

Variables y Operadores

Convenciones de nomenclatura en Visual Basic

Cuando escribe código en Visual Basic, declara y asigna nombre a muchos elementos (procedimientos *Sub* y *Function*, variables, constantes, etc.). Los nombres de procedimientos, variables y constantes que declara en el código de Visual Basic deben seguir estas directrices:

- Deben comenzar por una letra.
- No pueden contener puntos o caracteres de declaración de tipos (caracteres especiales que especifican tipos de datos).
- No pueden superar los 255 caracteres. Los nombres controles, formularios y módulos no deben exceder los 40 caracteres.
- No pueden ser iguales que las palabras clave restringidas.

Una *palabra clave restringida* es una palabra que Visual Basic utiliza como parte de su lenguaje. Esto incluye a las instrucciones predefinidas (como *If* y *Loop*), funciones (como *Len* y *Abs*) y operadores (como *Or* y *Mod*).

Los formularios y controles pueden tener el mismo nombre que una palabra clave restringida. Por ejemplo, puede tener un control que se llame *Loop*. Sin embargo, en el código no puede hacer referencia a ese control de la forma habitual, ya que Visual Basic supone que se está refiriendo a la palabra clave *Loop*. Por ejemplo, este código provoca un error:

```
Loop.Visible = True
```

Para referirse a un formulario o un control que tenga el mismo nombre que una palabra clave restringida, debe calificarlo o ponerlo entre corchetes: []. Por ejemplo, este código no provoca un error:

```
MiForm.Loop.Visible = True (Calificado con el nombre del formulario.)
```

```
[Loop].Visible = True (Con corchetes también funciona.)
```

Puede usar corchetes de esta forma cuando haga referencia a formularios y controles, pero no cuando declare una variable o defina un procedimiento con el mismo nombre que una palabra clave restringida. También se pueden usar corchetes para que Visual Basic acepte nombres de otras bibliotecas de tipos que entren en conflicto con palabras clave restringidas.

Puesto que escribir corchetes puede resultar tedioso, evite la utilización de palabras clave restringidas en los nombres de formularios y controles. Sin embargo, puede usar esta técnica si una versión posterior de Visual Basic define una palabra clave nueva que entre en conflicto con un nombre de formulario o control existente y esté actualizando el código para que funcione con la versión nueva.

Variables

En Visual Basic puede utilizar variables para almacenar valores temporalmente durante la ejecución de una aplicación. Las variables tienen un *nombre* (la palabra que utiliza para referirse al valor que contiene la variable) y un *tipo de dato* (que determina la clase de datos que la variable puede almacenar).

Puede considerar una variable como un marcador de posición en memoria de un valor desconocido. Por ejemplo, suponga que está creando un programa para una frutería que haga un seguimiento del precio de las manzanas. No sabe el precio de una manzana o la cantidad que se ha vendido hasta que no se produce realmente la venta. Puede utilizar dos variables para almacenar los valores desconocidos (vamos a llamarlos *PrecioManzanas* y *ManzanasVendidas*). Cada vez que se ejecuta el programa, el usuario especifica los valores de las dos variables. Para calcular las ventas totales y mostrarlas en un cuadro de texto llamado *txtVentas*, el código debería parecerse al siguiente:

```
txtVentas.txt = PrecioManzanas * ManzanasVendidas
```

La expresión devuelve un total distinto cada vez, dependiendo de los valores que indique el usuario. Las variables le permiten realizar un cálculo sin tener que saber antes cuáles son los valores especificados.

En este ejemplo, el tipo de dato de *PrecioManzanas* es *Currency*; el tipo de dato de *ManzanasVendidas* es *Integer*. Las variables pueden representar otros muchos valores como valores de texto, fechas, diversos tipos numéricos e incluso objetos.

Almacenamiento y recuperación de datos

Utilice instrucciones de asignación para realizar cálculos y asignar el resultado a una variable:

```
ManzanasVendidas = 10 (Se pasa el valor 10 a la variable)
```

```
ManzanasVendidas = ManzanasVendidas + 1 (Se incrementa la variable)
```

Observe que el signo igual del ejemplo es un operador de asignación, no un operador de igualdad; el valor (10) se asigna a la variable (*ManzanasVendidas*).

Declaración de variables

Declarar una variable es decirle al programa algo de antemano. Se declara una variable mediante la instrucción *Dim*, proporcionando un nombre a la variable:

```
Dim nombre_variable [As tipo]
```

Las variables que se declaran en un procedimiento mediante la instrucción *Dim* sólo existen mientras se ejecuta el procedimiento. Cuando termina el procedimiento, desaparece el valor de la variable. Además, el valor de una variable de un procedimiento es *local* de dicho procedimiento; es decir, no puede tener acceso a una variable de un procedimiento desde otro procedimiento. Estas características le permiten utilizar los mismos nombres de variables en distintos procedimientos sin preocuparse por posibles conflictos o modificaciones accidentales.

La cláusula opcional *As tipo* de la instrucción *Dim* le permite definir el tipo de dato o de objeto de la variable que va a declarar. Los tipos de datos definen el tipo de información que almacena la variable. Algunos ejemplos de tipos de datos son *String*, *Integer* y *Currency*.

Hay otras formas de declarar variables:

- Declarar una variable en la sección *Declaraciones* de un módulo de formulario, estándar o de clase, en vez de en un procedimiento, hace que la variable esté disponible para todos los procedimientos del módulo.
- Declarar una variable mediante la palabra clave *Public* hace que esté accesible para toda la aplicación.
- Declarar una variable local mediante la palabra clave *Static* preserva su valor aunque termine el procedimiento.

Declaración Implícita

No tiene por qué declarar una variable antes de utilizarla. Por ejemplo, podría escribir una función donde no hiciera falta declarar *TempVal* antes de utilizarla:

```
Function SafeSqr(num)
    TempVal = Abs(num)
    SafeSqr = Sqr(TempVal)
End Function
```

Visual Basic crea automáticamente una variable con ese nombre, que puede utilizar como si la hubiera declarado explícitamente. Aunque es cómodo, puede provocar errores sutiles en el código si se equivoca de nombre de variable. Por ejemplo, suponga que ha escrito esta función:

```
Function SafeSqr(num)
    TempVal = Abs(num)
    SafeSqr = Sqr(TemVal)
End Function
```

A primera vista, parece igual. Pero como se ha escrito erróneamente la variable *TempVal* en la tercera línea, la función devolverá siempre cero. Cuando Visual Basic encuentra un nombre nuevo, no puede averiguar si realmente desea declarar una variable nueva o simplemente ha escrito de forma errónea una variable existente, por lo que crea una variable nueva con ese nombre.

Declaración Explícita

Para evitar problemas al equivocarse de nombre en las variables, puede estipular que Visual Basic le avise siempre que encuentre un nombre que no se haya declarado explícitamente como una variable.

Para declarar variables de forma explícita:

Incluya esta instrucción en la sección *Declaraciones* del módulo de clase, de formulario o estándar:

```
Option Explicit
```

o bien, en el menú *Herramientas*, elija *Opciones*, haga clic en la ficha *Editor* y active la opción *Declaración de variables requerida*. Esto inserta automáticamente la instrucción *Option Explicit* en los módulos nuevos, pero no en los ya creados, por lo que tendrá que agregar manualmente *Option Explicit* a los módulos existentes en el proyecto.

Si hubiera tenido efecto dicha instrucción en el módulo de formulario o estándar que contiene la función *SafeSqr*, Visual Basic habría reconocido *TempVal* y *TemVal* como variables no declaradas y habría generado errores para ambas. Debería, por tanto, declarar explícitamente *TempVal*:

```
Function SafeSqr(num)
    Dim TempVal
    TempVal = Abs(num)
    SafeSqr = Sqr(TemVal)
End Function
```

Ahora podrá comprender el problema inmediatamente porque Visual Basic habrá mostrado un mensaje de error para la variable *TemVal* que se ha escrito de forma incorrecta. Como la instrucción *Option Explicit* le ayuda a interceptar esta clase de errores, es conveniente utilizarla en todo el código.

La instrucción *Option Explicit* funciona sobre módulo a módulo; debe colocarse en la sección *Declaraciones* de todos los módulos de formulario, estándar o de clase en los que desee que Visual Basic obligue a declarar explícitamente las variables. Si activa *Declaración de variables requerida*, Visual Basic insertará *Option Explicit* en todos los módulos de formulario, estándares o de clase siguientes, pero no lo agregará en el código existente. Deberá agregar manualmente *Option Explicit* a los módulos ya existentes en el proyecto.

Otra forma de declarar variables es utilizando los caracteres de declaración de tipo. Por ejemplo:

I%	Variable entera
R#	Variable real de precisión doble
Nombre\$	Cadena de caracteres de longitud variable
f@	Variable fraccionaria

Alcance de las variables

El alcance de una variable define qué partes del código son conscientes de su existencia. Cuando declara una variable en un procedimiento, sólo el código de dicho procedimiento puede tener acceso o modificar el valor de la variable; tiene un alcance que es local al procedimiento. A veces, sin embargo, se necesita utilizar una variable con un alcance más general, como aquella cuyo valor está disponible para todos los procedimientos del mismo módulo o incluso para todos los procedimientos de toda la aplicación. Visual Basic le permite especificar el alcance de una variable cuando la declara.

Establecimiento del alcance de las variables

Dependiendo de cómo se declara, una variable tiene como alcance un procedimiento (local) o un módulo.

Alcance	Privado	Público
<i>Nivel de procedimiento</i>	Las variables son privadas del procedimiento en el que aparecen.	No es aplicable. No puede declarar variables públicas dentro de un procedimiento.
<i>Nivel de módulo</i>	Las variables son privadas del módulo en el que aparecen.	Las variables están disponibles para todos los módulos.

Variables utilizadas en un procedimiento

Las variables a nivel de procedimiento sólo se reconocen en el procedimiento en el que se han declarado. Se las conoce también como variables locales. Se declaran mediante las palabras clave *Dim* o *Static*. Por ejemplo:

```
Dim intTemp As Integer
```

o bien

```
Static intPermanent As Integer
```

Los valores de variables locales declaradas con *Static* existen mientras se ejecuta la aplicación, mientras que las variables declaradas con *Dim* sólo existen mientras se ejecuta el procedimiento.

Las variables locales resultan una elección apropiada para cálculos temporales. Por ejemplo, puede crear una docena de procedimientos distintos que contengan una variable llamada *intTemp*. Como cada *intTemp* se ha declarado como una variable local, cada procedimiento sólo reconoce su propia versión de *intTemp*. Cualquier procedimiento puede alterar el valor de su *intTemp* local sin que ello afecte a las variables *intTemp* de los demás procedimientos.

Variables utilizadas en un módulo

De forma predeterminada, una variable a nivel de módulo está disponible para todos los procedimientos del módulo, pero no para el código de otros módulos. Cree variables a nivel de módulo declarándolas con la palabra clave *Private* en la sección *Declaraciones* al principio del módulo. Por ejemplo:

```
Private intTemp As Integer
```

A nivel de módulo, no hay diferencia entre *Private* y *Dim*, pero es preferible *Private* porque contrasta con *Public* y hace que el código sea más fácil de comprender.

Variables utilizadas por todos los módulos

Para hacer que una variable a nivel de módulo esté disponible para otros módulos, utilice la palabra clave *Public* para declarar la variable. Los valores de las variables públicas están disponibles para todos los procedimientos de la aplicación. Al igual que todas las variables a nivel de módulo, las variables públicas se declaran en la sección *Declaraciones* al principio del módulo. Por ejemplo:

```
Public intTemp As Integer
```

No puede declarar variables públicas en un procedimiento, sólo en la sección *Declaraciones* de un módulo.

Variables con el mismo nombre

Una variable local y otra a nivel del módulo pueden tener el mismo nombre, pero no son la misma variable. La regla para estos casos es que el procedimiento siempre utiliza la variable de nivel más cercano (local, módulo y global; en este orden). En otro caso, debe calificar la variable. Por ejemplo:

```
Public Temp As Integer      'declaración en Form1
Public Sub Test()          'procedimiento en Modulo1
  Dim Temp As Integer
  Temp = 10                'referencia a la variable local
  MsgBox Form1.Temp        'referencia a la variable global
End Sub
```

Si varias variables públicas comparten el mismo nombre en diferentes módulos, para diferenciarlas en el momento de referenciarlas es necesario especificar pertenencia. Por ejemplo, si hay una variable entera X declarada tanto en el modulo *Form1* como en el *Modulo1*, debemos referirnos a ella así:

```
MsgBox Form1.X            'visualiza el valor de la X de Form1
MsgBox Modulo1.X          'visualiza el valor de la X de Modulo1
```

Es aconsejable en programación que los nombres de las variables sean diferentes entre sí, así como los nombres de las propiedades y de los módulos.

Tipos de datos de las variables

Las variables son marcadores de posición que se utilizan para almacenar valores; tienen nombre y un tipo de dato. El tipo de dato de la variable determina cómo se almacenan los bits que representan esos valores en la memoria del equipo. Cuando declare una variable, también puede proporcionarle un tipo de dato. Todas las variables tienen un tipo de dato que determina la clase de datos que pueden almacenar.

De forma predeterminada, si no proporciona un tipo de dato, la variable toma el tipo de dato *Variant*. El tipo de dato *Variant* puede representar diferentes tipos de datos en distintas situaciones. No tiene que convertir estos tipos de datos cuando los asigne a una variable *Variant*: Visual Basic realiza automáticamente cualquier conversión necesaria.

Sin embargo, si sabe que una variable almacenará siempre un tipo de dato determinado, Visual Basic tratará de forma más eficiente los datos si declara la variable con ese tipo. Por ejemplo, se representa mejor una variable que va a almacenar nombres de personas como el tipo de dato *String*, ya que un nombre está siempre compuesto de caracteres.

Los tipos de datos se aplican a otras cosas además de a las variables. Cuando asigna un valor a una propiedad, dicho valor tiene un tipo de dato; los argumentos de las funciones tienen también tipos de datos. De hecho, todo lo relacionado con datos en Visual Basic tiene un tipo de dato.

Declarar variables con tipos de datos

Antes de utilizar una variable que no sea *Variant* debe utilizar las instrucciones *Private*, *Public*, *Dim* o *Static* para declararla *As tipo*. Por ejemplo, la siguiente instrucción declara un tipo *Integer*, *Double*, *String* y *Currency*, respectivamente:

```
Private I As Integer
Dim Cantidad As Double
Static SuNombre As String
Public PagadoPorJuan As Currency
```

La instrucción de declaración puede combinar varias declaraciones, como en las instrucciones siguientes:

```
Private I As Integer, Amt As Double
Private SuNombre As String, PagadoPorJuan As Currency
Private Prueba, Cantidad, J As Integer
```

Si no proporciona un tipo de dato, se asigna a la variable el tipo predeterminado. En el ejemplo anterior, las variables *Prueba* y *Cantidad* tienen un tipo de dato *Variant*. Esto puede sorprenderle si su experiencia con otros lenguajes de programación le lleva a esperar que todas las variables contenidas en la misma instrucción de declaración tengan el mismo tipo que ha especificado (en este caso, *Integer*).

Distintos tipos de variables utilizados en Visual Basic

Tipo	Descripción	Car-tipo	Rango
<i>Integer</i>	Entero - 2 Bytes	%	-32.768 a 32.768
<i>Long</i>	Entero Largo - 4 Bytes	&	-2.147.483.648 a 2.147.483.647
<i>Single</i>	Coma flotante. Precisión simple - 4 Bytes	!	-3,40E+38 a 3,40E+38
<i>Double</i>	Coma flotante. Precisión Doble - 8 Bytes	#	-1,79D+308 a 1,79D+308
<i>Currency</i>	Número con punto decimal fijo - 8 Bytes	@	+/- 922.337.203.685.477,5807
<i>String</i>	Cadena de caracteres de longitud fija - 1 Byte por carácter	ninguno	Hasta 64K aproximadamente
<i>String</i>	Cadena de caracteres de longitud variable - 10 Bytes + 1 Byte por carácter	\$	Hasta 231 caracteres aproximadamente
<i>Byte</i>	Carácter - 1 Byte	ninguno	0 a 255
<i>Boolean</i>	Booleano - 2 Bytes	ninguno	True o False
<i>Date</i>	Fecha/Hora - 8 Bytes	ninguno	1/Enero/100 a 31/Diciembre/9.999
<i>Object</i>	Referencia a un objeto - 4 Bytes	ninguno	Cualquier referencia a tipo de Objeto
<i>Variant</i>	Con números - 16 Bytes	ninguno	Cualquier valor numérico hasta el intervalo de tipo Double
<i>Variant</i>	Con caracteres - 22 Bytes + 1 por carácter	ninguno	El mismo intervalo que para un tipo String de longitud variable
<i>Decimal</i>	Números con 0 a 28 decimales - 14 Bytes (No se puede declarar una variable de este tipo. Sólo se puede utilizar con Variant)	ninguno	Valor mayor con 0 decs: +/- 79.228.162.514.264.337.593.543.950.335 Valor mayor con 28 decs: +/- 7,9228162514264337593543950335

Tipos de datos numéricos

Visual Basic proporciona varios tipos de datos numéricos: *Integer*, *Long* (entero largo), *Single* (signo flotante de simple precisión), *Double* (signo flotante de doble precisión) y *Currency*. Utilizar un tipo de dato numérico emplea normalmente menos espacio de almacenamiento que un tipo *Variant*.

Si sabe que una variable siempre va a almacenar números enteros (como 12) en vez de números fraccionarios (como 3,57), declárela como un tipo *Integer* o *Long*. Las operaciones con enteros son más rápidas y estos tipos consumen menos memoria que otros tipos de datos.

Si la variable contiene una fracción, declárela como variable *Single*, *Double* o *Currency*. El tipo de dato *Currency* acepta hasta cuatro dígitos a la derecha del separador decimal y hasta quince dígitos a la izquierda; es un tipo de dato de signo fijo adecuado para cálculos

monetarios. Los números de signo flotante (*Single* y *Double*) tienen más intervalo que *Currency*, pero pueden estar sujetos a pequeños errores de redondeo.

El tipo de dato Byte

Todos los operadores que funcionan con enteros funcionan con el tipo de dato *Byte* excepto el de resta. Puesto que *Byte* es un tipo sin signo con el intervalo 0 - 255, no puede representar un valor negativo.

Es posible asignar todas las variables numéricas entre sí y a variables del tipo *Variant*. Visual Basic redondea en vez de truncar la parte fraccionaria de un número de signo flotante antes de asignarlo a un entero.

El tipo de dato String

Si tiene una variable que siempre contendrá una cadena y nunca un valor numérico, puede declararla del tipo *String*:

```
Private S As String
```

Así podrá asignar cadenas a esta variable y manipularla mediante funciones de cadena:

```
S = "Base de datos"  
S = Left(S, 4)
```

De forma predeterminada, una variable o argumento de cadena es una *cadena de longitud variable*; la cadena crece o disminuye según le asigne nuevos datos. También puede declarar cadenas de longitud fija. Especifique una *cadena de longitud fija* con esta sintaxis:

String * tamaño

Por ejemplo, para declarar una cadena que tiene siempre 50 caracteres de longitud, utilice un código como este:

```
Dim NombreEmp As String * 50
```

Si asigna una cadena con menos de 50 caracteres, *NombreEmp* se rellenará con espacios en blanco hasta el total de 50 caracteres. Si asigna una cadena demasiado larga a una cadena de longitud fija, Visual Basic simplemente truncará los caracteres.

Puesto que las cadenas de longitud fija se rellenan con espacios al final, verá que las funciones *Trim* y *RTrim*, que quitan los espacios en blanco, resultan muy útiles cuando se trabaja con ellas.

Tipo enumerado

La declaración de un tipo enumerado es simplemente una lista de valores que pueden ser tomados por una variable de ese tipo. Los valores de un tipo enumerado se presentarán con identificadores, que serán las constantes del nuevo tipo. Por ejemplo:

```
Public Enum DíasSemana  
    lunes  
    martes  
    miércoles  
    jueves  
    viernes  
    sábado  
    domingo  
End Enum  
  
Dim hoy As DíasSemana
```

Este ejemplo declara la variable *hoy* del tipo enumerado *DíasSemana*. Esta variable puede tomar cualquier valor de los especificados, lunes a domingo. Las constantes son de tipo *Long* y sus valores por omisión son 0, 1, 2, etc. Según esto, el valor de lunes es 0, el valor de martes es 1, y así sucesivamente. Por lo tanto, una sentencia como

hoy = domingo

será equivalente a

hoy = 6

A cualquier identificador de la lista se le puede asignar un valor inicial de tipo *Long* por medio de una expresión constante. Los identificadores sucesivos tomarán valores correlativos a partir de éste. Por ejemplo:

```
Public Enum DíasLaborales
    sábado
    domingo = 0
    lunes
    martes
    miércoles
    jueves
    viernes
    NoVálido = -1
End Enum
```

Este ejemplo declara un tipo enumerado llamado *DíasLaborales*. Los valores asociados con cada una de las constantes son: *sábado* = 0, *domingo* = 0, *lunes* = 1, *martes* = 2, *miércoles* = 3, *jueves* = 4, *viernes* = 5, *NoVálido* = -1.

A los tipos enumerados se les aplican las siguientes reglas:

- Un tipo enumerado puede declararse *Private* o *Public*.
- De forma predeterminada, la primera constante de una enumeración se inicia a 0 y las siguientes constantes reciben un valor superior en una unidad al de constante anterior.
- Dos o más constantes pueden tener un mismo valor.
- Una constante puede aparecer en más de un tipo.
- Para evitar referencias ambiguas cuando haga referencia a una constante individual, califique el nombre de la constante mediante su enumeración. Por ejemplo:

```
hoy = DíasSemana.domingo
```

- No es posible leer o escribir directamente un valor de un tipo enumerado; esto es, cuando se escribe una variable de un tipo enumerado lo que se escribe es el valor asociado, y cuando se lee, hay que introducir el valor asociado. En cambio en asignaciones o en comparaciones sí se pueden utilizar los identificadores del tipo enumerado.

El tipo de dato Boolean

Si tiene una variable que siempre contendrá solamente información del tipo verdadero y falso, sí y no o activado o desactivado, puede declararla del tipo *Boolean*. El valor predeterminado de *Boolean* es *False*. En el siguiente ejemplo, *blnEjecutando* es una variable *Boolean* que almacena un simple sí o no.

```
Dim blnEjecutando As Boolean
' Comprueba si la cinta está en marcha.
If Recorder.Direccion = 1 Then
    blnEjecutando = True
End if
```

El tipo de dato Date

Los valores de fecha y hora pueden almacenarse en el tipo de dato específico *Date* o en variables *VARIANT*. En ambos tipos se aplican las mismas características generales a las fechas.

El tipo de dato Variant

Una variable *Variant* es capaz de almacenar todos los tipos de datos definidos en el sistema. No tiene que convertir entre esos tipos de datos si los asigna a una variable *Variant*; Visual Basic realiza automáticamente cualquier conversión necesaria. Por ejemplo:

```
Dim AlgunValor
AlgunValor = "17"
AlgunValor = AlgunValor - 15
AlgunValor = "U" & AlgunValor
```

En la primera instrucción se define la variable *AlgunValor*, la cual de forma predeterminada es un tipo *Variant*. En la segunda *AlgunValor* contiene "17" (cadena de dos caracteres), en la tercera *AlgunValor* ahora contiene el valor numérico 2 y en la última *AlgunValor* ahora contiene "U2" (una cadena de dos caracteres).

Si bien puede realizar operaciones con variables *Variant* sin saber exactamente el tipo de dato que contienen, hay algunas trampas que debe evitar.

- Si realiza operaciones aritméticas o funciones sobre un *Variant*, el *Variant* debe contener un número.
- Si está concatenando cadenas, utilice el operador **&** en vez del operador **+**.

Cuando utilice el operador **&**, tenga cuidado de dejar un espacio entre el nombre de la variable y el operador, de lo contrario Visual Basic interpretaría que se trata del carácter de declaración de tipo *Long*.

Si intenta ejecutar una operación o función matemática sobre una variable *Variant* que no tenga un valor que pueda ser interpretado como un número, ocurrirá error. Para evitar esto, utilice la función *IsNumeric* para interrogar si dicha variable contiene un valor que pueda ser utilizado como un número. Por ejemplo:

```
Dim Dato           'Variant por omisión
Dato = "123"       'cadena de caracteres
If IsNumeric(Dato) Then
  Dato = Dato + 1111 'resultado Double 1234
End If
Dato = Dato & "5"   'cadena de caracteres "12345"
```

Cuando asignamos un valor numérico a una variable *Variant* utiliza la representación más compacta para registrar el valor.

```
Dim Dato           'variable de tipo Variant
Dato = 123         'contiene un valor Integer
Dato = Dato + 1111 'resultado Integer 1234
```

Una variable *Variant* no es una variable sin tipo; más bien, es una variable que puede cambiar su tipo libremente (una variable genérica). Cuando usted quiera conocer el tipo de dato que almacena una variable *Variant*, utilice la función *VarType*. Cada tipo de dato tiene asociado en Visual Basic un número entero que lo diferencia de los demás (por ejemplo, *Integer* tiene asociado el 2, *Single* el 4, etc.). Ese número entero es devuelto por la función *VarType*. En el ejemplo siguiente puede ver el valor asociado con cada tipo:

```
Sub TipoDato(Dato As Variant)
  Dim Tipo As Integer
  Tipo = VarType(Dato) 'almacena en Tipo un entero
  Select Case Tipo
    Case 0 'si Tipo es 0 la variable Dato no está iniciada, Empty
      MsgBox "Vacío"
    Case 1 'si Tipo es 1, Dato no tiene datos válidos (Null)
      MsgBox "Null"
    Case 2 'si Tipo es 2, Dato se corresponde con un entero
      MsgBox "Integer: " & dato
```

```

Case 3
    MsgBox "Long: " & Dato
Case 4
    MsgBox "Single: " & Dato
Case 5
    MsgBox "Double: " & Dato
Case 6
    MsgBox "Currency: " & Dato
Case 7
    MsgBox "Fecha/Hora: " & Dato
Case 8
    MsgBox "String: " & Dato
Case 11
    MsgBox "Boolean : " & Dato
Case 17
    MsgBox "Byte: " & Dato
End Select
End Sub

```

La sentencia *Case* y la función y el procedimiento *MsgBox* se verán más adelante.

En ocasiones, cuando operamos con variables *Variant* obtenemos resultados tipo *Double*, cuando en realidad esperamos otro tipo de resultado. Por ejemplo:

```
Dato = Dato + 1111 'Para Dato = "123", valor Double 1234
```

En este ejemplo quizá usted esperaba obtener un valor *Integer*. Para que así sea, utilice la función *CInt* para convertir la variable *Variant* a un valor entero.

```
Dato = CInt(Dato) + 1111 'valor entero
```

Las variables *Variant* también pueden contener valores de tipo *Date* y, con ellas, puede ejecutar operaciones aritméticas y comparaciones. Por ejemplo:

```

Dim Dato                                'Variant por omisión
Dato = Now                              'contenido: dd/mm/aa
Dato = Dato - 1                          'menos 1 día
If Dato > #24/10/99 3:00:00# Then
    Dato = Dato - 1 / 24 'menos 1 hora
End If
Dato = Dato - 10 / 24 / 60                'menos 10 minutos

```

Visual Basic acepta para una variable *Variant*, datos en diversos formatos de fecha/hora, incluidos entre #. Por ejemplo:

```

Dim Dato                                'Variant por omisión
Dato = #18-4-01 15:30#
Dato = #18 Apr, 2001 3:30pm#
Dato = #18-Apr-01#
Dato = #18 Apr1 2001#

```

Para verificar si un determinado valor puede considerarse como uno del tipo *Date* (fecha/hora), utilice la función *IsDate*. Para convertir dicho valor en uno del tipo *Date*, utilice la función *CDate*. Por ejemplo, suponga que el contenido una caja de texto *Text1* es 18 4 01. Las siguientes sentencias

```

If IsDate(Text1.Text) Then
    Dato = CDate(Text1.Text)
End If

```

almacenan en la variable *Dato* el valor 18/04/01. Una alternativa a la función *CDate* es *DateValue*.

Además de poder actuar como otros tipos de datos estándar, los *Variant* también pueden contener tres valores especiales: *Empty*, *Null* y *Error*.

El valor Empty

A veces necesitará saber si se ha asignado un valor a una variable existente. Una variable *Variant* tiene el valor *Empty* antes de asignarle un valor. El valor *Empty* es un valor especial distinto de 0, una cadena de longitud cero ("") o el valor *Null*. Puede probar el valor *Empty* con la función *IsEmpty*:

```
If IsEmpty(Z) Then Z = 0
```

Cuando un *Variant* contiene el valor *Empty*, puede utilizarlo en expresiones; se trata como un 0 o una cadena de longitud cero, dependiendo de la expresión.

El valor *Empty* desaparece tan pronto como se asigna cualquier valor (incluyendo 0, una cadena de longitud cero o *Null*) a una variable *Variant*. Puede establecer una variable *Variant* de nuevo como *Empty* si asigna la palabra clave *Empty* al *Variant*.

El valor Null

El tipo de dato *Variant* puede contener otro valor especial: *Null*. *Null* se utiliza comúnmente en aplicaciones de bases de datos para indicar datos desconocidos o que faltan. Debido a la forma en que se utiliza en las bases de datos, *Null* tiene algunas características únicas:

- Las expresiones que utilizan *Null* dan como resultado siempre un *Null*. Así, se dice que *Null* se "propaga" a través de expresiones; si cualquier parte de la expresión da como resultado un *Null*, la expresión entera tiene el valor *Null*.
- Al pasar un *Null*, un *Variant* que contenga un *Null* o una expresión que dé como resultado un *Null* como argumento de la mayoría de las funciones hace que la función devuelva un *Null*.
- Los valores *Null* se propagan a través de funciones intrínsecas que devuelven tipos de datos *Variant*.

También puede asignar un *Null* mediante la palabra clave *Null*:

```
Z = Null
```

Puede utilizar la función *IsNull* para probar si una variable *Variant* contiene un *Null*:

```
If IsNull(X) And IsNull(Y) Then
    Z = Null
Else
    Z = 0
End If
```

Si asigna *Null* a una variable de un tipo que no sea *Variant*, se producirá un error interceptable. Asignar *Null* a una variable *Variant* no provoca ningún error y el *Null* se propagará a través de expresiones que contengan variables *Variant* (*Null* no se propaga a través de determinadas funciones). Puede devolver *Null* desde cualquier procedimiento *Function* con un valor de devolución de tipo *Variant*.

Null no se asigna a las variables a menos que se haga explícitamente, por lo que si no utiliza *Null* en su aplicación, no tendrá que escribir código que compruebe su existencia y lo trate.

Conversiones entre datos numéricos

Cuando una variable numérica de un tipo se asigna a otra variable numérica de un tipo diferente, Visual Basic convierte el dato asignado al tipo de la variable destino de los datos. Las conversiones se hacen de acuerdo con las reglas siguientes:

```
Dim ent1 As Integer, ent2 As Integer
Dim fracsp As Singler, fracdp As Double
```

1. Si se asigna un valor numérico de una precisión a una variable numérica de precisión diferente, el número será almacenado con la precisión declarada en el nombre de la variable destino. Por ejemplo:

```
ent1 = 40.17
```

```
ent2 = 37.83
Print ent1, ent2 'resultado: 40 38
```

2. Si se asigna un valor numérico de una precisión más alta a una variable de precisión más baja, dicho valor se redondea, no se trunca. Por ejemplo:

```
fracsp = 22.3445577
ent1 = 17.5
Print fracsp, ent1 'resultado: 22.34456 18
ent1 = 17.4
ent2 = 17.6
Print ent1, ent2 'resultado: 17 18
fracsp = 22.37
Print fracsp 'resultado: 22.37
```

Tenga en cuenta que un valor de tipo Single no tiene más de 7 cifras significativas (cifras enteras más cifras decimales). Por lo tanto, el redondeo ocurre cuando tratamos de asignar más cifras de las aceptadas por la variable. Un razonamiento similar se puede seguir para el resto de los tipos.

3. Si se cambia un número de precisión más baja a una precisión más alta, el número de precisión más alta resultante no podrá ser más exacto que el número de precisión más baja. Por ejemplo:

```
fracsp = 17.55
fracdp = fracsp
Print fracsp, fracdp 'resultado: 17.55 17.5499992370605
```

En este ejemplo se ha asignado un valor de precisión simple a una variable de precisión doble. Quizás usted esperaba como resultado el mismo valor asignado. Para comprender este otro resultado, piense en lo dicho anteriormente respecto a que al convertir un valor en base 10 de tipo *Single* o *Double* al mismo valor en base 2 y viceversa, ocurre un error por no poderse representar exactamente el valor decimal en binario.

4. Durante la evaluación de una expresión, todos los operandos en una operación aritmética o de relación se cambian al mismo grado de precisión, que es el del operando más preciso. Por ejemplo:

```
ent1 = 8
fracdp = 3
Print ent1 / fracdp 'resultado: 2.66666666666667
```

Constantes simbólicas

A menudo utilizamos valores constantes una y otra vez en el código, o bien el código depende de ciertos números difíciles de recordar. En estos casos, la mejor solución es definir estos valores como *constantes simbólicas* y utilizar en lo sucesivo los nombres que identifican estas constantes.

Para definir una constante simbólica, utilice la siguiente sintaxis:

```
[Public/Private] Const constante [As tipo] = expresión
```

Si no declara explícitamente el tipo de constante (utilizando *As tipo*), se asigna a la constante el tipo de datos más apropiado a su valor.

Para nombrar una *constante*, se utilizan las mismas reglas que se aplican para nombrar variables. La *expresión* puede ser numérica, alfanumérica o de tipo fecha y hora. Por ejemplo:

```
Public Const MAX_ELEMS = 99
Public Const VERSION = "ver. 4.05.0A"
Const PI = 3.1415926, DOS_PI = 2 * PI
Const FECHA_POR_DEFECTO = #1/01/01#
```

Es aconsejable definir todas las constantes globales en un único módulo. El ámbito de una constante se define por las mismas reglas que las variables.

Utilizando el examinador de objetos puede insertar en su código muchas de las constantes intrínsecas o definidas por el sistema.

Operadores

La tabla que se muestra a continuación presenta el conjunto de operadores que soporta Visual Basic ordenados de mayor a menor prioridad respecto a cómo se evalúan cuando varios de ellos intervienen en una misma expresión. Las operaciones que aparecen sobre una misma línea tienen igual prioridad. Las operaciones que se incluyan entre paréntesis se evaluarán antes, ejecutándose primero los paréntesis más internos.

Operador	Tipo	Operación
^	Aritmético	Exponenciación
-		Cambio de signo
*, /		Multiplicación y división
\		División entera
Mod		Resta de una división
+, -		Suma y resta
&	Concatenación	Concatenar o enlazar
=, <>, <, >, <=, >=	Relacional	Igual, distinto, menor, mayor ...
Like	Otros	Comparar dos expresiones de caracteres
Is		Comparar dos referencias a objetos
Not	Lógico	Negación
And		And
Or		Or inclusiva
Xor		Or exclusiva
Eqv		Equivalencia (opuesto a Xor)
Imp		Implicación (falso si primer operando verdadero y segundo operando falso)

Si al evaluar una expresión sucede que alguno de los operandos tiene un valor *Null*, el resultado es *Null*.

Cuando en una expresión aritmética intervienen operandos de diferentes tipos el resultado se expresa, generalmente, en la precisión del operando que la tiene más alta. El orden, de menor a mayor, según la precisión, es *Integer*, *Long*, *Single*, *Double* y *Currency*.

Los *operadores relacionales*, también conocidos como *operadores de comparación*, comparan dos expresiones dando un resultado *True* (verdadero), *False* (falso) o *Null* (no válido).

El operador *&* realiza la concatenación de dos operandos. Para el caso particular de que ambos operandos sean cadenas de caracteres, puede utilizarse también el operador *+*. No obstante; para evitar ambigüedades utilice *&*. El resultado es de tipo *String* si ambas expresiones son de tipo *String*; en otro caso, el resultado es, de tipo *Variant*.

Los operadores lógicos podemos utilizarlos de dos formas: para obtener un resultado de tipo *Boolean* (*True* o *False*), una vez evaluadas dos expresiones a *True* o a *False*, o para realizar una operación lógica bit a bit entre dos expresiones numéricas, colocando el resultado en la variable que se especifique.

Cuando otros tipos de datos numéricos se convierten a *Boolean*, 0 pasa a ser *False*, mientras que todos los demás valores pasan a ser *True*. Cuando los valores *Boolean* se convierten en otros tipos, *False* pasa a ser 0, mientras que *True* se transforma en -1.

Las tablas de verdad correspondientes a estos operadores son las siguientes:

A	B	Not A	A And B	A or B	A Xor B	A Eqv B	A Imp B
True	True	False	True	True	False	True	True
True	False	False	False	True	True	False	False
False	True	True	False	True	True	False	True
False	False	True	False	False	False	True	True

El operador *Not*, no lógico, se utiliza para negar la afirmación de un argumento. Por ejemplo, para expresar "si a no es mayor que b, entonces escribir a es menor o igual que b"; escribiremos:

```
If Not a > b Then Print a ; " es menor o igual que "; b
```

Si queremos expresar "mientras no existe, hacer...", escribiremos

```
Dim existe As Boolean
existe = False
While Not existe
...
Wend
```

En este ejemplo, *Not existe* es *True*, ya que *Not False* es *True* (verdadero).

El operador *And*, Y lógico, recibe también a nivel de bits el nombre de producto lógico. Por ejemplo, para expresar "si a es mayor que b y b es mayor que c entonces escribir a", escribiremos:

```
If a > b And b > c Then Print a 'si el resultado es True se escribe a
```

El operador *Or*, O lógico, recibe también a nivel de bits e1 nombre de suma lógica u *Or* inclusivo. Por ejemplo, para expresar "si a es igual a 0 o b es igual a 0, entonces finalizar", escribiremos:

```
If a = 0 Or b = 0 Then End 'si el resultado es True se finaliza
```

El operador *Xor*, Or exclusivo, recibe también a nivel bits el nombre de suma binaria. Por ejemplo:

```
a = 10: b = 8: c = 6           'inicia variables
resu = a > b Xor b > c         'devuelve False
resu = b > a Xor b > c         'devuelve True
resu = a Xor b                 'devuelve 2 (comparación bit a bit)
```

Como vemos, el operador *Xor* ejecuta también una comparación bit a bit para identificar bits de dos expresiones numéricas y establece el bit correspondiente en el resultado según la siguiente tabla de decisión lógica:

Si bit en operando1 es	Si bit en operando2 es	El resultado es
0	0	0
0	1	1
1	0	1
1	1	0

La expresión *a Xor b* da lugar a 2, puesto que si 10 es 1010 en binario y 8 es 1000 en binario, *10 Xor 8* es 0010 en binario, que en decimal es 2.

Podemos hacer un razonamiento similar para el resto de los operadores.

El operador *Like* se utiliza para comparar dos cadenas de caracteres. La sintaxis para este operador es la siguiente:

```
[resultado =] expresión Like patrón
```

El resultado será *True* si la expresión coincide con alguna de las definidas por el patrón, *False* si no hay coincidencia y *Null* si la expresión y/o el patrón son *Null*. Por omisión, en las comparaciones, se diferencian mayúsculas de minúsculas; esta característica puede ser alterada por la sentencia *Option Compare*. En el patrón se pueden incluir los caracteres comodín siguientes:

Car. Comodín	Descripción
?	Un solo carácter
*	Cero o más caracteres
#	Un solo dígito (0-9)
[lista_caracteres]	Un solo carácter de los pertenecientes a la lista
[!lista_caracteres]	Un solo carácter de los no pertenecientes a la lista

Por ejemplo:

Con este patrón	Esta expresión retorna True	Esta expresión retorna False
a?a	"aaa", "a5a", "aBa", "aba"	"aBbca"
a*a	"aa", "aBa", "aBBba"	"aBbc"
a#a	"a0a", "a2a", "a9a"	"aaa", "aba", "a29a"
a[*]a	"a*a"	"aaa", "a0a"
[a-z]	"f", "p", "j"	"3", "A", "."
[!0-9]	"A", "a", ".", "-"	"0", "4", "7"
a[!i-n]#	"ab1", "az0", "a99"	"aic", "ak0", "Ab1", "az90"

En el siguiente ejemplo el resultado de la comparación será *True* cuando la cadena de caracteres *Cadena* coincida con una cadena de tres caracteres, en la que el primero sea una 'a', el segundo cualquier carácter que no sea 'i', 'j', 'k', 'l', 'm' o 'n', y el tercero un dígito 0 a 9.

```
Dim Cadena As String
...
If . Cadena Like "a[!i-n]" Then
    'acciones a ejecutar si el resultado es True
End If
```

El operador *Is* se utiliza para comprobar si una variable definida se refiere a un objeto de los existentes, o si dos variables definidas se refieren al mismo objeto. Por ejemplo:

```
Dim A As Form    'referencia a un formulario
Dim B As Control 'referencia a un control
Set A = Form1    'hacer que A se refiera al mismo
                  'formulario identificado por Form1
Set B = Text1    'hacer que B se refiera a la misma
                  'caja de texto identificada por Text1
If A Is Form1 And B Is Text1 Then
    'acciones a ejecutar si el resultado es True
Else
    'acciones a ejecutar si el resultado es False
End If
```

Form y *Control* son clases de objetos predefinidas en Visual Basic. En cambio, *Form1* y *Text1* son objetos definidos por el usuario al diseñar la aplicación (un formulario y una caja de texto).

Sentencias

Una sentencia es una línea de texto que indica una o más operaciones a realizar. Una línea puede tener varias sentencias, separadas unas de otras por dos puntos:

```
sueldo = horasTrabajadas * pagoPorHora: sueldoReal = sueldo - deducciones
```

Una sentencia Visual Basic puede escribirse en varias líneas físicas utilizando el *carácter de continuación de línea* (un espacio en blanco seguido del carácter de subrayado). Por ejemplo:

```
sueldoReal = horasTrabajadas * pagoPorHora _
- deducciones
```

La sentencia más común en Visual Basic es la sentencia de asignación.

$$\text{variable} = \text{expresión}$$

que indica que el valor que resulte de evaluar la *expresión* tiene que ser almacenada en la *variable* especificada. Por ejemplo:



```
Dim Cont As Integer
Dim Intereses As Double, Capital As Double
Dim TantoPorCiento As Single
Dim Mensaje As String
' . . .
Cont = Cont + 1
Intereses = Capital * TantoPorCiento / 100
Mensaje = "La operación es correcta"
```

Comentarios

Cuando una frase se encuentra precedida de una comilla simple ('), Visual Basic interpreta que esa frase

```
'Este comentario comienza en el borde es un comentario y no
'izquierdo de la ventana ejecuta acción alguna
Texto.Text = "Hola" 'Este comentario sigue a una instrucción sobre ella.
```

Los comentarios pueden seguir a una instrucción en la misma línea o pueden ocupar una línea completa.

Los comentarios no pueden ir detrás de un carácter de continuación de línea en la misma línea. Puede agregar o eliminar símbolos que indican comentarios en un bloque de código seleccionando dos o más líneas de código y seleccionando los botones *Bloque con comentarios* () o *Bloque sin comentarios* () en la barra de herramientas *Edición*.

Constantes numéricas y de caracteres

Una constante es un valor que no cambia durante la ejecución de un programa. Visual Basic admite números decimales (base 10), hexadecimales (base 16) y octales (base 8). Un número hexadecimal va precedido por &H, un número octal va precedido por &O (letra O). El siguiente ejemplo muestra los mismos números en decimal, hexadecimal y octal.

Decima	Octal	Hexadecimal
9	&O11	&H9
15	&O17	&HF
16	&O20	&H10
20	&O24	&H14
255	&O377	&HFF

Una constante de caracteres o constante alfanumérica es una cadena de caracteres encerrada entre comillas dobles. Por ejemplo, "Hola".

Desarrollaremos ahora una aplicación para ejemplificar los temas expuestos anteriormente. La aplicación consiste en ingresar dos números en dos cajas de texto y de acuerdo al botón de pulsación en el cual se hizo clic se efectuará la operación aritmética correspondiente, mostrando el resultado de dicha operación en un control *Label*. Comience por diseñar un formulario similar al siguiente:



Los controles utilizados en el diseño del formulario junto con los valores de sus propiedades que han sido modificadas se detallan a continuación:

Objeto	Propiedad	Valor
Text1	Nombre Text	txtVariable1 0
Text2	Nombre Text	txtVariable2 0
Command1	Nombre Caption	cmdSuma Suma
Command2	Nombre Caption	cmdResta Resta
Command3	Nombre Caption	cmdDivisión División
Command4	Nombre Caption	cmdMultiplicacion Multiplicación
Label1	Nombre Caption	lblResultado "

En primer lugar nos ocuparemos de la declaración de las variables que contendrán a los valores ingresados en las caja de texto. En la sección de declaraciones del formulario escriba:

```
'declaración de variables
Dim Primero As Integer
Dim Segundo As Integer
```

Ahora nos ocuparemos de escribir el código correspondiente a los botones de pulsación. En el evento *clic* del control *cmdSuma* se asignarán los valores de las cajas de texto a las variables definidas anteriormente y se procederá a mostrar el resultado de la suma de dichas variables en control *lblResultado*:

```
Private Sub cmdSuma_Click()
'capturar números introducidos en los cuadros de texto
Primero = txtVariable1.Text
Segundo = txtVariable2.Text
lblResultado.Caption = Primero + Segundo
End Sub
```

La lógica de los restantes botones es la misma, solamente variará la operación aritmética a realizar. En el caso de la división, como sabrá, al intentar dividir cualquier número por cero dará error. Por lo cual deberá agregar dicha validación en el evento clic del botón de pulsación *División* y mostrar un mensaje si se intentar dividir por cero. El código correspondiente se muestra a continuación:

```
Private Sub cmdDivisión_Click()
'capturar números introducidos en los cuadros de texto
Primero = txtVariable1.Text
Segundo = txtVariable2.Text
If Segundo = 0 Then
MsgBox "No se puede dividir por 0."
```

```
Exit Sub
End If
lblResultado.Caption = Primero / Segundo
End Sub
```

Nota: La forma en se valida el ingreso de datos se detallará en capítulos posteriores.
Ejecute la aplicación y observe los resultados.