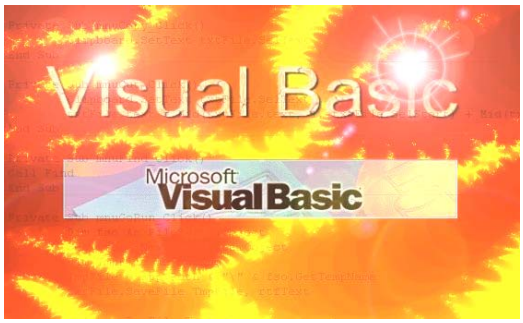
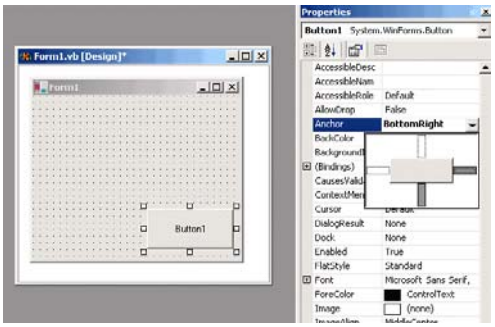
CURSO

Curso Completo de Visual Basic 6.0





Escuela Superior de Ingenieros Industriales
UNIVERSIDAD DE NAVARRA
Javier García de Jalón · José Ignacio Rodríguez
Alfonso Brazález · Patxi Funes · Eduardo Carrasco · Jesús Calleja

3. LENGUAJE BASIC

3.1 INTRODUCCIÓN

En este capítulo se explican los fundamentos del lenguaje de programación ***Basic*** utilizado en el sistema de desarrollo para ***Visual Basic 6.0*** de ***Microsoft***. En este manual se supone que el lector no tiene conocimientos previos de programación.

Un ***programa*** –en sentido informático– está constituido en un sentido general por ***variables*** que contienen los datos con los que se trabaja y por ***algoritmos*** que son las sentencias que operan sobre estos datos. Estos datos y algoritmos suelen estar incluidos dentro de ***funciones*** o ***procedimientos***.

Un procesador digital únicamente es capaz de entender aquello que está constituido por conjuntos de *unos* y *ceros*. A esto se le llama *lenguaje de máquina* o *binario*, y es muy difícil de manejar. Por ello, desde casi los primeros años de los ordenadores, se comenzaron a desarrollar los llamados *lenguajes de alto nivel* (tales como el ***Fortran***, el ***Cobol***, etc.), que están mucho más cerca del lenguaje natural. Estos lenguajes están basados en el uso de *identificadores*, tanto para los *datos* como para las componentes elementales del programa, que en algunos lenguajes se llaman *rutinas*, *procedimientos*, o *funciones*. Además, cada lenguaje dispone de una *sintaxis* o conjunto de reglas con las que se indica de modo inequívoco las operaciones que se quiere realizar.

Los *lenguajes de alto nivel* son más o menos comprensibles para el usuario, pero no para el procesador. Para que éste pueda ejecutarlos es necesario traducirlos *a su propio lenguaje de máquina*. Al paso del lenguaje de alto nivel al lenguaje de máquina se le denomina ***compilación***.

En ***Visual Basic*** esta etapa no se aprecia tanto como en otros lenguajes donde el programador tiene que indicar al ordenador explícitamente que realice dicha compilación. Los programas de ***Visual Basic*** se dice que son ***interpretados*** y no compilados ya que el código no se convierte a código máquina sino que hay otro programa que durante la ejecución "interpreta" las líneas de código que ha escrito el programador.

En general durante la ejecución de cualquier programa, el código es cargado por el sistema operativo en la memoria RAM.

3.2 COMENTARIOS Y OTRAS UTILIDADES EN LA PROGRAMACIÓN CON VISUAL BASIC.

Visual Basic 6.0 interpreta que todo lo que está a la derecha del ***carácter (')*** en una línea cualquiera del programa es un ***comentario*** y no lo tiene en cuenta para nada. El comentario puede empezar al comienzo de la línea o a continuación de una instrucción que debe ser ejecutada, por ejemplo:

```
' Esto es un comentario
A = B*x+3.4 ' también esto es un comentario
```

Los comentarios son tremendamente útiles para poder entender el código utilizado, facilitando de ese modo futuras revisiones y correcciones. En programas que no contengan muchas líneas de código puede no parecer demasiado importante, pero cuando se trata de proyectos realmente complejos, o desarrollados por varias personas su importancia es tremenda. En el caso de que el código no esté comentado este trabajo de actualización y revisión puede resultar complicadísimo.

Otro aspecto práctico en la programación es la posibilidad ***de escribir una sentencia en más de una línea***. En el caso de sentencias bastante largas es conveniente cortar la línea para que entre en la pantalla. En otro caso la lectura del código se hace mucho más pesada.

Para ello es necesario dejar un ***espacio en blanco*** al final de la línea y escribir el ***carácter (_)*** tal y como se muestra en el siguiente ejemplo:

```
str1 = "Londres" : str2 = "París"          'Se inicializan las variables
Frase = "Me gustaría mucho viajar a " & _
str1 & " y a " & str2
'El contenido de Frase sería: "Me gustaría mucho viajar a Londres y a
París
```

Una limitación a los comentarios en el código es que no se pueden introducir en una línea en la que se ha introducido el carácter de continuación (_).

La sintaxis de ***Visual Basic 6.0*** permite también incluir ***varias sentencias en una misma línea***. Para ello las sentencias deben ir separadas por el ***carácter dos puntos (:)***.

Por ejemplo:

```
m = a : n = b : resto = m Mod n ' Tres sentencias en una línea
```

3.3 PROYECTOS Y MÓDULOS

Un ***proyecto*** realizado en ***Visual Basic 6.0*** es el conjunto de todos los ficheros o ***módulos*** necesarios para que un programa funcione. La información referente a esos ficheros se almacena en un fichero del tipo ***ProjectName.vbp***. La extensión ****.vbp*** del fichero hace referencia a ***Visual Basic Project***.

Si se edita este fichero con cualquier editor de texto se comprueba que la información que almacena es la localización en los discos de los módulos que conforman ese proyecto, los controles utilizados (ficheros con extensión ***.ocx***), etc. En el caso más simple un proyecto está formado por un único formulario y constará de dos ficheros: el que define el proyecto (****.vbp***) y el que define el formulario (****.frm***).

Los módulos que forman parte de un proyecto pueden ser de varios tipos: aquellos que están asociados a un formulario (****.frm***), los que contienen únicamente líneas de código ***Basic*** (****.bas***) llamados ***módulos estándar*** y los que definen agrupaciones de código y datos denominadas clases (****.cls***), llamados ***módulos de clase***.

Un módulo ****.frm*** está constituido por un ***formulario*** y toda la información referente a los ***controles*** (y a sus propiedades) en él contenidos, además de todo el código programado en los ***eventos*** de esos controles y, en el caso de que existan, las ***funciones*** y ***procedimientos*** propios de ese formulario.

En general se llama ***función*** a una porción de código independiente que realiza una determinada actividad. En ***Visual Basic*** existen dos tipos de funciones: las llamadas ***function***, que se caracterizan por tener valor de retorno, y los ***procedimientos*** o *procedures*, que no lo tienen. En otros lenguajes, como C y C++, las ***function*** realizan los dos papeles.

Un módulo de código estándar ****.bas*** contendrá una o varias funciones y/o procedimientos, además de las variables que se desee, a los que se podrá acceder desde cualquiera de los módulos que forman el proyecto.

3.3.1 Ámbito de las variables y los procedimientos

Se entiende por ***ámbito*** de una variable[1] la parte de la aplicación donde la variable es *visible* (accesible) y por lo tanto puede ser utilizada en cualquier expresión.

3.3.1.1 Variables y funciones de ámbito local

Un módulo puede contener variables y procedimientos o funciones ***públicos*** y ***privados***. Los ***públicos*** son aquellos a los que se puede acceder libremente desde cualquier punto del proyecto.

Para definir una variable, un procedimiento o una función como ***público*** es necesario preceder a la definición de la palabra ***Public***, como por ejemplo:

```
Public Variable1 As Integer
Public Sub Procedimiento1 (Parametro1 As Integer, …)
Public Function Funcion1 (Parametro1 As Integer, …) As Integer
```

Para utilizar una variable ***Public*** o llamar a una función ***Public*** definidas en un formulario desde otro módulo se debe preceder al nombre de la variable o procedimiento del nombre del formulario al que pertenece:

```
Modulo1.Variable1
Call Modulo1.Procedimiento1(Parametro1, …)
Retorno = Modulo1.Funcion1(Parametro1, …)
```

Sin embargo si el módulo al que pertenecen la variable o el procedimiento ***Public*** es un módulo estándar (****.bas***) no es necesario poner el nombre del módulo más que si hay coincidencia de nombres con los de otro módulo también estándar. Una variable ***Private***, por el contrario, no es accesible desde ningún otro módulo distinto de aquél en el que se haya declarado. Una variable ***local*** -definida dentro de un procedimiento- no es accesible más que en el procedimiento en que está definida.

Una variable ***local*** es reinicializada (a cero, por defecto) cada vez que se entra en el *procedimiento*. Es decir, una variable ***local*** no conserva su valor entre una llamada al *procedimiento* y la siguiente. Para hacer que esto suceda, hay que declarar la variable como ***static***. ***Visual Basic*** inicializa una variable estática solamente la primera vez que se llama al *procedimiento*. Para declarar una variable estática, se utiliza la palabra ***Static*** en lugar de ***Dim*****.** Un poco más adelante se verá que ***Dim*** es una palabra utilizada para crear variables. Si un procedimiento se declara ***Static*** todas sus variables locales tienen carácter ***Static***.

3.3.1.2 Variables y funciones de ámbito global

Se puede acceder a una variable o función global desde cualquier parte de la aplicación. Para hacer que una variable sea global, hay que declararla en la ***parte general*** de un módulo ****.bas*** o de un formulario de la aplicación. Para declarar una variable global se utiliza la palabra ***Public***.

Por ejemplo:

```
Public var1_global As Double, var2_global As String
```

De esta forma se podrá acceder a las variables ***var1_global***, ***var2_global*** desde todos los formularios. La Tabla 3.1 muestra la accesibilidad de las variable en función de dónde y cómo se hayan declarado.

La diferencia entre las variables y/o procedimientos ***Public*** de los formularios y de los módulos estándar está en que las de los procedimientos deben ser cualificadas (precedidas) por el nombre del formulario cuando se llaman desde otro módulo distinto, mientras que las de un módulo estándar (****.bas***) sólo necesitan ser cualificadas si hay colisión o coincidencia de nombres.

Tipo de variable	Lugar de declaración	Accesibilidad
Global o Public	Declaraciones de *.bas	Desde todos los formularios
Dim o Private	Declaraciones de *.bas	Desde todas las funciones de ese módulo
Public	Declaraciones de *.frm	Desde cualquier procedimiento del propio formulario y desde otros precedida del nombre del modulo en el que se ha declarado
Dim o Private	Declaraciones de *.frm	Desde cualquier procedimiento del propio formulario
Dim	Cualquier procedimiento de un módulo	Desde el propio procedimiento

Tabla 3.1. Accesibilidad de las variables.

3.4 VARIABLES

3.4.1 Identificadores

La memoria de un computador consta de un conjunto enorme de ***bits*** (1 y 0), en la que se almacenan *datos* y *programas*. Las necesidades de memoria de cada tipo de dato no son homogéneas (por ejemplo, un carácter alfanumérico ocupa un ***byte*** (*8 bits*), mientras que un número real con 16 cifras ocupa 8 *bytes*), y tampoco lo son las de los programas. Además, el uso de la memoria cambia a lo largo del tiempo dentro incluso de una misma sesión de trabajo, ya que el sistema *reserva* o *libera* memoria a medida que la va necesitando.

Cada posición de memoria en la que un dato está almacenado (ocupando un conjunto de bits) puede identificarse mediante un número o una ***dirección***, y éste es el modo más básico de referirse a una determinada información. No es, sin embargo, un sistema cómodo o práctico, por la nula relación nemotécnica que una dirección de memoria suele tener con el dato contenido, y porque – como se ha dicho antes– la dirección física de un dato cambia de ejecución a ejecución, o incluso en el transcurso de una misma ejecución del programa.

Lo mismo ocurre con partes concretas de un programa determinado.
Dadas las citadas dificultades para referirse a un dato por medio de su dirección en memoria, se ha hecho habitual el uso de ***identificadores***. *Un identificador es un nombre simbólico que se refiere a un dato o programa determinado*. Es muy fácil elegir identificadores cuyo nombre guarde estrecha relación con el sentido físico, matemático o real del dato que representan. Así por ejemplo, es lógico utilizar un identificador llamado **salario_bruto** o **salarioBruto** para representar el coste anual de un empleado.

El usuario no tiene nunca que preocuparse de direcciones físicas de memoria: el sistema se preocupa por él por medio de una *tabla*, en la que se relaciona cada *identificador* con el *tipo de dato* que representa y la *posición de memoria* en la que está almacenado.

Visual Basic 6.0, como todos los demás lenguajes de programación, tiene sus propias reglas para elegir los ***identificadores***. Los usuarios pueden elegir con gran libertad los nombres de sus variables y funciones, teniendo siempre cuidado de respetar las reglas del lenguaje y de no utilizar un conjunto de ***palabras reservadas*** (***keywords***), que son utilizadas por el propio lenguaje. En el apartado 3.4.3 se explicarán las reglas para elegir nombres y cuáles son las palabras reservadas del lenguaje ***Visual Basic 6.0***.

3.4.2 Variables y constantes

Una ***variable*** es un nombre que designa a una zona de memoria (se trata por tanto de un ***identificador***), que contiene un valor de un tipo de información.

Tal y como su nombre indica, las variables pueden cambiar su valor a lo largo de la ejecución de un programa. Completando a las variables existe lo que se denomina ***constantes*** las cuales son identificadores pero con la particularidad de que el valor que se encuentra en ese lugar de la memoria sólo puede ser asignado una única vez.

El tratamiento y tipos de datos es igual al de las variables.

Para declarar un dato como constante únicamente es necesario utilizar la palabra ***Const*** en la declaración de la variable. Si durante la ejecución se intenta variar su valor se producirá un error.

Ejemplos:

```
Const MyVar = 459 ’ Las constantes son privadas por defecto.
Public Const MyString = "HELP" ' Declaración de una constante pública.
Private Const MyInt As Integer = 5 ' Declaración de un entero constante.
Const Str = "Hi", PI As Double = 3.14 ' Múltiples
```

Visual Basic 6.0 tiene sus propias constantes, muy útiles por cierto. Algunas ya se han visto al hablar de los colores. En general estas constantes empiezan por ciertos caracteres como ***vb*** (u otros similares que indican a que grupo pertenecen) y van seguidas de una o más palabras que indican su significado. Para ver las constantes disponibles se puede utilizar el comando ***View/Object Browser***, tal como se muestra en la Figura 3.1.

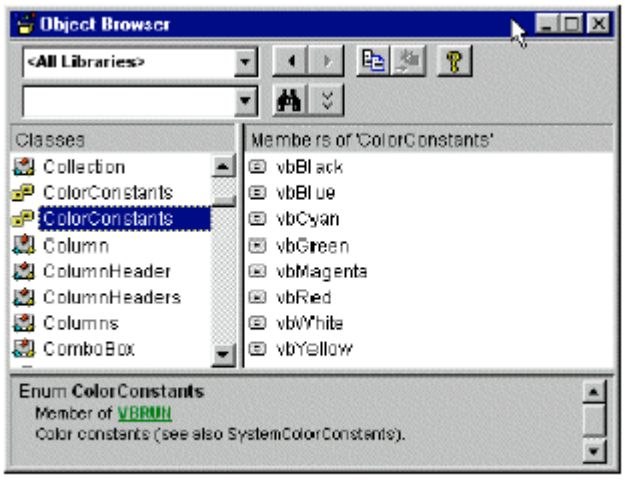


Figura 3.1. Constantes de color predefinidas.

3.4.3 Nombres de variables

El nombre de una variable (o de una constante) tiene que comenzar siempre por una letra y puede tener una longitud hasta 255 caracteres. No se admiten espacios o caracteres en blanco, ni puntos (.), ni otros caracteres especiales.

Los caracteres pueden ser letras, dígitos, el carácter de subrayado (_) y los caracteres de declaración del tipo de la variable (%, &, !, @, y $). El nombre de una variable no puede ser una ***palabra reservada*** del lenguaje (***For***, ***If***, ***Loop***, ***Next***, ***Val*** , ***Hide***, ***Caption***, ***And***, ***...***).

Para saber cuáles son las palabras reservadas en ***Visual Basic 6.0*** puede utilizarse el ***Help*** de dicho programa, buscando la referencia ***Reserved Words***. De ordinario las palabras

reservadas del lenguaje aparecen de color azul en el editor de código, lo que hace más fácil saber si una palabra es reservada o no.

A diferencia de ***C***, ***Matlab***, ***Maple*** y otros lenguajes de programación, ***Visual Basic 6.0 no distingue entre minúsculas y mayúsculas***. Por tanto, las variables ***LongitudTotal*** y ***longitudtotal*** son consideradas como idénticas (la misma variable). En ***Visual Basic 6.0*** es habitual utilizar las letras mayúsculas para separar las distintas palabras que están unidas en el nombre de una variable, como se ha hecho anteriormente en la variable ***LongitudTotal***.

También es habitual entre los programadores, aunque no obligado, el utilizar nombres con todo mayúsculas para los nombres de las constantes simbólicas, como por ejemplo ***PI***.

3.4.4 Tipos de datos

Al igual que ***C*** y otros lenguajes de programación, ***Visual Basic*** dispone de distintos tipos de datos, aplicables tanto para constantes como para variables. La Tabla 3.2 muestra los tipos de datos disponibles en ***Visual Basic***.

Tipo	Descripción	Carácter de declaración	Rango
Boolean	Binario		True o False
Byte	Entero corto		0 a 255
Integer	Entero (2 bytes)	%	-32768 a 32767
Long	Entero largo (4 bytes)	&	-2147483648 a 2147483647
Single	Real simple precisión (4 bytes)	!	-3.40E+38 a 3.40E+38
Double	Real doble precisión (8 bytes)	#	-1.79D+308 a 1.79D+308
Currency	Número con punto decimal fijo (8 bytes)	@	-9.22E+14 a 9.22E+14
String	Cadena de caracteres (4 bytes + 1 byte/car hasta 64 K)	$	0 a 65500 caracteres.
Date	Fecha		1 de enero de 100 a 31 de diciembre de 9999. Indica también la hora, desde 0:00:00 a 23:59:59.
Variant	Fecha/hora; números enteros, reales, o caracteres (16 bytes + 1 byte/car. en cadenas de caracteres)	ninguno	F/h: como Date números: mismo rango que el tipo de valor almacenado
User-defined	Cualquier tipo de dato o estructura de datos. Se crean utilizando la sentencia Type (Ver apartado 3.10)	ninguno	

Tabla 3.2. Tipos de datos en **Visual Basic 6.0**.

En el lenguaje ***Visual Basic 6.0*** existen dos formas de agrupar varios valores bajo un mismo nombre. La primera de ellas son los ***arrays*** (vectores y matrices), que agrupan datos

de tipo homogéneo. La segunda son las ***estructuras***, que agrupan información heterogénea o de distinto tipo. En ***Visual Basic 6.0*** las estructuras son verdaderos *tipos de datos definibles por el usuario*.

Para declarar las variables se utiliza la sentencia siguiente:

```
Dim NombreVariable As TipoVariable
```

cuyo empleo se muestra en los ejemplos siguientes:

```
Dim Radio As Double, Superficie as Single
Dim Nombre As String
Dim Etiqueta As String * 10
Dim Francos As Currency
Dim Longitud As Long, X As Currency
```

En ***Visual Basic 6.0*** no es estrictamente necesario declarar todas las variables que se van a utilizar (a no ser que se elija la opción ***Option Explicit*** que hace obligatorio el declararlas), y hay otra forma de declarar las variables anteriores, utilizando los caracteres especiales vistos anteriormente.
Así por ejemplo, el tipo de las variables del ejemplo anterior se puede declarar al utilizarlas en las distintas expresiones, poniéndoles a continuación el carácter que ya se indicó en la Tabla 3.2, en la forma:

```
Radio#          doble precisión
Nombre$         cadena de caracteres
Francos@        unidades monetarias
Longitud&       entero largo
```

Esta forma de indicar el tipo de dato no es la más conveniente. Se mantiene en las sucesivas versiones de ***Visual Basic*** por la compatibilidad con códigos anteriores. Es preferible utilizar la notación donde se escribe directamente el tipo de dato.

3.4.5 Elección del tipo de una variable

Si en el código del programa se utiliza una variable que no ha sido declarada, se considera que esta variable es de tipo ***Variant***. Las variables de este tipo se adaptan al tipo de información o dato que se les asigna en cada momento.

Por ejemplo, una variable tipo ***Variant*** puede contener al principio del programa un *string* de caracteres, después una variable de *doble precisión*, y finalmente un número *entero*. Son pues variables muy flexibles, pero su uso debe restringirse porque ocupan ***más memoria*** (almacenan el tipo de dato que contienen, además del propio valor de dicho dato) y requieren ***más tiempo de CPU*** que los restantes tipos de variables.

En general es el tipo de dato (los valores que puede tener en la realidad) lo que determina qué tipo de variable se debe utilizar. A continuación se muestran algunos ejemplos:

- ***Integer*** para numerar las filas y columnas de una matriz no muy grande

- ***Long*** para numerar los habitantes de una ciudad o los números de teléfonos

- ***Boolean*** para una variable con sólo dos posibles valores (sí o no)

- ***Single*** para variables físicas con decimales que no exijan precisión

- ***Double*** para variables físicas con decimales que exijan precisión

- ***Currency*** para cantidades grandes de dinero

Es muy importante tener en cuenta ***que se debe utilizar el tipo de dato más sencillo que represente correctamente el dato real*** ya que en otro caso se ocupará más memoria y la ejecución de los programas o funciones será más lenta.

3.4.6 Declaración explícita de variables

Una variable que se utiliza sin haber sido declarada toma por defecto el tipo ***Variant***. Puede ocurrir que durante la programación, se cometa un error y se escriba mal el nombre de una variable.

Por ejemplo, se puede tener una variable " declarada como *entera*, y al programar referirse a ella por error como "; ***Visual Basic*** supondría que ésta es una nueva variable de tipo ***Variant***.

Para evitar este tipo de errores, se puede indicar a ***Visual Basic*** que genere un mensaje de error siempre que encuentre una variable no declarada previamente. Para ello lo más práctico es establecer una opción por defecto, utilizando el comando ***Environment*** del menú ***Tools/Options***; en el cuadro que se abre se debe poner ***Yes*** en la opción ***Require Variable Declaration***. También se puede hacer esto escribiendo la sentencia siguiente en la sección de declaraciones de cada formulario y de cada módulo:

```
Option Explicit
```

Continuará.....

Nota de Radacción: El lector puede descargar este capítulo y capítulos anteriores del curso desde la sección ***"Artículos Técnicos"*** en el sitio web de **EduDevices (www.edudevices.com.ar)**