MS-Excel - VBA

Comprendiendo objetos, propiedades, métodos y eventos

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

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

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

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

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

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

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

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

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

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

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

```
End Sub
```

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

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

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

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

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

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

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

Para un mejor conocimiento de los conceptos: procedimiento, objeto, propiedades, métodos y eventos, ver en el glosario.

Ejemplo de un procedimiento:

```
Sub Command125_Click ()
  Range("A2").Value= 7
  Application.Close
End Sub
```

"Sub" es el procedimiento; "Command125_Click" es el evento; "Range ("A2")" y "Application" son objetos; "Value" es una propiedad, y "Close" es un método.

Este procedimiento dice que cuando el usuario 'clickee' sobre el botón de comando '125', la celda A2 tome el valor de 7, y luego cierre la aplicación (salir de Excel).

Reglas de asignación de nombres en Visual Basic

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

- El primer carácter debe ser una letra.
- En el nombre no se pueden utilizar espacios, puntos (**.**), signos de interjección (**!**), ni los caracteres **@**, **&**, **$, #**.
- El nombre no puede tener más de 255 caracteres de longitud.
- Como regla general, no se deben usar nombres iguales a los de los procedimientos Function, instrucciones y métodos de Visual Basic. No obstante puede utilizar las mismas palabras clave que utiliza el lenguaje. Para utilizar una función intrínseca del lenguaje, o una instrucción o método, cuyo nombre coincide con uno de los nombres asignados, es preciso entonces identificarlos explícitamente. Para ello se sitúa delante del nombre de la función intrínseca, instrucción o método, el nombre de la biblioteca de tipos asociada. Por ejemplo, si utiliza una variable llamada `Left`, la única forma de utilizar la función **Left** es escribiendo `VBA.Left`.
- Los nombres no se pueden repetir dentro del mismo nivel de alcance. Por ejemplo, no se pueden declarar dos variables con el nombre `edad` dentro del mismo procedimiento. Sin embargo, se puede declarar una variable privada `edad` y una variable de nivel de procedimiento llamada `edad` dentro del mismo módulo.

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

Declaración de variables

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

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

```
Dim NombreTexto As String
```

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

```
Public NombreTexto As String
```

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

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

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

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

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

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

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

Utilizar la instrucción Public

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

```
Public NombreTexto As String
```

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

Utilizar la instrucción Private

La instrucción **Private** se puede usar para declarar variables privadas de nivel de módulo, no dentro de procedimientos (allí se declara con Dim y las variables siempre son privadas al procedimiento).

```
Private MiNombre As String
```

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

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

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

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

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

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

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

Declaración de matrices

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

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

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

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

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

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

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

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

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

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

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

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

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

Declarar una matriz dinámica

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

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

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

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

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

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

Declaración de constantes

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

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

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

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

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

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

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

Creando procedimientos recursivos

Los procedimientos tienen un espacio limitado para almacenar variables. Cada vez que un procedimiento se llama a sí mismo, consume más de ese espacio. Un procedimiento que se llama a sí mismo es lo que se conoce como un procedimiento recursivo. Un procedimiento recursivo que se llama continuamente a sí mismo producirá finalmente un error. Por ejemplo:

```
Function Agotar(Máximo)
    Agotar = Agotar(Máximo)
End Function
```

Este error puede resultar menos evidente cuando dos procedimientos se llaman uno al otro de forma indefinida, o cuando nunca se cumple la condición definida como fin de un bucle. Las funciones recursivas tienen diversos usos. Por ejemplo, el siguiente procedimiento utiliza una función recursiva para calcular el factorial:

```
Function Factorial (N)
    If N <= 1 Then        ' Se ha llegado al fin de las llamadas recursivos.
        Factorial = 1    ' (N = 0) abandona las llamadas.
    Else                   ' Llama nuevamente a Factorial si N > 0.
        Factorial = Factorial(N - 1) * N
    End If
End Function
```

Debe probar el procedimiento recursivo para comprobar que no se llama a sí mismo tantas veces que agota la memoria disponible. Si se produce un error, compruebe que el procedimiento no se llama a sí mismo de forma indefinida. Si no es así, trate de ahorrar memoria mediante:

- La eliminación de variables innecesarias.
- El uso de tipos de datos distintos a **Variant**.

- Un nuevo estudio de la lógica del procedimiento. A menudo es posible sustituir bucles anidados por un procedimiento recursivo.

Creando bucles mediante código

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

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

Elegir la instrucción condicional a utilizar:

- If...Then...Else: salta a una instrucción cuando una condición es **True** o **False**

- Select Case: selección de la instrucción a ejecutar en función de un conjunto de condiciones

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

Elegir el bucle a utilizar:

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

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

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

Utilizando instrucciones If...Then...Else

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

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

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

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

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

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

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

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

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

Utilizar instrucciones For...Next

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

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

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

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

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

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

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

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

Se puede abandonar una instrucción **For...Next** antes de que el contador alcance su valor final, para ello se utiliza la instrucción **Exit For**.

Instrucción SET

Set *variable_objeto* = {[**New**] *expresión_objeto* | **Nothing**}

La sintaxis de la instrucción **Set** consta de las siguientes partes:

Parte	Descripción
variable_objeto	Requerido. Nombre de la variable o de la propiedad; sigue las convenciones estándar de nombres de variables.
New	Opcional. **New** se utiliza normalmente durante una declaración para permitir la creación implícita de un objeto. Cuando utiliza **New** con la instrucción **Set** se crea una nueva instancia de la clase. Si *variable_objeto* contenía una referencia a un objeto, esta referencia se libera cuando se asigna el nuevo objeto. La palabra clave **New** no se puede utilizar para crear nuevas instancias de cualquier tipo de datos intrínseco ni para crear objetos dependientes.
expresión_objeto	Requerido. Expresión que consiste en el nombre de un objeto, otra variable declarada del mismo tipo de objetos, o una función o método que devuelve un objeto del mismo tipo de objeto.
Nothing	Opcional. Interrumpe una asociación de *variable_objeto* con cualquier objeto específico. Al asignar **Nothing** a *variable_objeto* se liberan todos los recursos del sistema y de memoria asociados con el objeto al que se hizo referencia previamente cuando ninguna otra variable se refiere a él.

Comentarios

Para ser válido, *variable_objeto* debe ser un tipo de objeto coherente con el objeto que se le ha asignado.

Las instrucciones **Dim**, **Private**, **Public**, **ReDim** y **Static** sólo declaran una variable que se refiere a un objeto. No se hará referencia a ningún objeto real hasta que use la instrucción **Set** para asignar un objeto específico.

El siguiente ejemplo ilustra el uso de Dim para declarar una matriz del tipo Form1. Actualmente no existe ninguna instancia de Form1. Set asigna referencias a nuevas instancias de Form1 a la variable misFormulariosSecundarios. Este código se podría utilizar para crear formularios secundarios en una aplicación MDI.

```
Dim misFormulariosSecundarios(1 to 4) As Form1
Set misFormulariosSecundarios(1) = New Form1
Set misFormulariosSecundarios(2) = New Form1
Set misFormulariosSecundarios(3) = New Form1
Set misFormulariosSecundarios(4) = Nwq Form1
```

Generalmente, cuando usa **Set** para asignar una referencia de objeto a una variable, no se crea ninguna copia del objeto para esa variable. En su lugar se crea una referencia al objeto. Más de una variable de objeto se puede referir al mismo objeto. Puesto que estas variables son referencias al objeto (no copias de él), cualquier cambio en el objeto se refleja en todas las variables que se refieren a él. No obstante, cuando utiliza la palabra clave **New** en la instrucción **Set**, en realidad está creando una instancia del objeto.

Conjunto Range

Representa una celda, una fila, una columna, una selección de celdas que contienen uno o más bloques contiguos de celdas o un rango 3D.

Uso del conjunto Range

Vamos a ver los siguientes métodos y propiedades para devolver un objeto **Range**:

- Propiedad **Range**
- Propiedad **Cells**
- **Range** y **Cells**
- Propiedad **Offset**
- Método **Union**

Propiedad Range

Use Range(arg), donde arg asigna un nombre al rango, para devolver un objeto Range que represente una sola celda o un rango de celdas. El ejemplo siguiente coloca el valor de la celda A1 en la celda A5.

```
Worksheets("Hoja1").Range("A5").Value = _
    Worksheets("Hoja1").Range("A1").Value
```

El ejemplo siguiente rellena el rango A1:H8 con números aleatorios estableciendo la fórmula de cada celda del rango. La propiedad Range, si se emplea sin un calificador de objeto (un objeto colocado a la izquierda del punto), devuelve un rango de la hoja

activa. Si la hoja activa no es una hoja de cálculo, este método no se llevará a cabo con éxito. Use el método Activate para activar una hoja de cálculo antes de usar la propiedad Range sin un calificador de objeto explícito.

```
Worksheets("Hoja1").Activate
Range("A1:H8").Formula = "=Rand()"     ' El rango está en la hoja activa
```

El ejemplo siguiente borra el contenido del rango denominado Criterios.

```
Worksheets(1).Range("Criteria").ClearContents
```

Si usa un argumento de texto para la dirección del rango, deberá especificar la dirección en notación de estilo A1 (no podrá usar la notación F1C1).

Propiedad Cells

Use Cells(fila; columna), donde fila es el índice de fila y columna es el índice de columna, para devolver una sola celda. El ejemplo siguiente establece en 24 el valor de la celda A1.

```
Worksheets(1).Cells(1, 1).Value = 24
```

El ejemplo siguiente establece la fórmula de la celda A2.

```
ActiveSheet.Cells(2, 1).Formula = "=Sum(B1:B5)"
```

Aunque también puede usar Range("A1") para devolver la celda A1, en algunas ocasiones la propiedad Cells puede ser más conveniente, ya que permite usar una variable para la fila o la columna. El ejemplo siguiente crea encabezados de fila y columna en la Hoja1. Tenga en cuenta que, después de activar la hoja de cálculo, puede usar la propiedad Cells sin una declaración explícita de hoja (devuelve una celda de la hoja activa).

```
Sub SetUpTable()
Worksheets("Hoja1").Activate
For Año = 1 To 5
    Cells(1, Año + 1).Value = 1990 + Año
Next Año
For Cuarto = 1 To 4
    Cells(Cuarto + 1, 1).Value = "Q" & Cuarto
Next Cuarto
End Sub
```

Aunque podría usar funciones de cadena de Visual Basic para modificar las referencias de estilo A1, es mucho más sencillo (y una mejor práctica de programación) usar la notación `Cells(1, 1)`.

Para devolver parte de un rango use *expresión*.**Cells**(*fila*; *columna*), donde *expresión* es una expresión que devuelve un objeto **Range** y *fila* y *columna* son relativas a la esquina superior izquierda del rango. El ejemplo siguiente establece la fórmula de la celda C5.

```
Worksheets(1).Range("C5:C10").Cells(1, 1).Formula = "=Rand()"
```

Ejemplos usando la propiedad cells al objeto worksheets

Este ejemplo establece en 14 puntos el tamaño de fuente de la celda C5 de Hoja1.

```
Worksheets("Hoja1").Cells(5, 3).Font.Size = 14
```

Este ejemplo borra la fórmula de la celda uno de Hoja1.

```
Worksheets("Hoja1").Cells(1).ClearContents
```

Este ejemplo establece en Arial de 8 puntos la fuente y el tamaño de fuente de todas las celdas de Hoja1.

```
With Worksheets("Hoja1").Cells.Font
    .Name = "Arial"
    .Size = 8
End With
```

Este ejemplo ejecuta un bucle en las celdas A1:J4 de Hoja1. Si una de las celdas contiene un valor menor que 0,001, el ejemplo reemplazará dicho valor por cero (0).

```
For rwIndex = 1 to 4
    For colIndex = 1 to 10
        With Worksheets("Hoja1").Cells(rwIndex, colIndex)
            If .Value < .001 Then .Value = 0
        End With
    Next colIndex
Next rwIndex
```

Este ejemplo establece el estilo de fuente de las celdas A1:C5 de Hoja1 como cursiva.

```
Worksheets("Hoja1").Activate
Range(Cells(1, 1), Cells(5, 3)).Font.Italic = True
```

Este ejemplo examina una columna de datos denominada "miRango". Si una celda contiene el mismo valor que la celda inmediatamente superior, el ejemplo mostrará la dirección de la celda que contiene los datos duplicados.

```
Set r = Range("miRango")
For n = 1 To r.Rows.Count
    If r.Cells(n, 1) = r.Cells(n + 1, 1) Then
        MsgBox "Hay duplicación en " & r.Cells(n + 1, 1).Address
    End If
Next n
```

Range y Cells

Para devolver un objeto **Range** use **Range**(*celda1; celda2*), donde *celda1* y *celda2* son objetos **Range** que especifican las celdas inicial y final. El ejemplo siguiente establece el estilo de línea de los bordes de las celdas A1:J10.

```
With Worksheets(1)
    .Range(.Cells(1, 1), _
        .Cells(10, 10)).Borders.LineStyle = xlThick
```

```
End With
```

Observe el punto delante de cada propiedad **Cells**. El punto es necesario si el resultado del enunciado **With** precedente se aplica a la propiedad **Cells**, en cuyo caso, se indica que las celdas están en la hoja de cálculo uno (sin el punto, la propiedad **Cells** devolvería las celdas de la hoja activa).

Propiedad Offset

Use **Offset**(*fila; columna*), donde *fila* y *columna* son los desplazamientos de fila y columna, para devolver un rango con un desplazamiento específico con respecto a otro. El ejemplo siguiente selecciona la celda situada tres filas debajo y una columna a la derecha de la celda de la esquina superior izquierda de la selección actual. No se puede seleccionar una celda que no esté en la hoja activa, por lo que primero deberá activar la hoja.

```
Worksheets("Hoja1").Activate
  'Can't select unless the sheet is active
Selection.Offset(3, 1).Range("A1").Select
```

Método Union

Use **Union**(*rango1, rango2, ...*) para devolver rangos de varias áreas, es decir, rangos compuestos por dos o más bloques contiguos de celdas. El ejemplo siguiente crea un objeto definido como la unión de los rangos A1:B2 y C3:D4 y, a continuación, selecciona el rango definido.

```
Dim r1 As Range, r2 As Range, myMultiAreaRange As Range
Worksheets("Hoja1").Activate
Set r1 = Range("A1:B2")
Set r2 = Range("C3:D4")
Set myMultiAreaRange = Union(r1, r2)
myMultiAreaRange.Select
```

La propiedad **Areas** es muy útil para trabajar con selecciones que contienen varias áreas. Divide una selección de varias áreas en objetos **Range** individuales y después devuelve los objetos en forma de conjunto. Puede usar la propiedad **Count** del conjunto devuelto para comprobar una selección que contiene varias áreas, como se muestra en el siguiente ejemplo.

```
Sub NoMultiAreaSelection()
    NumberOfSelectedAreas = Selection.Areas.Count
    If NumberOfSelectedAreas > 1 Then
        MsgBox "No puede utilizar este comando " & _
            "en una selección multi-áreas"
    End If
End Sub
```

Otras propiedades y ejemplos

Propiedad ActiveCell

Devuelve un objeto **Range** que representa la celda activa de la ventana activa (la ventana superior) o de la ventana especificada. Si la ventana no contiene una hoja de cálculo, esta propiedad fallará. Es de sólo lectura.

Comentarios

Si no especifica un calificador de objeto, esta propiedad devolverá la celda activa de la ventana activa.

Celda activa no es lo mismo que selección. La celda activa es una sola celda de la selección actual. La selección puede contener más de una celda, pero sólo una es la celda activa.

Todas las expresiones siguientes devuelven la celda activa y son equivalentes:

```
ActiveCell
Application.ActiveCell
ActiveWindow.ActiveCell
Application.ActiveWindow.ActiveCell
```

Ejemplo

Este ejemplo usa un cuadro de mensaje para mostrar el valor de la celda activa. Puesto que la propiedad **ActiveCell** falla si la hoja activa no es una hoja de cálculo, el ejemplo activará Hoja1 antes de utilizar la propiedad **ActiveCell**.

```
Worksheets("Hoja1").Activate
MsgBox ActiveCell.Value
```

En este ejemplo se cambia el formato de fuente de la celda activa.

```
Worksheets("Hoja1").Activate
With ActiveCell.Font
    .Bold = True
    .Italic = True
End With
```

Propiedad Column

Devuelve el número de la primera columna del primer área del rango especificado. **Long** de sólo lectura.

Comentarios

Column A devuelve 1, column B devuelve 2 y así sucesivamente.

Para devolver el número de la última columna del rango, use la siguiente expresión:

```
myRange.Columns(myRange.Columns.Count).Column
```

Ejemplo

Este ejemplo establece en 4 puntos el ancho de las columnas salteadas de Hoja1.

```
For Each col In Worksheets("Hoja1").Columns
    If col.Column Mod 2 = 0 Then
        col.ColumnWidth = 4
    End If
Next col
```

Propiedad Columns

Propiedad Columns tal como se aplica al objeto Application.

Devuelve un objeto **Range** que representa todas las columnas de la hoja de cálculo activa. Si el documento activo no es una hoja de cálculo, esta propiedad fallará. Es de sólo lectura.

expresión.**Columns**

expresión: Requerida. Expresión que devuelve un objeto de la lista Aplicar a.

Propiedad Columns tal como se aplica al objeto **Range**.

Devuelve un objeto **Range** que representa las columnas del rango especificado. Es de sólo lectura.

expresión.**Columns**

expresión: Requerida. Expresión que devuelve un objeto de la lista Aplicar a.

Propiedad Columns tal como se aplica al objeto **WorkSheet**.

Devuelve un objeto **Range** que representa todas las columnas de la hoja de cálculo especificada. Es de sólo lectura.

expresión.**Columns**

expresión: Requerida. Expresión que devuelve un objeto de la lista Aplicar a.

Para obtener información sobre cómo devolver un solo elemento de un conjunto, consulte Devolver un objeto de un conjunto.

Comentarios

El uso de esta propiedad sin calificador de objeto equivale a usar `ActiveSheet.Columns`.

Si se aplica a un objeto **Range** que sea una selección de varias áreas, la propiedad sólo devolverá las columnas de la primera área del rango. Por ejemplo, si el objeto **Range** tiene dos áreas - A1:B2 y C3:D4 - `Selection.Columns.Count` devuelve 2, no 4. Si desea utilizar esta propiedad en un rango que puede contener una selección de varias áreas, compruebe `Areas.Count` para determinar si el rango contiene más de una área. En ese caso, ejecute un bucle sobre cada área del rango.

Ejemplo

Este ejemplo da formato de negrita a la fuente de la columna uno (columna A) de Hoja1.

```
Worksheets("Hoja1").Columns(1).Font.Bold = True
```

Este ejemplo establece como 0 (cero) el valor de todas las celdas de la columna uno del rango denominado "miRango".

```
Range("miRango").Columns(1).Value = 0
```

Este ejemplo muestra el número de columnas de la selección de Hoja1. Si se ha seleccionado más de una área, el ejemplo ejecutará un bucle en cada área.

```
Worksheets("Hoja1").Activate
areaCount = Selection.Areas.Count
If areaCount <= 1 Then
    MsgBox "La selección contiene " & _
        Selection.Columns.Count & " columnas."
Else
    For i = 1 To areaCount
        MsgBox "Area " & i & " de la selección contiene " & _
            Selection.Areas(i).Columns.Count & " columnas."
    Next i
End If
```

Propiedad EntireColumn

Devuelve un objeto **Range** que representa toda la columna (o columnas) que contiene el rango especificado. Es de sólo lectura.

Ejemplo

Este ejemplo establece el valor de la primera celda de la columna que contiene la celda activa. El ejemplo debe ejecutarse desde una hoja de cálculo.

```
ActiveCell.EntireColumn.Cells(1, 1).Value = 5
```

Propiedad Row

Devuelve el número de la primera fila de la primera área del rango. **Long** de sólo lectura.

Ejemplo

Este ejemplo establece en 4 puntos el alto de las filas alternativas de Hoja1.

```
For Each rw In Worksheets("Hoja1").Rows
    If rw.Row Mod 2 = 0 Then
        rw.RowHeight = 4
    End If
Next rw
```

Propiedad Rows

Para un objeto **Application**, devuelve un objeto **Range** que representa todas las filas de la hoja de cálculo activa. Si el documento activo no es una hoja de cálculo, esta propiedad fallará. Para un objeto **Range**, devuelve un objeto **Range** que representa las filas del rango especificado. Para un objeto **Worksheet**, devuelve un objeto **Range** que representa todas las filas de la hoja de cálculo especificada. Objeto **Range** de sólo lectura.

Comentarios

Para obtener información sobre cómo devolver un solo elemento de un conjunto, consulte Devolver un objeto de un conjunto.

El uso de esta propiedad sin calificador de objeto equivale a usar `ActiveSheet.Rows`.

Si se aplica a un objeto **Range** que es una selección múltiple, la propiedad sólo devolverá las filas de la primera área del rango. Por ejemplo, si el objeto **Range** tiene dos áreas -A1:B2 y C3:D4- `Selection.Rows.Count` devuelve 2, no 4. Si desea utilizar esta propiedad en un rango que puede contener una selección múltiple, compruebe `Areas.Count` para determinar si el rango es una selección múltiple. En ese caso, ejecute un bucle sobre cada área del rango, como se muestra en el tercer ejemplo.

Ejemplo

Este ejemplo elimina la fila tres de Hoja1.

```
Worksheets("Hoja1").Rows(3).Delete
```

Este ejemplo elimina las filas de la región actual de la hoja de cálculo uno en las que el valor de la celda uno de la fila es el mismo que el valor de la celda uno de la fila anterior.

```
For Each rw In Worksheets(1).Cells(1, 1).CurrentRegion.Rows
    this = rw.Cells(1, 1).Value
    If this = last Then rw.Delete
    last = this
Next
```

Este ejemplo muestra el número de filas de la selección de Hoja1. Si se ha seleccionado más de una área, el ejemplo ejecutará un bucle en cada área.

```
Worksheets("Hoja1").Activate
areaCount = Selection.Areas.Count
If areaCount <= 1 Then
    MsgBox "La selección contiene " & _
        Selection.Rows.Count & " filas."
Else
    i = 1
    For Each a In Selection.Areas
        MsgBox "Area " & i & " de la selección contiene " & _
            a.Rows.Count & " filas."
        i = i + 1
    Next a
End If
```

Propiedad EntireRow

Devuelve un objeto **Range** que representa toda la fila (o filas) que contiene el rango especificado. Es de sólo lectura.

Ejemplo

Este ejemplo establece el valor de la primera celda de la fila que contiene la celda activa. El ejemplo debe ejecutarse desde una hoja de cálculo.

```
ActiveCell.EntireRow.Cells(1, 1).Value = 5
```

Propiedad Formula

Veremos esta propiedad, tal como se aplica al objeto **Range**.

Devuelve o establece la fórmula del objeto en notación de estilo A1 y en el lenguaje de la macro. **Variant** de Lectura/Escritura.

expresión.**Formula**

expresión: Requerida. Expresión que devuelve un objeto **Range**.

Comentarios

Si la celda contiene una constante, esta propiedad la devolverá. Si la celda está vacía, la propiedad **Formula** devuelve una cadena vacía. Si la celda contiene una fórmula, la propiedad **Formula** devuelve la fórmula como una cadena con el mismo formato en que se presentaría en la barra de fórmulas (incluido el signo igual).

Si se define el valor o la fórmula de una celda como una fecha, Microsoft Excel comprobará si dicha celda ya tiene uno de los formatos numéricos de fecha u hora. De lo contrario, cambiará el formato numérico al formato numérico de fecha corto predeterminado.

Si el rango tiene una o dos dimensiones, puede definir la fórmula conforme a una matriz de Visual Basic de las mismas dimensiones. Asimismo, es posible introducir la fórmula en una matriz de Visual Basic.

Si define la fórmula para un rango de varias celdas, se rellenarán todas las celdas del rango con la fórmula.

Ejemplo

Este ejemplo establece la fórmula de la celda A1 de Hoja1.

```
Worksheets("Hoja1").Range("A1").Formula = "=$A$4+$A$10"
```

Propiedad Value

Veremos tal como se aplica al objeto **Range**.

Devuelve o establece el valor del rango especificado. **Variant** de Lectura/Escritura.

expresión.**Value**(***RangeValueDataType***)

expresión: Requerida. Expresión que devuelve un objeto **Range**.

RangeValueDataType: **Variant** opcional. Tipo de datos del valor del rango. Puede ser una constante **xlRangeValueDataType**.

xlRangeValueDataType puede ser una de estas constantes xlRangeValueDataType.

xlRangeValueDefault: *default*. Si el objeto **Range** está vacío, devuelve el valor **Empty** (utilice la función **IsEmpty** para comprobar si este es el caso). Si el objeto **Range** contiene más de una celda, devuelve una matriz de valores (utilice la función **IsArray** para comprobar ese caso).

xlRangeValueMSPersistXML: Devuelve la representación del conjunto de registros del objeto **Range** expecificado en formato XML.

xlRangeValueXMLSpreadsheet: Devuelve los valores, formatos, fórmulas y nombres del objeto **Range** especificado en formato de hoja de cálculo XML.

Comentarios

Al definir un rango de celdas con el contenido de un archivo de hoja de cálculo XML, sólo se utilizan los valores de la primera hoja del libro. No se puede definir un rango de celdas no contiguas en formato de hoja de cálculo XML.

Ejemplo

Este ejemplo establece el valor de la celda A1 de Hoja1 como 3.14159.

```
Worksheets("Hoja1").Range("A1").Value = 3.14159
```

Este ejemplo ejecuta un bucle sobre las celdas A1:D10 de Hoja1. Si una de las celdas tiene un valor menor que 0.001 el código sustituirá el valor por 0 (cero).

```
For Each c in Worksheets("Hoja1").Range("A1:D10")
    If c.Value < .001 Then
        c.Value = 0
    End If
Next c
```

Moviéndonos a través de Excel

En los siguientes ejemplos, usaremos el método "Select". Note que sin definir la selección, puede realizar cambios de propiedades o ejecutar métodos, lo que resulta más veloz.

```
ActiveCell.Offset(13, 14).Select
Selection.ClearContents
```

Puede ser también codificado:

```
ActiveCell.Offset(13, 14).ClearContents
```

Nota: Cuando usa las propiedades "Offset" y "Resize", como en los ejemplos de abajo, puede sustituir los números entre paréntesis por una expresión numérica.

```
Range("A1").CurrentRegion.Select
varNbRows=Selection.Rows.Count
Range("A1").Offset(varNbRows - 1,0).Select
```

Selecionar el libro desde donde el procedimiento se está ejecutando es:

```
ThisWorkbook.Activate
```

Seleccionar otro libro distinto (el libro debe estar abierto; no olvidar el ".xls"):

```
Windows("Totales.xls").Activate
```

Seleccionar una hoja:

```
Sheets("Balance").Activate.
```

Seleccionar una celda:

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

Seleccionar celdas contiguas:

```
Range("A1:G8").Select
```

Seleccionar celdas no contiguas:

```
Range("A1,B6,D9").Select
Range("A1,B6:B10,D9").Select
```

Moverse entre celdas:

```
ActiveCell.Offset(13, 14).Select
Selection.Offset(-3, -4).Select
Range("G8").Offset(-3, -4).Select
```

Nota : Puede crear una variable y usarla como desplazamiento, como se indica:

```
varFilaInicial = Range("A1").CurrentRegion.Rows.Count
ActiveCell.Offset([varFilaInicial], 0).Select
```

Para seleccionar una hoja entera:

```
Cells.Select
```

Para seleccionar filas o columnas:

```
Rows("1").Select
Columns("A").Select
```

ó

```
ActiveCell.EntireRow.Select
ActiveCell.EntireColumn.Select
```

Para seleccionar varias filas o columnas contiguas:

```
Columns("A:C").Select
Rows("1:5").Select
```

Para seleccionar varias filas o columnas no contiguas:

NOTA: Use el objeto "Range" y no “Columns” o “Rows”

```
Range("A:A, C:C, E:F").Select
Range("1:1,5:6,9:9").Select
```

Para seleccionar un rango desde la celda hasta la última celda no vacía en la columna:

```
Range("A1", Range("A1").End(xlDown)).Select
Range(ActiveCell, ActiveCell.End(xlDown)).Select
```

Para seleccionar un rango desde la celda hasta la primera celda no vacía en la columna:

```
Range("A32", Range("A32").End(xlUp)).Select
Range(ActiveCell, ActiveCell.End(xlUp)).Select
```

Para seleccionar un rango desde la celda hasta la última celda no vacía en la fila (por izquierda o derecha) (Notar el uso de `xltoRight` y de `xltoLeft`):

```
Range("A1", Range("A1").End(xltoRight)).Select
Range(ActiveCell, ActiveCell.End(xltoLeft)).Select
```

Para seleccionar un rango desde la celda hasta 10 celdas a la derecha:

```
Range("A2", Range("A2").Offset(0, 10)).Select
Range(ActiveCell, ActiveCell.Offset(0, 10)).Select
```

Para seleccionar un rango desde la celda hasta 10 celdas a la izquierda:

```
Range("A20", Range("A20").Offset(0, -10)).Select
Range(ActiveCell, ActiveCell.Offset(0, -10)).Select
```

Para seleccionar un rango desde la celda hasta 10 celdas hacia abajo:

```
Range("a2", Range("a2").Offset(10, 0)).Select
Range(ActiveCell, ActiveCell.Offset(10, 0)).Select
```

Para seleccionar la primera celda vacía en una columna:

```
Range("A1").End(xlDown).Offset(1, 0).Select
```

Para seleccionar la primera celda vacía en una fila:

```
Range("A1").End(xltoRight).Offset(0,1).Select
```

Para redefinir el tamaño del rango (de A1:B5 a A1:D10)

Nota: un rango no es expandido sino que se redefine desde la celda activa.

```
Selection.Resize(4, 10).Select
```

y no

```
Selection.Resize(2, 5).Select
```

Si queremos seleccionar celdas que llamamos Var_1, Var_2 y Var_3, podemos usar el siguiente bucle:

```
Sub asigna()
  For i = 1 To 3
    varNo1 = "Var_" & i
    Range([varNo1]).Select
....--- cualquier proceso con la celda seleccionada ---
  Next i
End Sub
```

GLOSARIO

evento

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

instrucción

Una unidad sintácticamente completa que expresa un tipo de acción, declaración o definición. Normalmente una instrucción tiene una sola línea aunque es posible utilizar dos puntos (:) para poner más de una instrucción en una línea. También se puede utilizar un carácter de continuación de línea (_) para continuar una sola línea lógica en una segunda línea física. Si desea realizar un comentario, colocar al principio de la línea la palabra REM o el signo apóstrofe (').

método

Un procedimiento que se aplica a un objeto.

módulo

Un conjunto de declaraciones y procedimientos.

objeto (object)

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

propiedad (propertie)

Un atributo con nombre de un objeto. Las propiedades definen características de objetos, como tamaño, color y ubicación en la pantalla, o comportamientos de objetos, como si está o no habilitado.

procedimiento (procedure)

Una secuencia con nombre de instrucciones que se ejecutan como una unidad. Por ejemplo, Function, Property y Sub son todos tipos de procedimientos. Un nombre de procedimiento siempre se define a nivel de módulo. Todo el código ejecutable debe estar contenido en un procedimiento. Los procedimientos no se pueden anidar dentro de otros procedimientos. Es similar al concepto con el que antes llevaba el nombre de macros