Msg)
        If IsNumeric(UserEntry) then
          If UserEntry >= 1 And UserEntry <= 12
            ValidEntry = True
          Else
            Msg = "El valor anterior no era válido"
            Msg = Msg & vbCrLf
            Msg = Msg & "Introduzca un valor entre 1 y 12"
          End If
          If UserEntry = "" Then Exit Sub
    Loop Until ValidEntry
    ActiveSheet.Range("A1").Value = UserEntry
End Sub
```


TRABAJAR CON RANGOS

Introducir un valor en la siguiente celda vacía

El siguiente ejemplo solicita un nombre y un valor al usuario, y después introduce el texto en la siguiente fila vacía.

```
Sub ObtenerDatos()
Do
    NextRow = Application.WorkShhetFunction.CountA(Range(A:A))+1
    Entry1 = InputBox("Introducir el nombre")
    If Entry1 = "" Then Exit Sub
    Entry2 = InputBox("Introducir la cantidad")
    If Entry2 = "" Then Exit Sub
    Cells(Next,Row, 1) = Entry1
    Cells(NextRow, 2) = Entry2
Loop
End Sub
```

Se ha usado la función CountA, para contar el número de entradas en la columna A y después incrementar su valor en uno.

Esto funciona solo sie el rango comienza en A1, y no contiene celdas vacías.

TRABAJAR CON RANGOS

Contar celdas seleccionadas

La siguiente instrucción despliega un cuadro de mensaje que contiene el número de celdas de la selección actual.

```
MsgBox  Selection.Count
```

Si la hoja activa contiene un rango llamado datos, la siguiente instrucción asigna al número de celdas en el rango datos, auna variable llamada ContarCeldas.

```
ContarCeldas = Range("datos").Count
```

También se puede determinar la cantidad de columnas o de filas que están contenidas en un rango.- La siguiente expresión calcula el número de columnas en el rango actual seleccionado.

```
Selection.Columns.Count
```

y para las filas : ContarFila = Range("datos").Rows.Count

TRABAJAR CON RANGOS

Iterar eficientemente por un rango seleccionado

Una tarea común es crear una macro que evalúe cada celda de un rango y realice una operación si la celda reúne un determinado criterio. En este ejemplo el procedimiento Colorselectivo1 aplica un color de relleno rojo a todas las celdas de la selección que tienen un valor negativo.

```
Sub ColorSelectivo1()
  If TypeName(Selection) <> "Range" then Exit Sub
  Const = REDINDEX = 3
  Application.ScreenUpdating = False
  For each cell In Selection
        If cell.value < 0 Then
              cell.Interior.ColorIndex = REDINDEX
        Else
              cell.Interior.ColorIndex = xlNone
        End If
Next Cell
End Sub
```

TRABAJAR CON RANGOS

Eliminar todas las fila vacías

El siguiente ejemplo elimina todas las filas vacía de un libro de trabajo activo. Controla sólo las filas del "rango usado", que está determinado por el empleo de la propiedad UsedRange del objeto WorkSheet.

```
Sub EliminarFilasVacías()
        LastRow = ActiveSheet.usedRange.Rows.Count
        Application.ScreenUpdating = False
        For r = LastRow To 1 Step -1
          If Application.WorksheetFunction.CountA(Rows(r)) = 0 Then _
                                                Rows(r).Delete
        Next r
End Sub
```


TRABAJAR CON RANGOS

Leer y escribir rangos

Muchas tarreas de hoja de cálculo implican la transferencia de valores desde una matriz a un rango, o desde un rango a una matriz.

```
Sub LeeryEscribirRango()
        Dim MiMatriz()
        NumElements = 60000
        ReDim MiMatriz(1 To NumElements)
        For i = 1 To NumElements
                MiMatriz(i) = i
        Next i
        For i To NumElements
                Cells(i,1) = i
        Next i
End Sub
```

TRABAJAR CON RANGOS

FUNCIÓN InputBox

La sintaxis para la función InputBox de VBA es

InputBox(prompt[, title[,default[,xpos[, ypos[,helpfile, context)

prompt Requerido. Eltexto desplegadob en el cuadro de entrada

title Opcional. El título de la ventana del cuadro de entrada

default Opcional. El valor predeterminado que se va a desplegar en el cuadro de diálogo

xpos, ypos Opcional. Coordenadas de la pantalla de la esquina superior izquierda de la ventana

helpfile, context Opcional. Archivo de ayuda y tema de ayuda

La función InputBox solicita información al usuario. La función siempre devuelve una cadena,por lo que puede ser necesario convertir los resultados a un valor.

Escuela Superior de Informatica
Universidad de Castilla-La Mancha
http://www.inf-cr.uclm.es

Excel con VBA

TRABAJAR CON RANGOS

FUNCIÓN InputBox

```
Sub GetName()
        Do Until UseName <> “”
          UserName = InputBox(“Introduzca su nombre completo: “, _
                              “Identificación”)
        Loop
        FirtSpace = InStr(UserName, “ “)
        If FirstSpace <> 0 Then
                UserName = Left(UserName, FirstSpace)
        End If
        MsgBox “Hola “ & UserName
End Sub
```

TRABAJAR CON RANGOS

Función MsgBox

La sintaxis de la función MsgBox de VBA es:

MsgBox (propmt[,buttons[,title[helpfile, context,)

prompt. Requerido. El texto desplegado en el mensaje

buttons. Opcional. Una expresión numérica que determina los botones y el icono que se va a desplegar en el mensaje

title. Opiconal. El título de la ventana del cuadro del mensaje

helpfile, context Opcional.Archivo de ayuda y temad de ayuda

Constantes usadas por buttons en la función MsgBox

vbOKOnly: 0, despliega sólo el botón Aceptar

vbOKCancel: 1, despliega los botones Aceptar y cancelar

vbAbortRetryIgnore: 2, despliega los botones Salir, reintentar e Ignorar

vbYesNoCancel: 3, despliega los botones Sí, No y Cancelar

vbYesNo: 4, despliega los botones Sí y No

vbRetryCancel: 5, despleiga los botones Reintentar y Cancelar

Parte 9:

Gráficos

TRABAJAR CON GRÁFICOS

En Excel, un gráfico se puede situar en dos lugares dentro de un libro de trabajo:

- Como un objeto incrustado en un hoja de cálculo (una hoja puede contener cualquier número de gráficos incrustados).
- En una hoja de gráfico aparte (una hoja de gráfico contiene un solo gráfico).

MODELO DE OBJETOS GRÁFICOS (CHART)

Jerarquía para un gráfico incrustado

Applicattion, WorkBook, WorkSheet, ChartObject, Chart, ChartTitle, Characters

Por ejemplo, para establecer el título del gráfico en “Ventas del Año” , se puede escribir la siguiente instrucción:

```
WorkSheets(“Hoja1”).ChartObjects(1).Chart.ChartTitle. Characters.Text=
                                                    “Ventas del Año”
```


MODELO DE OBJETOS GRÁFICOS (CHART)

Jerarquía para una hoja de gráfico

Applicattion, WorkBook, Chart, ChartTitle, Characters

Por ejemplo, para establecer el título del gráfico en “Ventas del Año” , se puede escribir la siguiente instrucción:

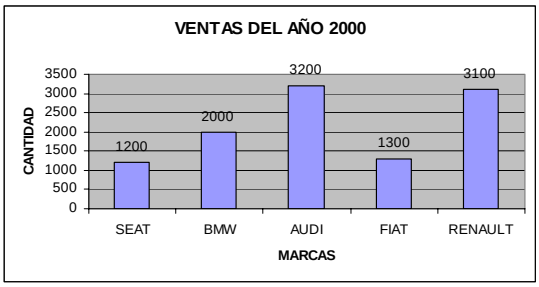
```
Sheets(“Gráfico1”).ChartTitle. Characters.Text  =  “Ventas del Año”
```

En otras palabras, una hoja de gráficoes un objeto de clase Chart y no contiene objeto de clase ChartObject.

GRABAR MACROS DE GRÁFICO

Quizá el mejor modo de familiarizarse con el modelo de objeto de clase Chart sea activar el Grabador de macro mientras se crean y manipulan gráficos.

	AÑO 2000
SEAT	1200
BMW	2000
AUDI	3200
FIAT	1300
RENAULT	3100



163


```
Sub grafico()
   Range("A2:E6").Select
   Charts.Add
   ActiveChart.ChartType = xlColumnClustered
   ActiveChart.SetSourceData Source:=Sheets("Hoja1").Range("A2:E6"), PlotBy:= _
      xlColumns
   ActiveChart.Location Where:=xlLocationAsObject, Name:="Hoja1"
   With ActiveChart
      .HasTitle = True
      .ChartTitle.Characters.Text = "VENTAS DEL AÑO 2000"
      .Axes(xlCategory, xlPrimary).HasTitle = True
      .Axes(xlCategory, xlPrimary).AxisTitle.Characters.Text = "MARCAS"
      .Axes(xlValue, xlPrimary).HasTitle = True
      .Axes(xlValue, xlPrimary).AxisTitle.Characters.Text = "CANTIDAD"
   End With
   ActiveChart.HasLegend = False
   ActiveChart.ApplyDataLabels Type:=xlDataLabelsShowValue, LegendKey:=False
End Sub
```

164

Cuando se genera un gráfico usando el método **Add** de la colección Charts, el gráfico es siempre una hoja de gráfico. En el código anterior, el método **Location** mueve el gráfico a una hoja de cálculo.

ACTIVAR UN GRÁFICO

El código VBA puede activar un gráfico incrustado usando el método Activate.

ActiveSheet.ChartObjects("Gráfico 1").Activate

Si el gráfico se encuentra en una hoja de gráfico, se usa una instrucciión como esta:

Sheets("Gráfico1")Activate

Una vez activado el gráfico, se puede hacer referencia a él en el código con ActiveChart. Por ejemplo, la siguiente instrucción despliega el nombre del gráfico activo. S no existe un gráfico activo, la instrucción genera un error.

MsgBox ActiveChart.name

Para modificar un gráfico en VBA, no es necesario activarlo. Veamos un ejemplo que cambia el tipo de gráfico a Areas.

```
Sub Modificar_Grafico
        ActiveSheet.ChartObjects("Grafico1").Chart.Type = xlArea
End Sub
```

Convertir un gráfico incrustado en una hoja de gráfico

```
Sub ConvertChart1()
   Sheets("Hoja1").ChartObjects(1).Chart.Location xlLocationAsNewShee, "Mi Gráfico"
End Sub
```

Convertir una hoja de gráfico en un gráfico incrustado

```
Sub ConvertChart2()
   Charts("Mi gráfico").Location xlLocationAsObject, "Hoja1"
End Sub
```

Determinar si un gráfico está activado

La siguiente función CharIsSelected devuelve Verdadero si está activa una hoja de grafico o si está activo un gráfico incrustado, y devuelve Falso si no está activado ningún gráfico:

```
Private Function ChartIsSelected() As boolean
        Dim x As String
        ChartIsSelected = False
        On Error Resume Next
        x = ActiveChart.Name
        if Err = 0 Then CharIsSelected = True
End Function
```

Eliminar ObjetosGráficos (ChartObjects) de gráficos (Charts)

Para eliminar objetos de clase **ChartObject** de un libro de trabajo, se puede usar simplemente el método **Delete** de la colección **ChartObjects**.

```
ActiveSheet.ChartObjects.Delete
```


Para eliminar hojas de gráfico de un libro de trabajo activo, se usa la siguiente instrucción:

```
ActiveWorkBook.Charts.Delete
```

Normalmente, si se eleminan hojas Excel presenta una advertencia. El usuario debe contestar a este aviso para que pueda continuar la macro. Para elimnar este aviso se usa la siguiente serie de instrucciones:

```
Application.DisplayAlerts = False
ActiveWork.Charts.Delete
Application.DisplayAlerts = False
```

Aplicar formato al gráfico

El siguiente ejemplo aplica diferentes tipos de formato al gráfico activo:

```
Sub ChartMod1()
        with ActiveChart
                .Type = xlArea
                .ChartArea.Font.Name = “Arial”
                .ChartArea.FontStyle = “Regular”
                .PlotArea.Interior.ColorIndex = xlNone
                .Axes(xlValue).TickLabels.Font.Bold = True
                .Axes(xlCategory).TickLabels.Font.Bold = True
                .HasLegend = True
                .Legend.Position = xlBottom
        End with
End Sub
```

Debe estar activo un gráfico o este procedimiento dará error.

Aplicar formato al gráfico

El siguiente ejemplo funciona en un gráfico concreto: el contenido en un ChartObject llamado Gráfico 1, ubicado en la Hoja1. Nótese que el gráfico nunca se activa.

```
Sub ChartMod2()
        with Sheets(“Hoja1”).ChartObjects(Gráfico 1”).Chart
                .Type = xlArea
                .ChartArea.Font.Name = “Arial”
                .ChartArea.FontStyle = “Regular”
                .PlotArea.Interior.ColorIndex = xlNone
                .Axes(xlValue).TickLabels.Font.Bold = True
                .Axes(xlCategory).TickLabels.Font.Bold = True
                .HasLegend = True
                .Legend.Position = xlBottom
        End with
End Sub
```

Bucle por todos los gráficos

En algunos casos es necesario ejecutar una operación en todos los gráficos. El siguiente ejemplo cambia el tipo de gráfico para cada gráfico incrustado en una hoja activa.

```
Sub Cambiar_Tipo_Grafico1()
        For each cht In ActiveSheet.ChartObjects
                cht.Chart.Type =  xlArea
        Next cht
End Sub
```

El siguiente procedimiento realiza la misma operación que el procedimiento anterior pero funciona en todas las hojas de gráfico de un libro de trabajo activo:

```
Sub Cambiar_Tipo_Grafico2()
        For each cht In ActiveWorkBook.Charts
                cht.Type =  xlArea
        Next cht
End Sub
```


Bucle por todos los gráficos

```
Sub Cambiar_Leyenda()
        For each cht In ActiveSheet.ChartObjects
                With cht.hart.Legend.Font
                        .Name = “Arial”
                        .FontStyle =  “Bold”
                        .Size = 12
                End With
        Next cht
End Sub
```

Alinear y ajustar el tamaño de los ObjetosGráficos (ChartObjects)

Un objeto de clase ChartObject posee propiedades de posición y tamaño estándar, a las que se puede acceder con el código VBA. El siguiente ejemplo ajusta el tamaño de todos los objetos de la clase ChartObject en HOJA1, de forma que se ajusten a las dimensiones del ChartObject llamado Gráfico 1. También ordena los objetos de clase ChartObject de forma que aparezcan uno después del otro a lo largo de la izquierda de la hoja de cálculo:

Alinear y ajustar el tamaño de los ObjetosGráficos (ChartObjects)

```
Sub Tamaño_Posición_Gráficos
        W = ActiveSheet.ChartObjects("Gráfico 1").Width
        H = ActiveSheet.ChartObjects("Gráfico 1").Height
        TopPos = 0
        For each chtObj in ActiveSheet.ChartObjects
                with chtObj
                        .Width = W
                        .Height = H
                        .Left = 0
                        .Top = TopPos
                End With
                TopPos = TopPos + H
        Next chtObj
End Sub
```


Presentar un gráfico en un UserForm

Supongamos que tenemos un procedimiento grafico que genera un gráfico incrustado en la hoja1.

```
Sub ShowChart()
  grafico
  UserForm1.Show
End Sub
Private Sub UserForm1_Initialize()
    Set CurrentChart = ActiveSheet.ChartObjects(1).Chart
    Fname = ThisWork.Path & "\temp.gif"
    CurrentChart.Export FileName:=Fname, FilterName:="GIF"
    Sheets("Hoja1").ChartObjects1(1).Delete

    Image1.Picture = LoadPicture(Fname)
    Application.ScreenUpdating = True
    Kill ThisWorkbook.Path & "\temp.gif"
End Sub
```