



Interacción Persona Ordenador

Visual Basic

Tema 3

Constantes, Variables y Expresiones

Visual Basic

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** y un número octal va precedido por **&O**.
 - 9, 15, 1034 Son números decimales.
 - **&H9**, **&HF**, **&H40A** Son números hexadecimales.
 - **&O11**, **&O17**, **&O2012** Son números octales.
- Una constante de caracteres o constante alfanumérica es una cadena de caracteres encerrada entre comillas dobles ("").
 - "Buenos Días"

Visual Basic

Variables

- Una **variable** contiene un valor que puede modificarse a lo largo de la ejecución de la aplicación. Cada variable tiene atributos propios, como:
 - **Nombre:** Es el nombre que utilizamos para referirnos a la variable en la aplicación.
 - **Tipo:** El tipo determina qué clase de valores puede almacenar la variable.
 - **Ámbito:** El ámbito especifica en que parte de la aplicación la variable es conocida y por tanto puede utilizarse.
- El nombre de una variable **tiene que comenzar por una letra**, puede tener hasta 255 caracteres de longitud y debe ser único dentro del ámbito.
- Los caracteres pueden ser letras, dígitos, el carácter de subrayado y los caracteres de declaración de tipo de variable (% , & , ! , # , @ y \$).
- No se pueden utilizar el punto ni otros caracteres que tienen un significado especial para Visual Basic; por ejemplo, los paréntesis. Los caracteres de declaración de tipo, cuando se utilicen, deben ocupar la última posición.
- Un nombre de una variable no puede ser una palabra reservada (*palabra que tiene un significado especial para Visual Basic*).

Visual Basic

Tipos de Datos

Tipo de Dato	Valores que puede contener	Car-tipo
Boolean	True(-1) o False(0) (2 bytes)	
Byte	Enteros (+) en el rango de: 0 a 255 (1byte)	
Integer	Enteros (+/-) en el rango de: -32768 a 32767 (2 bytes)	%
Long	Enteros Largos (+/-), de: -2147483648 a 2147483647 (4 bytes)	&
Single	Punto flotante simple precisión (+/-). Valor mayor positivo: 3,4 veces 10 a la 38ª potencia (4 bytes)	!
Double	Punto flotante doble precisión (+/-). Valor mayor positivo: 1,8 veces 10 a la 308ª potencia (8 bytes)	#
Currency	Entero con punto decimal fijo (+/-) (8 bytes)	@
String	Cadenas de longitud fija (1 byte por carácter) (hasta 64 K aproximadamente)	
String	Cadenas de longitud variable (10 bytes + 1 byte por carácter) (2 ³¹ caracteres aprox.)	\$
Date	Fechas de rango de 1/1/100 a 31/12/9999 (8 bytes)	
Decimal	Números con 0 a 28 decimales (14 bytes) (<i>no se puede declarar una variable de este tipo. Sólo se puede utilizar con Variant</i>)	

Visual Basic

Tipos de Datos

Tipo de Dato	Valores que puede contener
Object	<p>Las variables Object se almacenan como direcciones de 32 bits (4 bytes) que hacen referencia a objetos dentro de una aplicación o de cualquier otra aplicación. Un variable declarada como Object es una variable que puede asignarse subsiguientemente (mediante la instrucción Set) para referirse a cualquier objeto real reconocido por la aplicación. Cuando declaremos variables objeto, debemos intentar clases específicas (como <i>TextBox</i> en vez de <i>Control</i>, o <i>DataBase</i> en vez de <i>Object</i>) mejor que el tipo genérico Object. Visual Basic puede resolver referencias a las propiedades y métodos de objetos con tipos específicos antes de que se ejecute la aplicación. Esto permite a la aplicación funcionar más rápido en tiempo de ejecución. En el Examinador de Objetos se muestran las clases específicas. Cuando trabajemos con objetos de otras aplicaciones, en vez de usar Variant o el tipo genérico <i>Object</i>, debemos declarar los objetos como se muestra en la lista <i>Clases</i> en el Examinador de Objetos. Esto asegura que Visual Basic reconozca el tipo específico de objeto al que estamos haciendo referencia, lo que permite resolver la referencia en tiempo de ejecución.</p>
Variant	<p>Una variable Variant es capaz de almacenar todos los tipos de datos definidos en el sistema. No tiene que realizar ninguna conversión si esos tipos de datos se asignan directamente a una variable tipo Variant. Visual Basic realiza automáticamente cualquier conversión necesaria. Si bien podemos realizar operaciones con variables Variant sin saber exactamente el tipo de dato que contienen, hay que hacer una serie de especificaciones.</p> <ul style="list-style-type: none">• Si realizamos operaciones aritméticas o funciones sobre un Variant, el Variant debe contener un número.• Si estamos concatenando cadenas, debemos utilizar el operador & en vez del operador +.

Visual Basic

Tipos de Datos

Tipo de Dato	Ejemplo
Boolean	<pre>Dim SienMarcha As Boolean ' Comprueba si la cinta está en marcha If Recorder.Direction = 1 Then SienMarcha = True End If</pre> <p>Podemos utilizar este tipo de dato en múltiples situaciones, por ejemplo para almacenar el resultado de una expresión relacional, que sólo puede ser cierta o falsa, o para devolver el valor de una función indicando si todo va bien, se suele retornar <i>True</i>, o se ha producido un error, devuelve <i>False</i>.</p>
Byte	<pre>Dim Numero As Byte ' Realizamos una suma</pre> <hr/> <pre>Private Sub Form_Click() Numero = Numero + 1 Print Numero End Sub</pre> <p>Tiene la ventaja de que al ocupar sólo un byte el trabajo con una variable de este tipo es muy rápido, además de ocupar bastante menos que cualquier otro tipo de dato.</p>

Visual Basic

Tipos de Datos

Tipo de Dato	Ejemplo
Integer	<pre>Dim Numero As Integer ' Realizamos una suma</pre> <hr/> <pre>Private Sub Form_Click() Numero = Numero + 1 Print Numero End Sub</pre> <p>Nos permite trabajar tanto con números enteros, tanto negativos como positivos, en un rango que en la mayoría de las ocasiones es suficiente para nuestras necesidades. Visual Basic redondea en vez de truncar la parte fraccionaria de un número de signo flotante antes de asignarlo a un entero.</p>
Long	<pre>Dim Numero As Long ' Realizamos una suma</pre> <hr/> <pre>Private Sub Form_Click() Numero = Numero + 1 Print Numero End Sub</pre> <p>Ocupa 4 bytes en memoria, y en ella podemos almacenar prácticamente cualquier número entero, positivo o negativo.</p>

Visual Basic

Tipos de Datos

Tipo de Dato	Ejemplo
Single	<pre>Dim Numero1 As Single Dim Numero2 As Single Numero1 = 2.2E+124 Numero2 = 1E+101</pre> <p>Son los que nos permiten utilizar números en punto o coma flotante, lo que significa que la coma decimal no tiene una posición predeterminada, sino que puede estar en cualquier lugar donde sea necesario. Desde cantidades ínfimas, en las que la coma decimal tiene detrás decenas o centenas de ceros, hasta magnitudes impensables.</p>
Double	<pre>Dim Numero1 As Double Dim Numero2 As Double Numero1 = 1.22E+121 Numero2 = 9.8E+307</pre> <p>Son los que nos permiten utilizar números en punto o coma flotante, lo que significa que la coma decimal no tiene una posición predeterminada, sino que puede estar en cualquier lugar donde sea necesario. Desde cantidades ínfimas, en las que la coma decimal tiene detrás decenas o centenas de ceros, hasta magnitudes impensables.</p>

Visual Basic

Tipos de Datos

Tipo de Dato	Ejemplo
Currency	<pre>Dim Numero1 As Currency Numero1 = 1.23456789012346E+19</pre> <p>Se caracteriza por tener una coma fija, con 4 dígitos decimales. Este tipo de dato está especialmente indicado para trabajar con cantidades grandes que siempre tendrán una parte decimal fija. La ocupación en memoria de una variable de este tipo es de 8 bytes, y el rango de valores que puede contener va hasta los ciento de billones, tanto positivos como negativos.</p>
String	<div><pre>Dim Cadena As String Cadena = "Longitud Variable"</pre></div> <div><pre>Dim LongitudFija As String * 50</pre></div> <p>De forma predeterminada, una variable o argumento de cadena es una Cadena de Longitud Variable; la cadena crece o disminuye según se le asigne nuevos datos. También podemos declarar Cadenas de Longitud Fija (<i>String * tamaño</i>). Si a la cadena de longitud fija LongitudFija le asignamos menos de 50 caracteres, se rellenará con espacios en blanco hasta el total de 50 caracteres. Si asignamos una cadena demasiado larga a una variable tipo Cadena de Longitud Fija, Visual Basic simplemente truncará los caracteres.</p> <p>El tipo String utilizado sin más genera una cadena de longitud variable, capaz de contener hasta 2 millones de caracteres. Aparentemente no tiene ningún sentido utilizar cadenas de longitud fija cuando siempre podemos utilizar cadenas de longitud variable, con la flexibilidad de saber que podemos utilizar cadenas con cualquier número de caracteres. Sin embargo, mientras que una cadena de longitud fija de un carácter ocupa 1 byte, una de longitud variable conteniendo el mismo carácter ocupa 11 bytes. Esto es así porque las variables de tipo cadena de longitud variable además de asignar memoria para almacenar los caracteres de la cadena necesitan un espacio adicional, para conocer la longitud actual de la cadena y su dirección (10 bytes).</p>

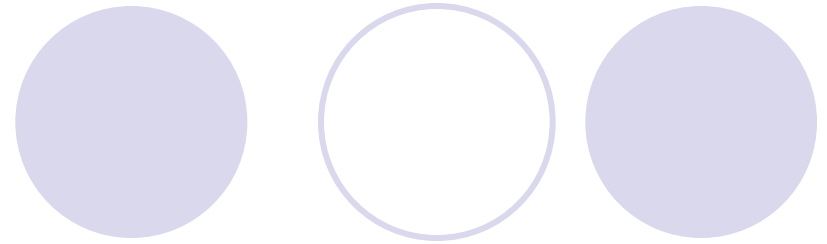
Visual Basic

Tipos de Datos

Tipo de Dato	Ejemplo
Date	<pre>Dim Uno As Date</pre> <hr/> <pre>Private Sub Form_Click() Print Uno End Sub</pre> <p>Nos permite almacenar cualquier fecha desde el 1 de enero del año 100 hasta el 31 de diciembre del año 9999. En realidad la representación interna de una variable de este tipo, que ocupa 8 bytes, es la de un número en coma flotante, en el que la parte entera contiene la fecha y la parte decimal la hora.</p>
Objeto	<pre>Dim VarObj As Object Set VarObj = opendatabase("c:\Vb5\biblio.mdb")</pre> <p>Object, que ocupa 4 bytes en memoria, es capaz de contener una referencia a cualquier objeto de la aplicación, por ejemplo, un formulario o un componente OLE. Es un tipo de dato muy potente, a partir del cual podemos, por ejemplo, crear varias instancias de un mismo formulario.</p>

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant



- Valor Empty.

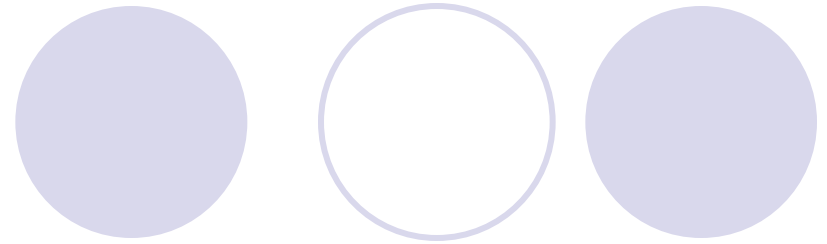
- A veces necesitaremos saber si se ha asignado un valor a una variable existente. Una variable **Variant** tiene el valor **Empty** antes de asignarle un valor. El valor **Empty** es un valor especial distinto de 0 o una cadena de longitud cero (" ") o el valor **Null**. Para comprobar este valor podemos utilizar la función **IsEmpty**:

If IsEmpty (Z) Then Z = 0

- Cuando un **Variant** contiene el valor **Empty**, podemos usarlo 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**. Podemos establecer una variable **Variant** de nuevo como **Empty** si asignamos la palabra clave **Empty** al **Variant**.

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant



• Valor 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 la 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 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 podemos asignar un **Null** mediante la palabra clave **Null**:

z = Null

- Podemos utilizar la función **IsNull** para comprobar si una variable **Variant** contiene un **Null**:

If IsNull (X) And IsNull (Y) Then

Z = Null

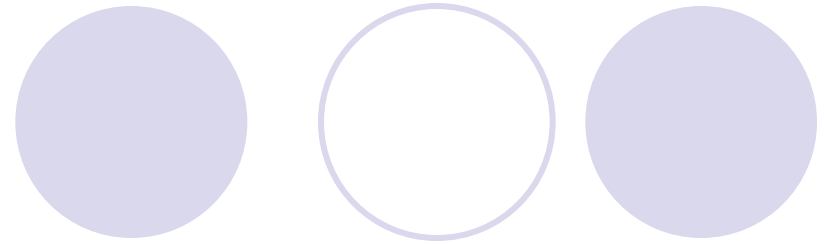
Else

Z = 0

End If

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant

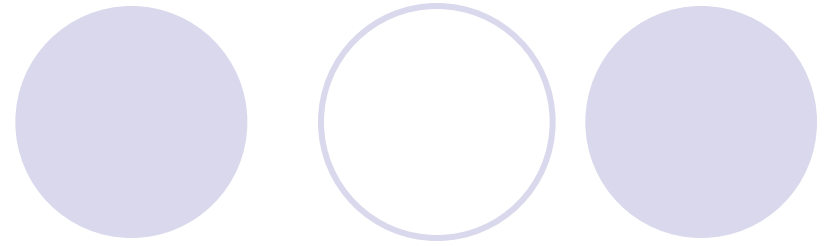


- Valor Null.

- Si se 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 contenga variables **Variant** (**Null** no se propaga a través de determinadas funciones). Podemos 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 utilizamos **Null** en nuestra aplicación, no tendremos que escribir código que compruebe su existencia y lo trate.

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant



- Valor Error.
 - En un **Variant**, **Error** es un valor especial que se utiliza para indicar que se ha producido una condición de error en un procedimiento. Sin embargo, a diferencia de otros tipos de error, no se produce el tratamiento de errores a nivel normal de la aplicación. Esto nos permite elegir alternativas basadas en el valor de error. Los valores de error se crean convirtiendo números reales en valores de error mediante la función **CVerr**.

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant

- Representación interna de los valores en los tipos Variant.

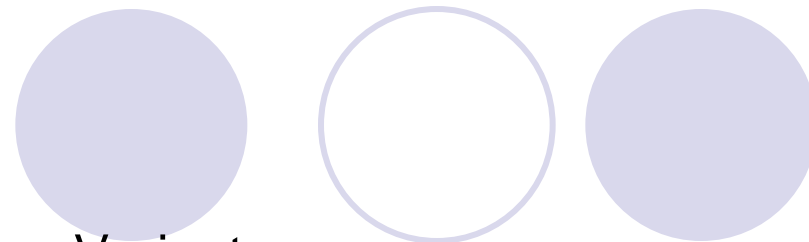
- Una variable **Variant** no es una variable sin tipo, sino una variable que puede cambiar libremente de tipo. Un tipo **Variant** siempre ocupa 16 bytes, independientemente de lo que almacenemos en él. Los objetos, las cadenas y las matrices no se almacenan físicamente en el **Variant**; en estos casos, 4 bytes del **Variant** se utilizan para almacenar una referencia de un objeto, o un puntero a la cadena o a la matriz. Los datos reales se almacenan en otro lugar. Visual Basic trata las conversiones automáticamente. Sin embargo, si deseamos saber que valor está utilizando Visual Basic, podemos emplear la función **VarType**:

If VarType (X) = 5 Then X = CSng (X) ‘ Convierte a Single.

- (Por ejemplo, si almacenamos valores decimales en una variable **Variant**, Visual Basic utiliza siempre la representación interna **Double**. Si sabemos que nuestra aplicación no necesita la máxima precisión, podemos convertir los valores a **Single** o **Currency**).

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant



- Valores numéricos almacenados en tipos Variant.

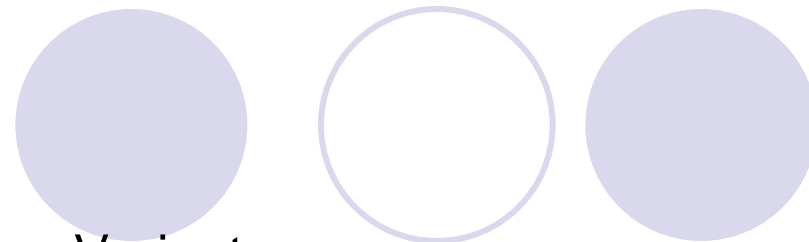
- Cuando almacenamos números enteros en variables **Variant**, Visual Basic utiliza la representación más compacta posible. Por ejemplo, si almacenamos un número pequeño sin fracción decimal en una variable **Variant**, Visual Basic utiliza una representación **Integer** para el valor. Si asignamos un número mayor, Visual Basic usará un valor **Long** o, si es muy largo o tiene un componente decimal, Visual Basic usará un valor **Double**.
- Podemos utilizar una variable **Variant** para almacenar un valor numérico como **Currency** para evitar errores de redondeo en cálculos posteriores. Visual Basic proporciona varias funciones de conversión que podemos utilizar para convertir valores de un tipo específico (por ejemplo, para convertir un valor a **Currency**, podemos utilizar la función **CCur**):

PagoporSemana = CCur (horas * PagoporHora)

- Si intentamos realizar una operación o función matemática sobre un tipo **Variant** que no contiene un número o algo que se pueda interpretar como un número, se producirá un error. No podemos realizar operaciones aritméticas con valores como B3 o 1040EZ, pero sí con +10 ó -1,7E6.
- Si queremos determinar si una variable **Variant** contiene un valor que se pueda usar como un número, podemos utilizar la función **IsNumeric**.

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant



- Valores numéricos almacenados en tipos Variant.

- Cuando Visual Basic convierte una representación que no es numérica (como una cadena que contenga un número) a un valor numérico, utiliza la *Configuración Regional* (especificada en el Panel de Control de Windows) para interpretar el separador de miles, el separador decimal y el símbolo de la moneda. Así, si la *Configuración Regional del Panel de Control de Windows* está establecida como Estados Unidos, Canadá o Australia, estas 2 instrucciones devolverían «verdadero»:

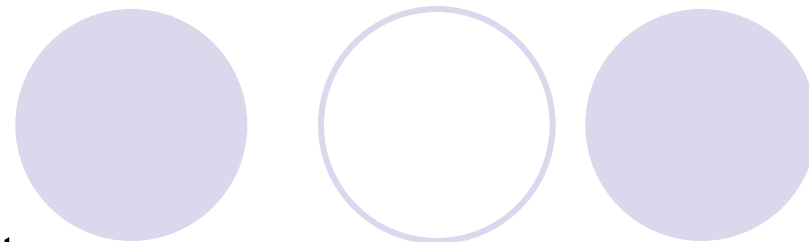
IsNumeric (“\$100”)

IsNumeric (“1,560.50”)

- Si asignamos un **Variant**, que contiene un número, a una variable de cadena o a una propiedad, Visual Basic convierte automáticamente la representación del número a una cadena. Si deseamos convertir explícitamente un número a una cadena, debemos utilizar la función **CStr**. También podemos utilizar la función **Format** para convertir un número a una cadena que incluya formato, como los símbolos de moneda, separador de miles y separador decimal. La función **Format** utiliza automáticamente los símbolos adecuados de acuerdo con el cuadro de diálogo ***Configuración Regional del Panel de Control de Windows***.

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant



• Cadenas almacenadas en tipos Variant.

- Si el operador **+** se utiliza con dos variables **Variant**, hay que tener en cuenta que si ambas variables contienen números, el operador **+** realizará una suma. Si ambas variables **Variant** contienen cadenas, el operador **+** realizará una concatenación de cadenas. Pero si uno de los valores se representa con un número y el otro con una cadena, la situación es más complicada. Visual Basic intenta primero convertir la cadena en un número. Si la conversión funciona, el operador **+** suma los dos valores, pero si falla, genera el error: *El tipo no coincide*.
- Para asegurarnos de que se produce la concatenación, sin tener en cuenta la representación del valor de las variables, debemos utilizar el operador **&**.

- Sub Form_Click ()
- Dim X, Y
- X = "6"
- Y = "7"
- Print X + Y, X & Y
- X = 6
- Print X + Y, X & Y
- End Sub

Resultados:

67 67

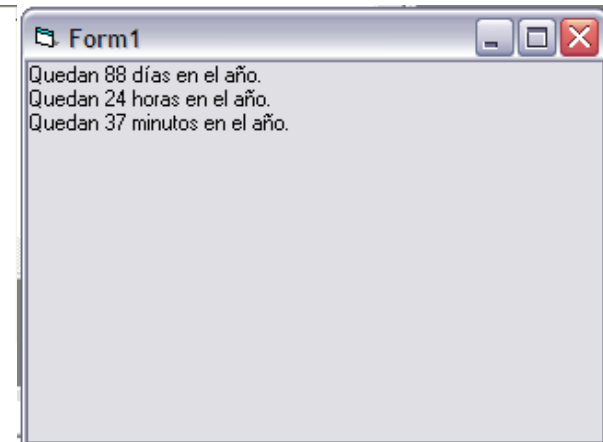
13 67

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant

- Valores de fecha y hora almacenados en tipos Variant.
 - La función **DateSerial** devuelve el número de días que quedan del año:

```
Private Sub Form_Load()  
    Dim ahora, días, horas, minutos  
    ahora = Now 'Now devuelve la fecha y la hora actuales.  
    días = Int(DateSerial(Year(ahora) + 1, 1, 1) - ahora)  
    horas = 24 - Hour(ahora)  
    minutos = 60 - Minute(ahora)  
    Print "Quedan " & días & " días en el año."  
    Print "Quedan " & horas & " horas en el año."  
    Print "Quedan " & minutos & " minutos en el año."  
End Sub
```



- También podemos realizar operaciones matemáticas sobre valores de fecha y hora. Sumar o restar enteros agrega o resta días; sumar o restar fracciones agrega o resta horas. Por tanto, al sumar 20, se agregan 20 días, mientras que al restar 1/24 se resta una hora.
- El intervalo para las fechas almacenadas en variables **Variant** es 1 de Enero de 0100 a 31 de Diciembre de 9999 (siempre que el cálculo de la fecha este dentro del calendario gregoriano: 1752).

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant

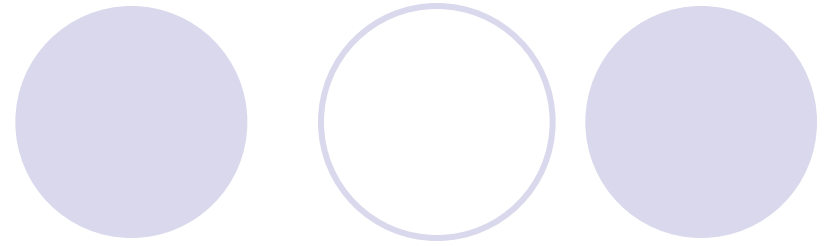
- Valores de fecha y hora almacenados en tipos Variant.
 - Podemos utilizar en el código literales de fecha y hora si los escribimos entre signos de número (#), de la misma forma que ponemos literales de cadenas entre comillas (“). Por ejemplo, podemos comparar un **Variant** que contenga un valor de fecha y hora con una fecha literal:
 - **If AlgunaFecha > #3 / 6 / 93# Then**
 - De forma similar, podemos comparar un valor de fecha y hora con un literal completo de fecha y hora:
 - **If AlgunaFecha > #3 / 6 / 93 1:20pm# Then**
 - Si no incluimos una hora en el literal de fecha y hora, Visual Basic establece la parte horaria del valor como medianoche (el inicio del día). Si no incluimos una fecha en un literal de fecha y hora, Visual Basic establece la parte de la fecha del valor como 30 de Diciembre de 1899.
 - Visual Basic acepta una amplia variedad de formatos de fecha y hora en literales:
 - **AlgunaFecha = #3-6-93 13:20#**
 - **AlgunaFecha = #Marzo 27, 1993 1:20am#**
 - **AlgunaFecha = #Abr-2-93#**
 - **AlgunaFecha = #4 Abril 1993#**

Visual Basic

Tipos de Datos – Peculiaridades sobre Variant

- Valores de fecha y hora almacenados en tipos Variant.
 - De la misma forma que podemos utilizar la función **IsNumeric** para averiguar si una variable **Variant** contiene un valor que se pueda considerar como valor numérico válido, podemos utilizar la función **IsDate** para averiguar si una variable **Variant** contiene un valor que se pueda considerar como un valor válido de fecha y hora. Podemos entonces utilizar la función **CDate** para convertir el valor en un valor de fecha y hora.
 - Por ejemplo, el siguiente código prueba la propiedad **Text** de un cuadro de texto mediante **IsDate**. Si la propiedad contiene texto que pueda considerarse una fecha válida, Visual Basic convierte el texto en una fecha y calcula los días que faltan hasta final de año:

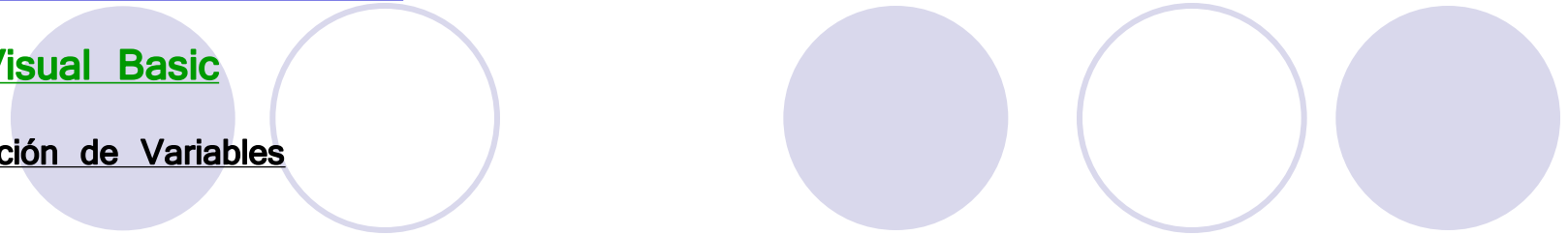
```
Dim AlgunaFecha, días
If IsDate(Text1.Text) Then
    AlgunaFecha = CDate(Text1.Text)
    días = DateSerial(Year(AlgunaFecha) + 1, 1, 1) - AlgunaFecha
    Text2.Text = "Quedan" & días & " días en el año."
Else
    MsgBox Text1.Text & " no es una fecha válida."
End If
```



- Objetos almacenados en tipos Variant.
 - Es posible almacenar objetos en variables **Variant**. Esto puede resultar muy útil cuando necesitamos tratar “elegantemente” gran variedad de tipos de datos, incluyendo objetos. Por ejemplo, todos los elementos de una matriz deben tener el mismo tipo de dato. Establecer el tipo de dato de una matriz como **Variant** le permite almacenar objetos junto con otros tipos de datos de una matriz.

Visual Basic

Declaración de Variables



- Antes de utilizar una variable, es aconsejable declarar su tipo. Una forma de hacerlo es utilizando la sentencia **Dim** (o una de las palabras **Public**, **Private** o **Static**). Cualquier declaración de éstas inicia las variables **numéricas** con el valor **cero** y las variables **alfanuméricas** con el valor **nulo**:

<code>Dim I As Integer</code>	<code>' I es una variable Entera.</code>
<code>Dim R As Double</code>	<code>' R es una variable Real de</code>
	<code>' posición doble.</code>
<code>Dim Nombre As String</code>	<code>' Nombre es una variable para</code>
	<code>' contener una cadena de caracteres</code>
	<code>' de longitud variable.</code>
<code>Dim Etiqueta As String * 10</code>	<code>' Etiqueta es una variable para</code>
	<code>' contener una cadena de caracteres</code>
	<code>' de longitud fija (10 caracteres).</code>
<code>Dim F As Currency</code>	<code>' F es una variable fraccionaria.</code>
<code>Dim L As Long, X As Currency</code>	<code>' L es una variable entera larga y</code>
	<code>' X es una variable fraccionaria.</code>

- En una sentencia **Dim** se pueden hacer más de una declaración. La cláusula opcional **As tipo** de la instrucción **Dim** nos permite definir el tipo de dato o de objeto de la variable que vamos a declarar.

Visual Basic

Declaración de Variables

- Cuando se declara una variable y no se especifica el tipo, se asume que es de tipo **Variant**:

```
Dim A, B As Integer
```

```
' A es de tipo Variant (por omisión)
```

```
' B es de tipo Entero.
```

- Otra forma de declarar una variable es utilizando los caracteres de declaración de tipo:

- **I%** Variable Entera.
- **R#** Variable Real de posición doble.
- **Nombre\$** Cadena de Caracteres de longitud variable.
- **F@** Variable Fraccionaria.

Visual Basic

Declaración de Variables

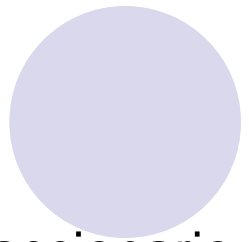
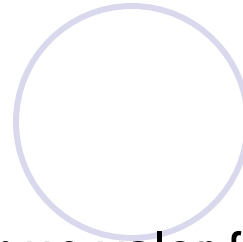
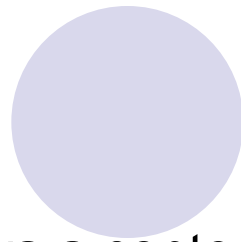
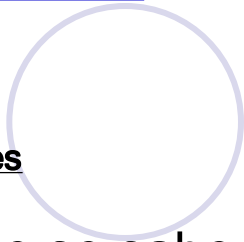
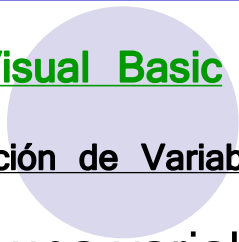
- Cuando una variable se utiliza y no se declara explícitamente, se asume que es de tipo **Variant**:

```
L = "Dato:"           ' Variable de tipo String.  
L = 3.25678           ' Variable de tipo Double.
```

- Suponiendo que **L** no ha sido declarada explícitamente, la sentencias anteriores declaran **L** como una variable **Variant** que ha cambiado su tipo para comportarse como una *cadena de caracteres*, y a continuación vuelve a cambiar su tipo para comportarse como una *variable real de precisión doble*.

Visual Basic

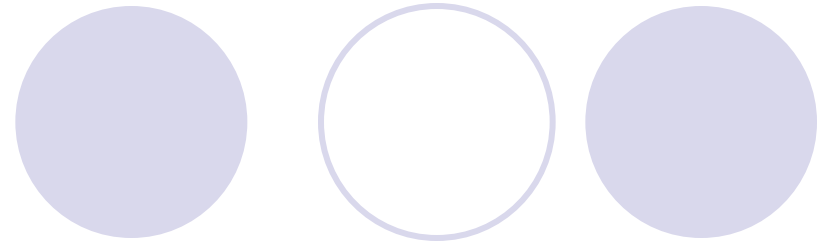
Declaración de Variables



- Si de una variable se sabe que nunca va a contener un valor fraccionario, es mejor declararla como entera, ya que las operaciones con enteros son más rápidas. En caso contrario, si el valor no va a tener más de 4 dígitos decimales y no más de 14 dígitos enteros, es conveniente declararla como fraccionaria (**Currency**). Esto es así porque, partiendo de que un ordenador internamente trabaja en binario, en las variables de tipo **Currency** no tiene lugar el error producido al convertir un valor en base 10 al mismo valor en base 2 y viceversa, que sí tiene lugar cuando la variable es de tipo **Single** o **Double**.
- Cuando una variable numérica de un tipo se asigna a otra variable numérica de un tipo diferente, Visual Basic realiza la conversión correspondiente.

Visual Basic

Declaración de Variables – Declaración Explícita



- En Visual Basic no es necesario declarar una variable antes de utilizarla. Sin embargo, esta forma de trabajar puede ser fuente de errores:

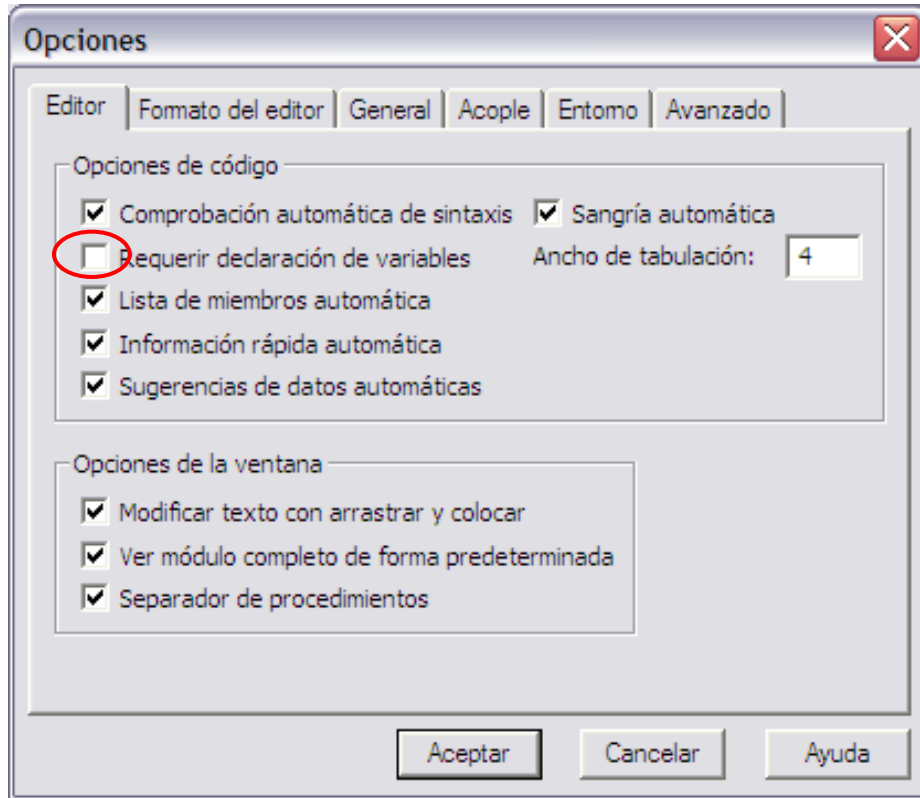
```
Dim M As Integer, N As Integer
VarTemp = M      ' Asignamos la variable declarada M a una
                  ' variable NO declarada VarTemp

N = VaTemp        ' Asignamos la variable anterior a la
                  ' variable declarada N. En este caso
                  ' hemos cometido un error y hemos
                  ' escrito mal la variable (VaTemp).
```

- En este caso, *VarTemp* no se ha declarado explícitamente. Esto no supone un error, ya que Visual Basic se encarga de crear dicha variable. Si por error escribimos mal la variable, cuando Visual Basic encuentre la nueva variable, no podrá determinar si hemos cometido un error o hemos definido una nueva variable.

Visual Basic

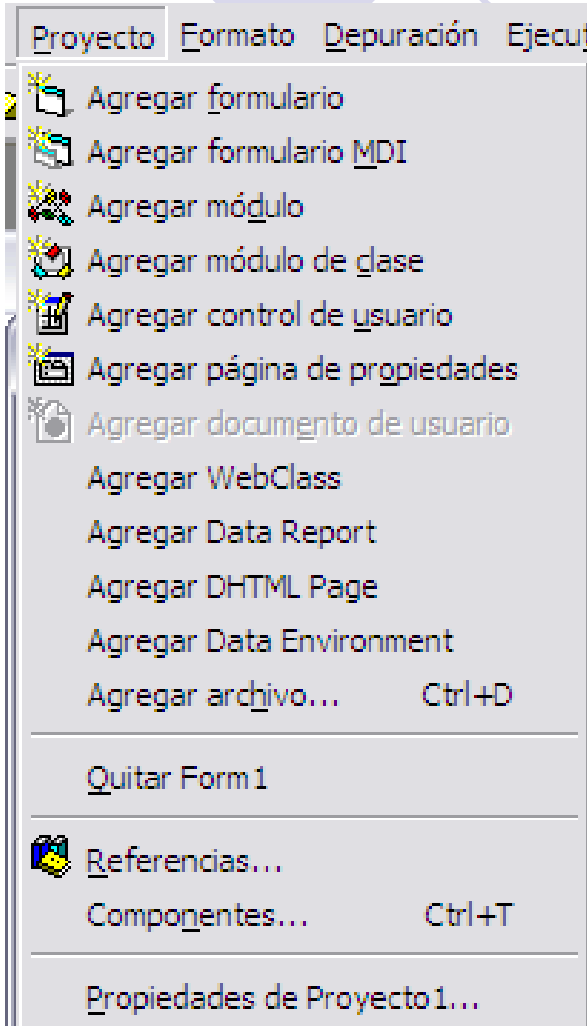
Declaración de Variables – Declaración Explícita



- Para evitar este tipo de errores, podemos indicar a Visual Basic que genere un mensaje de error siempre que se encuentre una variable no declarada explícitamente. Para ello, escribiremos la sentencia **Option Explicit** en la sección de declaraciones del formulario, del módulo o de la clase.
- Para tener esta opción activa para todo el código de la aplicación:
(Herramientas – Opciones... - Editor – Requerir declaración de variables)

Visual Basic

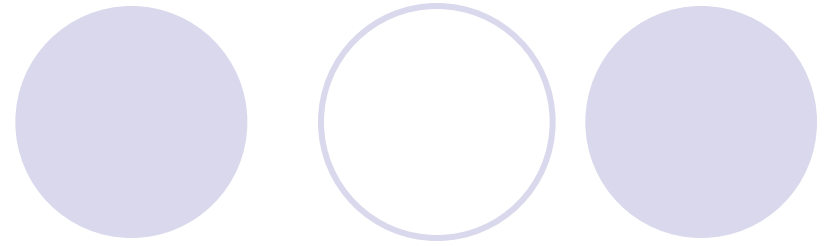
Declaración de Variables – Módulos de Visula Basic



- El código de una aplicación Visual Basic se edita en módulos. Los **módulos** tienen incorporadas funciones automáticas de formato y comprobación de sintaxis. Hay 3 tipos de módulos: de **formulario**, **estándar** y de **clase**. Para agregar uno de estos módulos en una aplicación, simplemente tenemos que ejecutar la orden correspondiente del menú **Proyecto**. Cada uno de ellos puede contener declaraciones y procedimientos. Un **formulario** (.frm) incluye **controles** más **código** y un **módulo** (.bas) o una **clase** (.cls) incluye sólo **código**.

Visual Basic

Declaración de Variables – Ámbito de las variables



- Se entiende por **ámbito** o alcance de una variable el espacio de la aplicación donde la variable es visible y por lo tanto se puede utilizar.

Ámbito	Declaración
Local	Dim , Static o ReDim (dentro de un procedimiento)
Módulo	Dim o Private (sección de declaraciones del módulo)
Global	Public (sección de declaraciones de un módulo)

- A nivel del **módulo**, no hay diferencias entre **Dim** y **Private**, pero se aconseja utilizar **Private** en contraste con **Public**. En un **procedimiento** no tiene lugar esta observación, ya que **no** se puede utilizar **Public**.

Visual Basic

Declaración de Variables – Variables Locales

- Una **variable local** se reconoce solamente en el **procedimiento** en el que está definida. Fuera de ese procedimiento, la variable no es conocida. Su utilización más común es intervenir en cálculos intermedios.

```
Private Sub Form_Load()  
    Dim ent1 As Integer, ent2 As Integer ' Definimos ent1 y ent2  
                                         ' como VARIABLES LOCALES con Dim.  
  
    ent1 = 40.17  
    ent2 = 37.83  
    Print ent1, ent2  
End Sub
```

- Una variable local es reiniciada cada vez que se entra en un procedimiento. Es decir, una variable local no conserva su valor entre una llamada al procedimiento y la siguiente. Para hacer que conserve su valor, hay que declarar la variable como **estática** (utilizar la palabra clave **Static** en lugar de **Dim**):

```
Private Sub Form_Load()  
    Static ent1 As Integer ' Definimos ent1 como VARIABLE  
                           ' LOCAL ESTATICA con Static.  
  
    Dim ent2 As Integer ' Definimos ent2 como VARIABLE  
                        ' LOCAL con Dim.  
  
    ent1 = 40.17  
    ent2 = 37.83  
    Print ent1, ent2  
End Sub
```

Visual Basic

Declaración de Variables – Variables Locales

- Para hacer que todas las variables de un procedimiento sean estáticas, podemos proceder declarando el procedimiento estático.

```
Private Static Sub Form_Load()  
    Dim ent1 As Integer, ent2 As Integer  
    ent1 = 40.17  
    ent2 = 37.83  
    Print ent1, ent2  
End Sub
```

' Declaramos todo el procedimiento
' como ESTATICO con Static.

- Si una variable aparece en un procedimiento y no está explícitamente declarada, es por omisión local.

Visual Basic

Declaración de Variables – Variables utilizadas dentro del módulo

- Una **variable declarada a nivel de módulo** (formulario, módulo estándar o clase) puede ser compartida por otros procedimientos de ese módulo. Una variable a nivel de módulo hay que declararla con **Dim** o **Private** en la sección de declaraciones del módulo (sección *General*). Para editar esta sección, hay que abrir la ventana de código del formulario, de un módulo estándar o de una clase, para lo que nos tenemos que dirigir al explorador de proyectos, seleccionar el módulo y hacer clic en el botón Ver código. Después en la ventana de código seleccionaremos “(General)”, lista de objetos y “(Declaraciones)”, lista de procedimientos.



- Este tipo de variables son por omisión, *estáticas*.

Visual Basic

Declaración de Variables – Variables Globales

- Una **variable global** es una variable declarada a nivel del módulo pero que puede ser accedida desde cualquier otro módulo. Para hacer que una variable sea global o pública, hay que declararla como **Public** en la sección de declaraciones del módulo. Para ello, si el módulo ya existe, lo seleccionaremos en la ventana *Proyecto* y haremos clic en el botón *Ver código*; y si no existe, lo crearemos ejecutando la orden correspondiente en el menú *Proyecto*.

```
Public Var1Global As Double, Var2Global As String
```

- Cuando una variable **Public**, por ejemplo *Conta*, se declara en un formulario, por ejemplo, *Form1*, para acceder a ella desde otro módulo es necesario especificar su pertenencia; esto es, de qué objeto es dato miembro dicha variable:

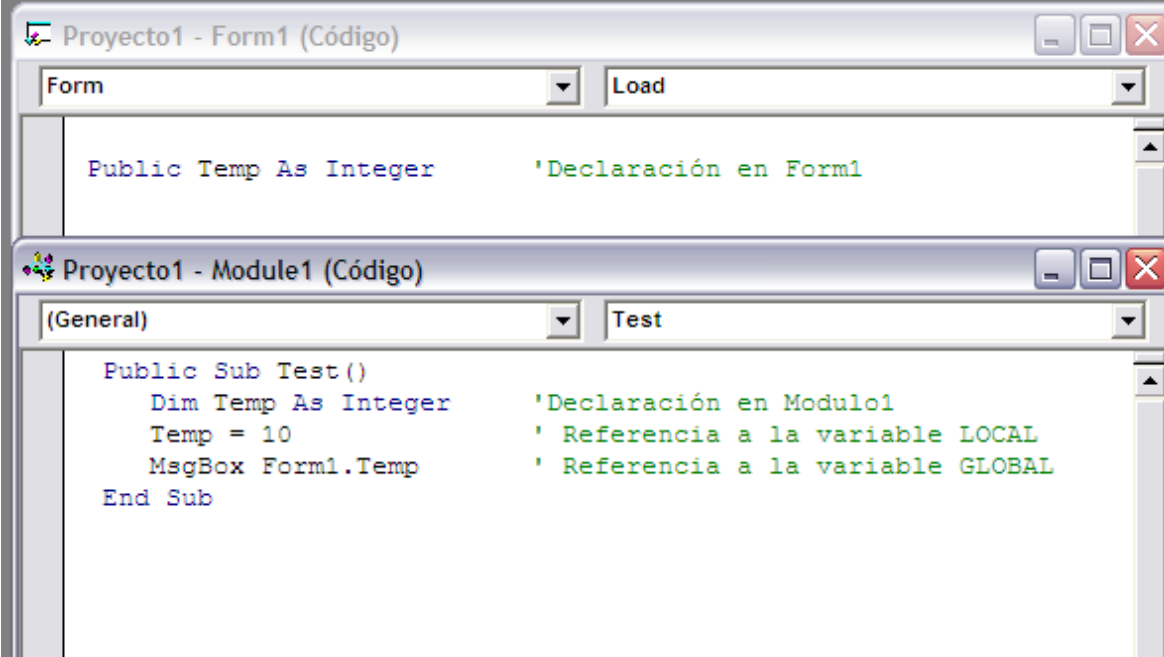
```
MsgBox Form1.Conta      'Visualiza el valor de Conta
```

- No se pueden declarar variables globales en un procedimiento.

Visual Basic

Declaración de Variables – 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):



The screenshot displays two overlapping code windows in Visual Basic. The top window, titled 'Proyecto1 - Form1 (Código)', shows a 'Form' dropdown and a 'Load' event. It contains the code: `Public Temp As Integer` followed by a green comment `'Declaración en Form1`. The bottom window, titled 'Proyecto1 - Module1 (Código)', shows a '(General)' dropdown and a 'Test' event. It contains a sub procedure `Public Sub Test()` with the following code: `Dim Temp As Integer` (commented as `'Declaración en Modulo1`), `Temp = 10` (commented as `' Referencia a la variable LOCAL`), and `MsgBox Form1.Temp` (commented as `' Referencia a la variable GLOBAL`). The `End Sub` statement follows.

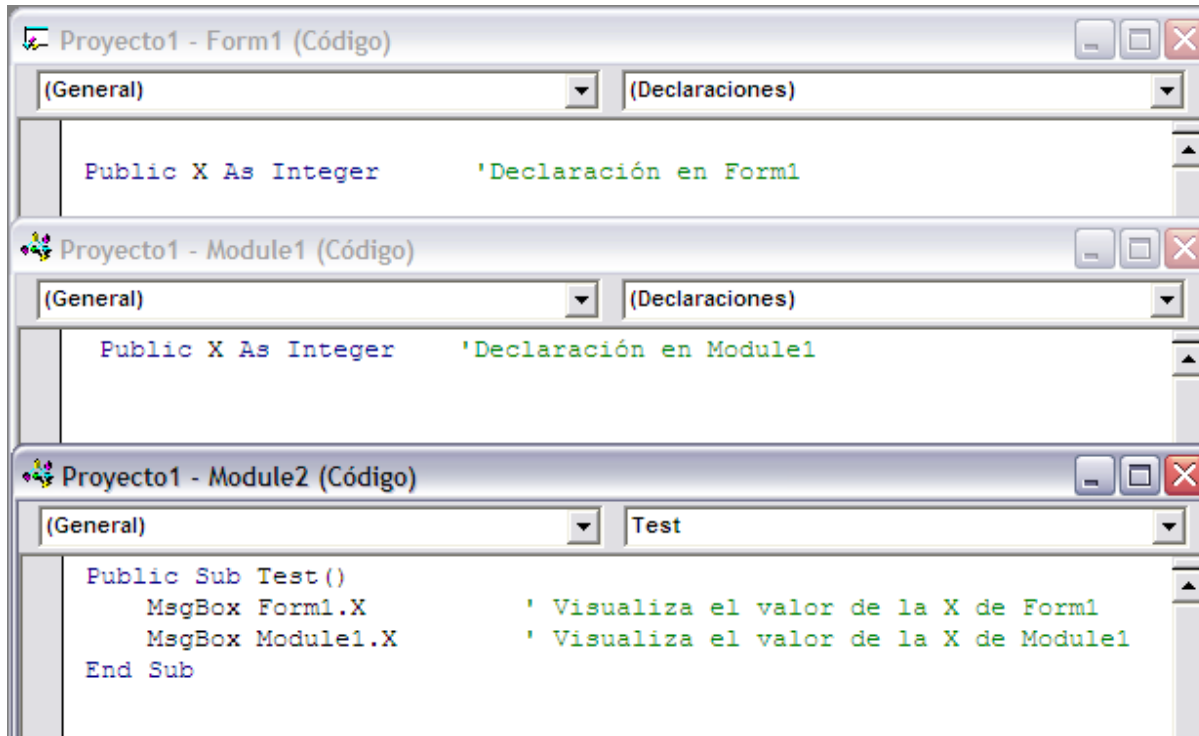
```
Proyecto1 - Form1 (Código)
Form Load
Public Temp As Integer      'Declaración en Form1

Proyecto1 - Module1 (Código)
(General) Test
Public Sub Test()
    Dim Temp As Integer     'Declaración en Modulo1
    Temp = 10               ' Referencia a la variable LOCAL
    MsgBox Form1.Temp       ' Referencia a la variable GLOBAL
End Sub
```

Visual Basic

Declaración de Variables – Variables con el mismo Nombre

- Si varias variables públicas comparten el mismo nombre en diferentes módulos, para diferenciarlas en el momento de referenciarlas es necesario especificar su pertenencia. Por ejemplo, si hay una variable entera X declarada en el módulo Form1 como en el módulo Module1, deberíamos referirnos a ella así:



```
Proyecto1 - Form1 (Código)
(General) (Declaraciones)
Public X As Integer 'Declaración en Form1

Proyecto1 - Module1 (Código)
(General) (Declaraciones)
Public X As Integer 'Declaración en Module1

Proyecto1 - Module2 (Código)
(General) Test
Public Sub Test()
    MsgBox Form1.X ' Visualiza el valor de la X de Form1
    MsgBox Module1.X ' Visualiza el valor de la X de Module1
End Sub
```

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.

Visual Basic

Ejercicio 00

- **Vamos a dibujar un formulario con dos botones. Cada vez que pulsemos el botón *Pulsar*, escribiremos unos valores en el formulario. Si pulsamos *Salir*, salimos de la aplicación (vamos a ver variables con el mismo nombre).**

The screenshot shows a Windows-style application window titled "Form1". Inside the window, there is a table with three columns and three rows of data. Below the table, there are two buttons: "Pulsar" and "Salir".

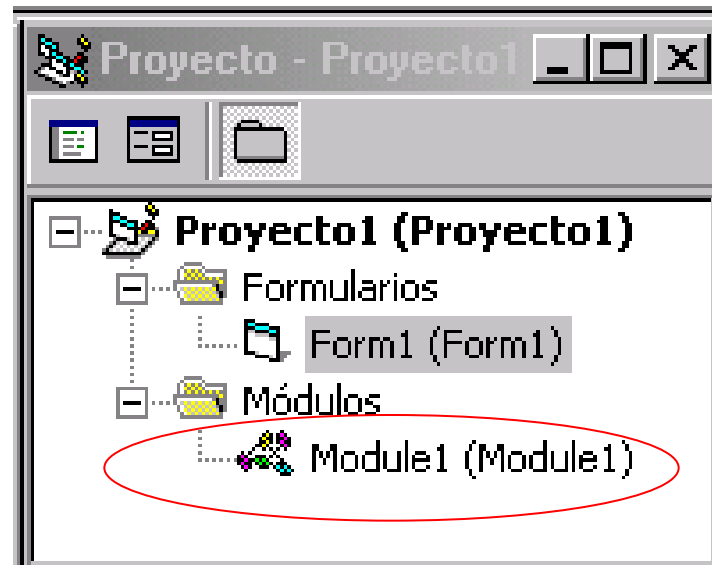
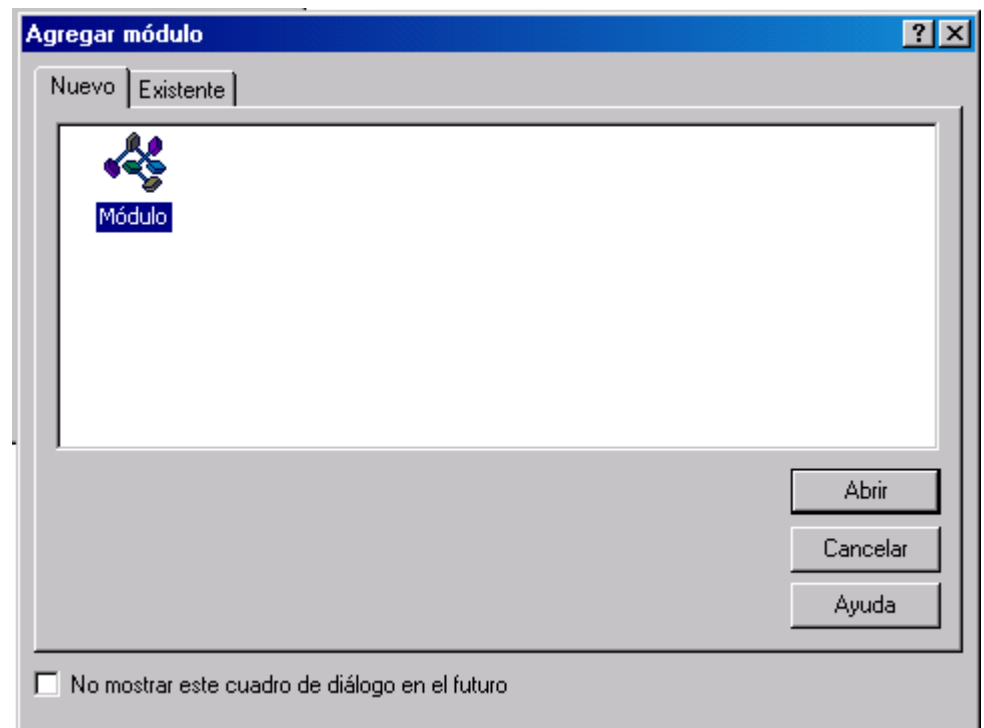
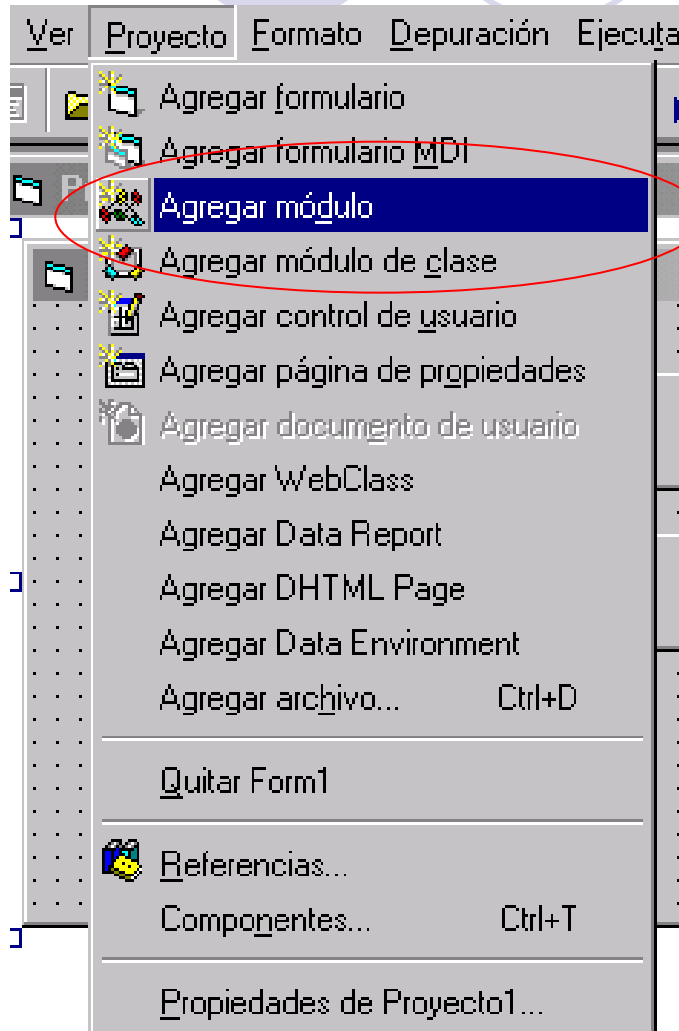
10	20	30
10	20	30
10	20	30

Pulsar

Salir

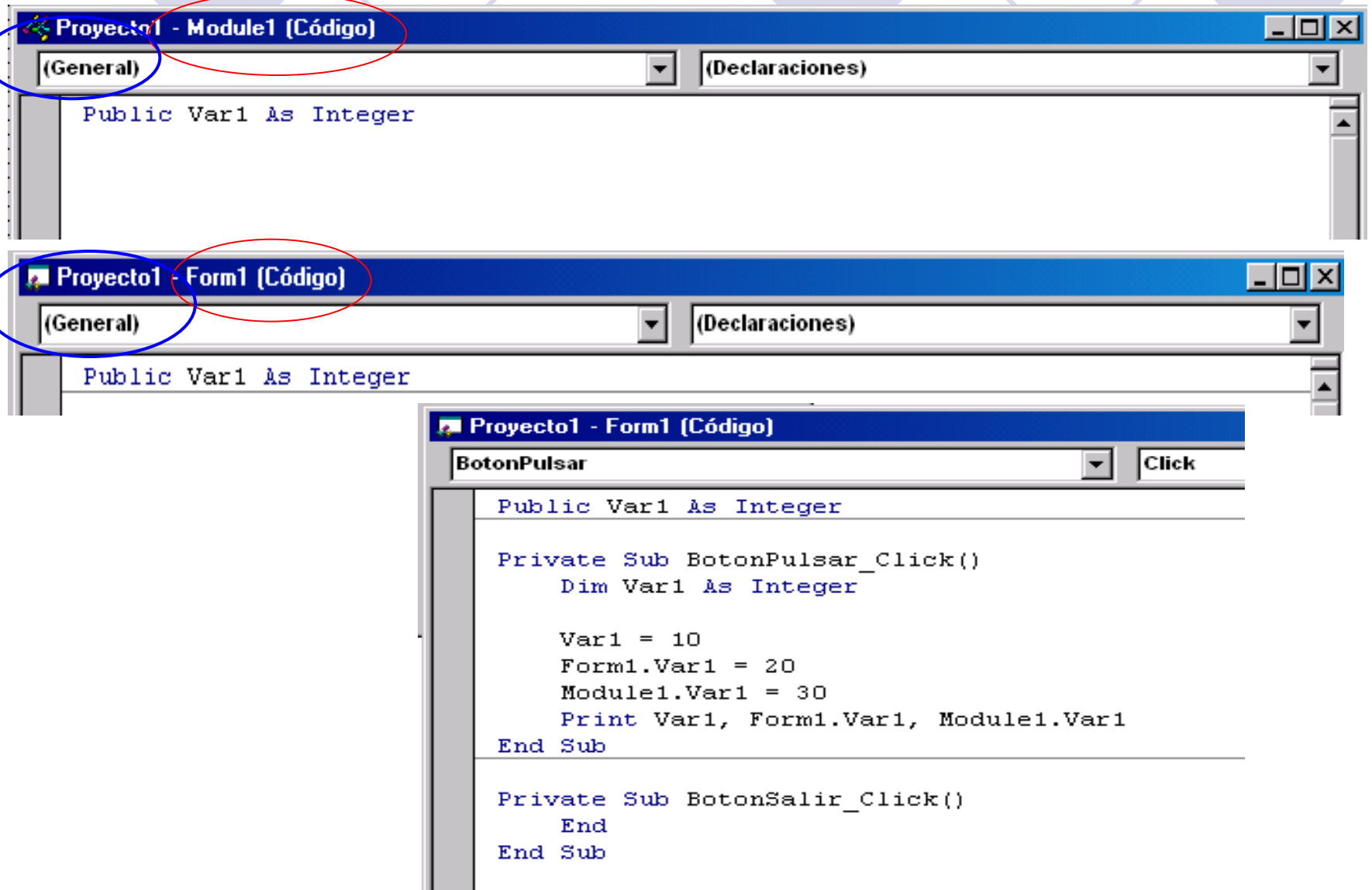
Visual Basic

Ejercicio 00



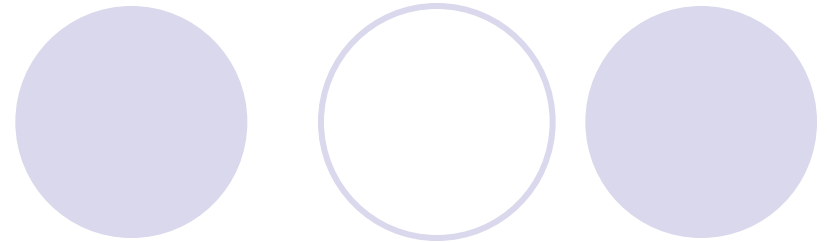
Visual Basic

Ejercicio 00



Visual Basic

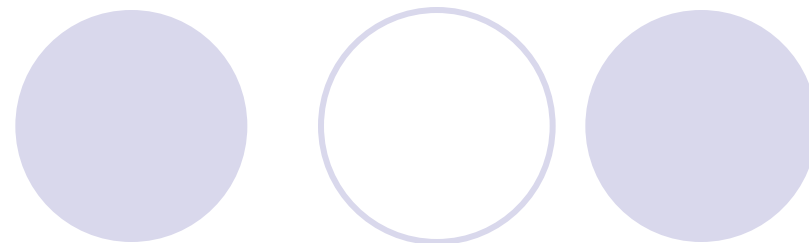
Declaración de Variables – Resumen



- Utilizaremos variables de ámbito global cuando éstas vayan a ser utilizadas desde cualquier punto de la aplicación, sin importar el módulo en el que esté el procedimiento que accede a ella.
- Sólo es posible declarar una variable global a nivel de módulo, es decir, fuera de cualquier procedimiento o función (**Public** variable *As* tipo).
- Si intentamos declarar una variable global en el interior de un procedimiento o una función, obtendremos un error durante la compilación de la aplicación.
- El tiempo de vida de una variable global comienza en el mismo momento en el que se carga la aplicación, y no muere hasta que ésta termina.
- Una variable local se crea automáticamente al entrar en el procedimiento o función en el que se declara, y al salir se destruye (pierde el valor asignado entre una llamada y otra).

Visual Basic

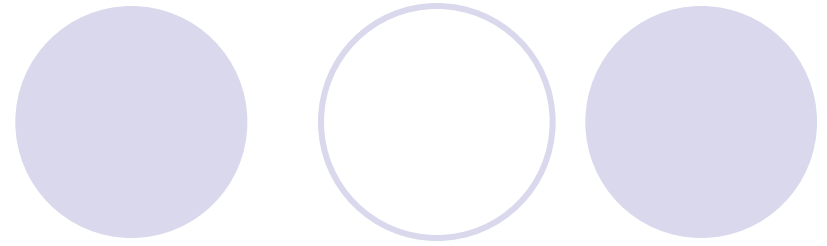
Declaración de Variables – Resumen



- El tiempo de vida de una variable estática (aún siendo local) comienza en el mismo momento en el que arranca la aplicación, y no muere hasta que ésta termina (**Static** variable *As* tipo).
- Por regla general, todas las variables declaradas en un procedimiento o función son locales no estáticas, a no ser que se especifique.
- En el caso de que todas las variables de nuestro procedimiento deban ser estáticas podemos anteponer la palabra **Static** delante de la definición de procedimiento o función (*Private Static Sub* Form_Load()).
- Un nombre de una variable debe comenzar siempre por una letra, incluso eñes y acentuadas.
- Puede estar seguido, de forma general, de hasta 39 caracteres más (letras, dígitos y el carácter de subrayado).
- No diferencia entre mayúsculas y minúsculas (num = NUM = Num = nUm = etc.).

Visual Basic

Declaración de Variables – Resumen



- Cuando definamos una variable, por ejemplo, en un procedimiento y la escribamos en el interior del procedimiento, Visual Basic escribirá el nombre de la variable “*tal como la definimos*”. Es más, si después de haber hecho referencia en varios puntos a la variable, cambiamos su definición, alterando sólo mayúsculas, minúsculas o acentos, veremos que en todos los puntos donde aparezca también será cambiada.
- Visual Basic permite añadir al final del nombre de una variable o de una constante, un carácter que hace referencia al tipo de dato (% , & , ! , # , @ , \$ - **Dim variable tipo_dato**) como declaración de la variable o constante.

Visual Basic

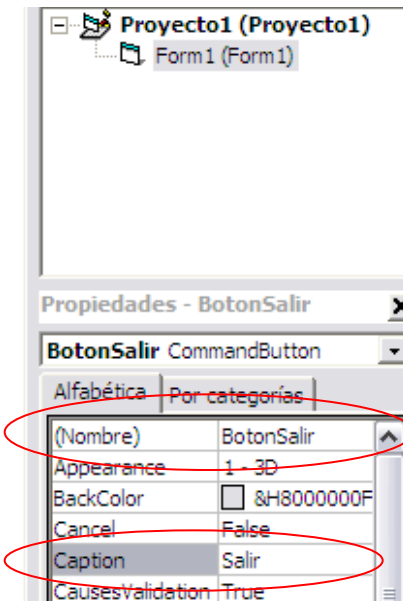
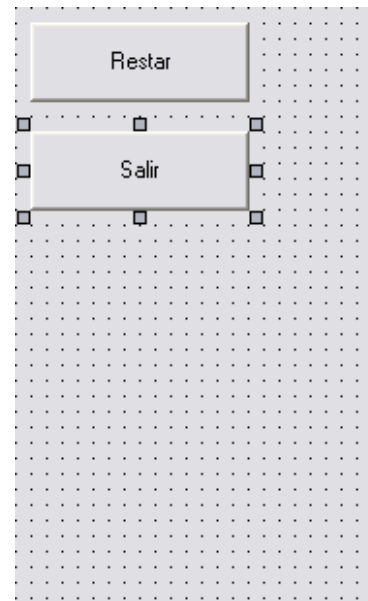
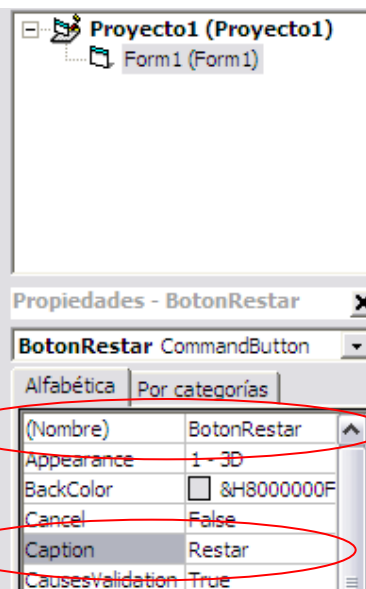
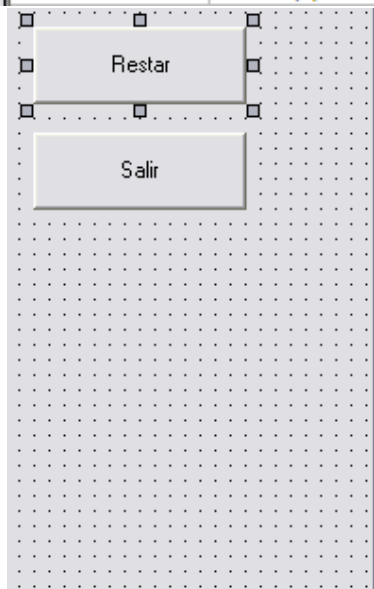
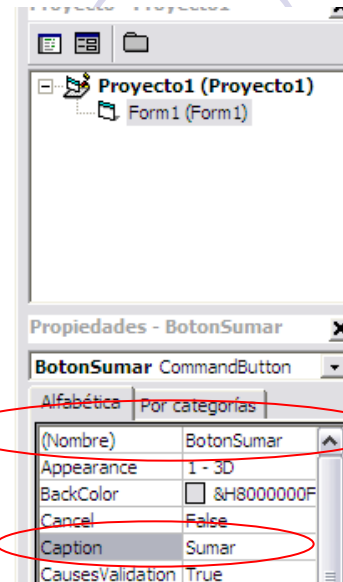
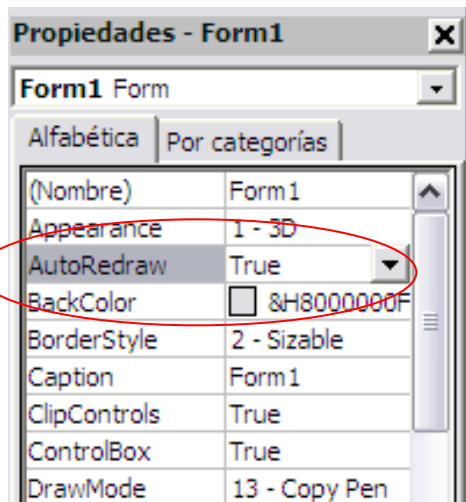
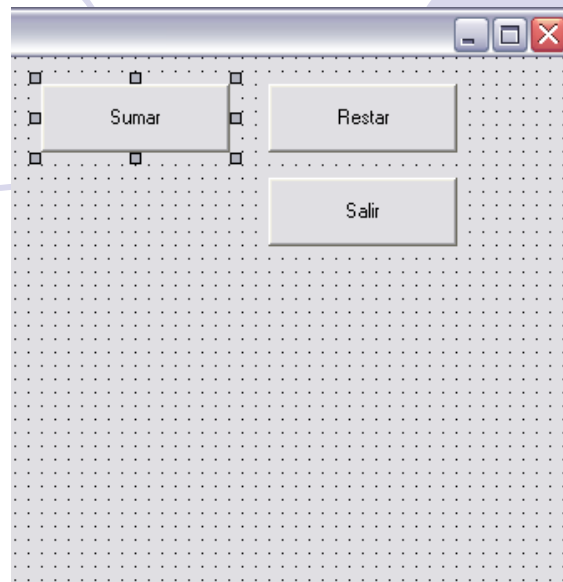
Ejercicio 1

- **Vamos a dibujar un formulario con tres botones. Cada vez que pulsemos el botón *Sumar*, sumaremos 1 a la variable “Número”. Cada que vez pulsemos el botón *Restar*, restaremos 1 a la variable “Número”.**

The image shows a screenshot of a Visual Basic form titled "Form1". On the left side, there is a vertical list of numbers: 1, 2, 3, 4, 5, 6, 7, 8, 7, 6, 5, 4, 3, 4, 5, 6. On the right side, there are three buttons arranged in a 2x2 grid. The top row contains the "Sumar" button and the "Restar" button. The bottom row contains an empty space and the "Salir" button. The "Sumar" button is highlighted with a dashed border.

Visual Basic

Ejercicio 1



Visual Basic

Ejercicio 1

Option Explicit

Dim Numero As Integer

```
Private Sub BotonSumar_Click()
```

```
    Numero = Numero + 1
```

```
    Print Numero
```

```
End Sub
```

```
Private Sub BotonRestar_Click()
```

```
    Numero = Numero - 1
```

```
    Print Numero
```

```
End Sub
```

```
Private Sub BotonSalir_Click()
```

```
    End
```

```
End Sub
```

Visual Basic

Ejercicio 1

- Vamos a introducir un cambio. Vamos a definir la variable “Numero” dentro de los procedimientos “BotonSumar” y BotonRestar”.

Option Explicit

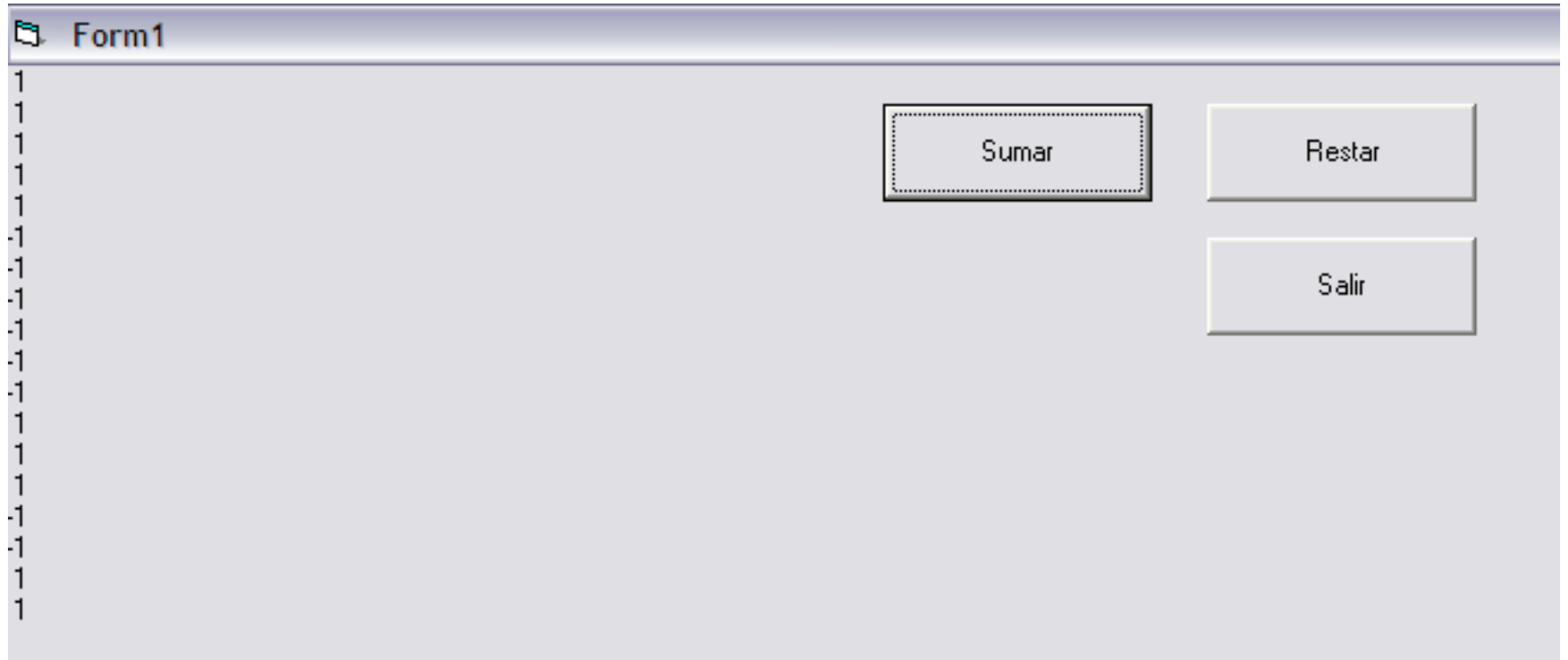
```
Private Sub BotonSumar_Click()  
    Dim Numero As Integer    ' Definimos la variable como LOCAL.  
    Numero = Numero + 1  
    Print Numero  
End Sub
```

```
Private Sub BotonRestar_Click()  
    Dim Numero As Integer    ' Definimos la variable como LOCAL.  
    Numero = Numero - 1  
    Print Numero  
End Sub
```

```
Private Sub BotonSalir_Click()  
    End  
End Sub
```

Visual Basic

Ejercicio 1



The image shows a screenshot of a Visual Basic form titled "Form1". The form has a light gray background and a standard Windows-style title bar. On the left side of the form, there is a vertical list of 15 small, identical icons, each consisting of a small square with a diagonal line. On the right side of the form, there are three buttons arranged in a vertical stack. The top button is labeled "Sumar", the middle button is labeled "Restar", and the bottom button is labeled "Salir". The "Sumar" button is highlighted with a thick black border, while the other two buttons have a standard gray border.

Visual Basic

Ejercicio 1

- Vamos a introducir otro cambio. Vamos a definir la variable “Numero” dentro de los procedimientos “BotonSumar” y BotonRestar” como **Estáticas**.

Option Explicit

```
Private Sub BotonSumar_Click()  
    Static Numero As Integer    ' Definimos la variable como local ESTATICA.  
    Numero = Numero + 1  
    Print Numero  
End Sub
```

```
Private Sub BotonRestar_Click()  
    Static Numero As Integer    ' Definimos la variable como local ESTATICA.  
    Numero = Numero - 1  
    Print Numero  
End Sub
```

```
Private Sub BotonSalir_Click()  
    End  
End Sub
```


Visual Basic

Ejercicio 1

Form1

1		
2		
3		
4		
5		
-1		
-2		
-3		
-4		
6		
7		
8		
9		
10		
-5		
-6		
-7		
11		
12		
13		

Sumar

Restar

Salir

Visual BasicEjercicio 11

- Vamos a dibujar un formulario con tres botones. Cada vez que pulsemos el botón **Sumar 5**, sumaremos 5 . Cada que vez pulsemos el botón **Restar 5**, restaremos 5. Con **Salir** salimos de la aplicación (utilizamos 3 variables distintas).

The screenshot shows a Visual Basic form titled 'Form1'. It contains a table with three columns of numbers and three buttons on the right side. The table has 16 rows of data. The buttons are 'Sumar 5', 'Restar 5', and 'Salir'.

5	5	5
10	10	5
15	15	5
20	20	5
15	-5	-5
10	-10	-5
5	-15	-5
0	-20	-5
5	25	5
10	30	5
15	35	5
20	40	5
15	-25	-5
10	-30	-5
5	-35	-5
0	-40	-5

Visual Basic

Ejercicio 11

(General)

Option Explicit

Dim Var1 As Integer

Private Sub BotonRestar_Click()

Static Var2 As Integer

Dim var3 As Integer

Var1 = Var1 - 5

Var2 = Var2 - 5

var3 = var3 - 5

Print Var1, Var2, var3

End Sub

Private Sub BotonSalir_Click()

End

End Sub

Private Sub BotonSumar_Click()

Static Var2 As Integer

Dim var3 As Integer

Var1 = Var1 + 5

Var2 = Var2 + 5

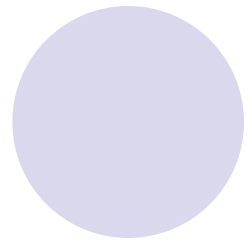
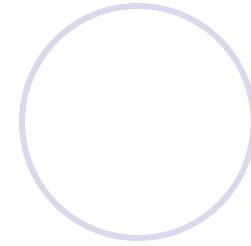
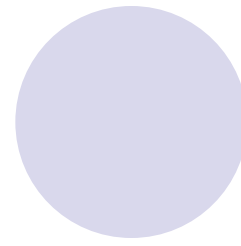
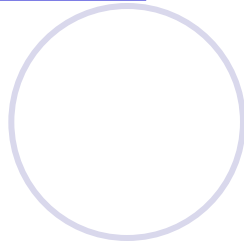
var3 = var3 + 5

Print Var1, Var2, var3

End Sub

Visual Basic

Constantes Simbólicas



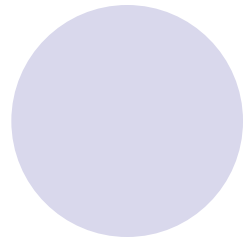
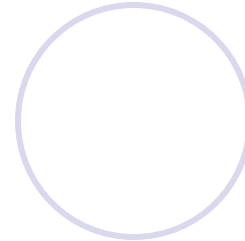
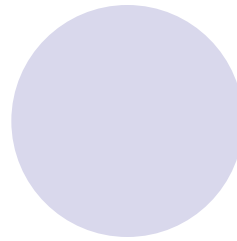
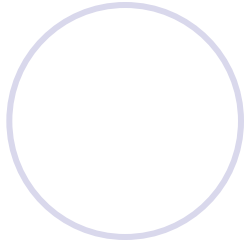
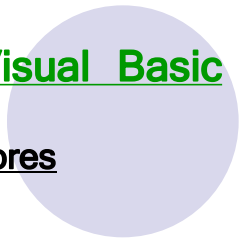
- Para definir una **constante simbólica** (valores constantes), se utiliza la siguiente sintaxis:
[Public|Private] Const *constante* [As *tipo*] = *expresión*
- Si no se declara explícitamente el tipo de constante (utilizando **As *tipo***), se asigna a la constante el tipo de datos más apropiado a 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 (siempre dentro de un módulo):

```
Public Const Max_Elementos = 99
Public Const Version = "Ver 4.05.0A"
Const PI = 3.1415926, Dos_PI = 2 * PI
Const Fecha_por_defecto = #1/1/2008#
```

- Es aconsejable definir todas las constantes globales en un único módulo.

Visual Basic

Operadores



- Lista de operadores ordenados de mayor a menor prioridad. Los operadores que aparecen en el mismo recuadro tienen la misma prioridad. Se evalúan primero las operaciones de los paréntesis más internos.

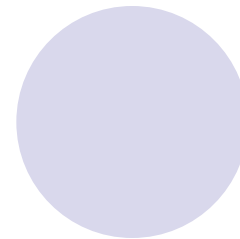
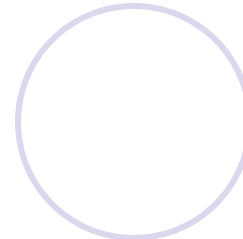
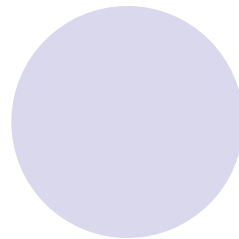
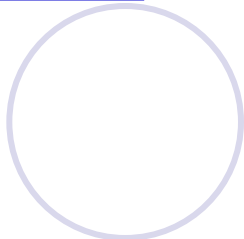
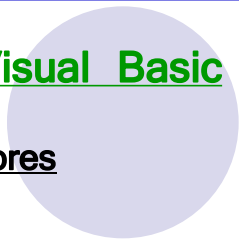
Visual Basic

Operadores

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

Visual Basic

Operadores

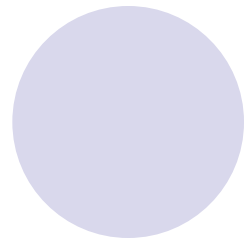
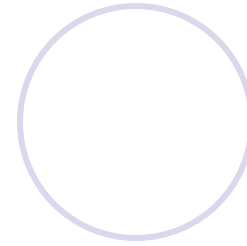
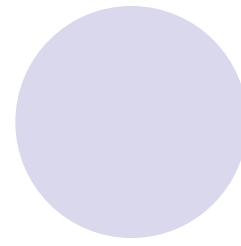
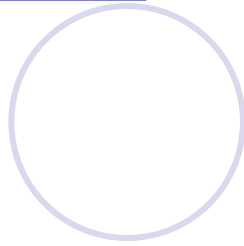


- N1 **operador** N2

Tipo	Operación	Operador
Aritmético	Eleva N1 al exponente indicado por N2	^
	Cambia de signo N1	-
	Multiplica N1 por N2 Divide N1 entre n2	*, /
	Divide N1 entre N2 sin hallar decimales	\
	Resto de la operación N1 \ N2	Mod
	Suma N1 y N2 Halla la diferencia entre N1 y N2	+, -

Visual Basic

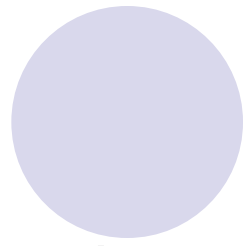
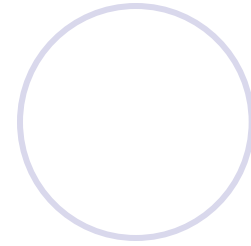
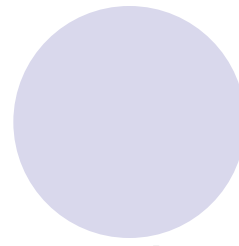
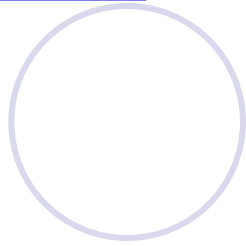
Operadores Aritméticos



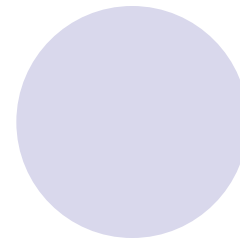
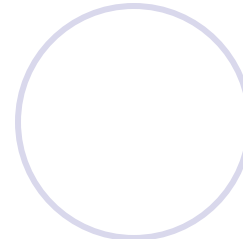
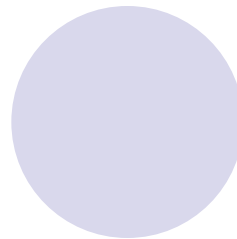
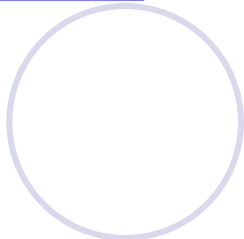
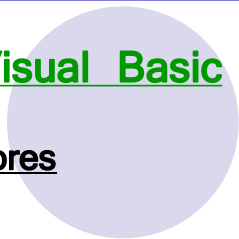
- Estos operadores, como es lógico, trabajan sobre cualquiera de los tipos numéricos que conocemos: enteros o coma flotante, independientemente de su precisión. Sin embargo, cuando en una misma expresión utilicemos operadores de distinto tipo numérico, Visual Basic realizará las conversiones necesarias para guardar la mayor precisión posible.
- Por medio del operador **+** podemos trabajar además con dos tipos de datos no numéricos. Con el podemos sumar un número con una fecha, obteniendo así una nueva fecha que vendrá determinada por los días transcurridos, especificados por el número, desde una fecha dada. También con **+** podemos sumar dos cadenas, tipo **String**, siendo en este caso el resultado la concatenación de ambas. Obtendremos, por tanto, una nueva cadena formada por la primera más la segunda. En Visual Basic existe un operador, **&**, que tiene precisamente la finalidad de concatenar dos datos formando una cadena, siendo preferente el uso de éste operador sobre el **+**.

Visual Basic

Operadores Aritméticos



- Al igual que podemos sumar un número de días a una fecha también podemos restarlo, utilizando el operador **-**. Con él también podemos hallar el número de días transcurridos entre dos fechas dadas.
- Cuando realicemos operaciones aritméticas con el tipo de dato **Variant**, el resultado vendrá dado por el tipo de dato que en ese momento contenga la variable, que puede ser numérico, de cadena, fecha u otro tipo. En cada caso se actuará de la misma forma que si la variable fuese del tipo correspondiente al valor que contiene en ese momento, por lo que en unos casos dos variables **Variant** pueden producir un valor numérico, al realizar una suma, mientras que en otras podemos obtener una cadena, si una de las dos variables contiene una cadena.

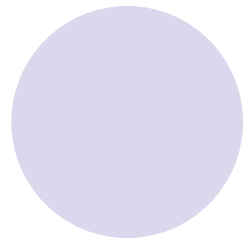
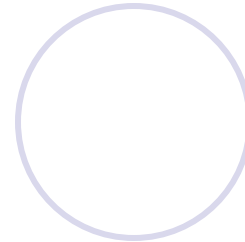
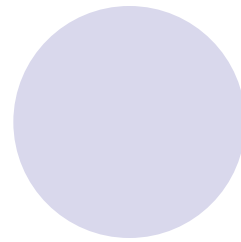
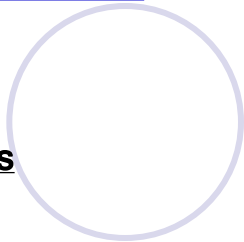
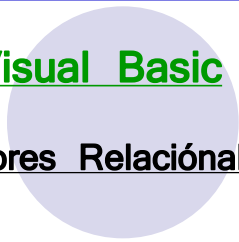


- N1 **operador** N2

Tipo	Operador	Devuelve True si	Devuelve False si
Relacional	=	N1 y N2 son iguales	N1 y N2 son distintos
	<>	N1 y N2 son distintos	N1 y N2 son iguales
	<	N1 es menor que N2	N1 es igual o mayor que N2
	<=	N1 es menor o igual que N2	N1 es mayor que N2
	>	N1 es mayor que N2	N1 es igual o menor que N2
	>=	N1 es mayor o igual que N2	N1 es menor que N2

Visual Basic

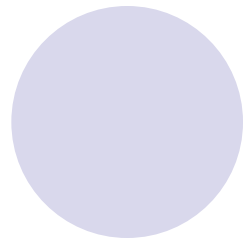
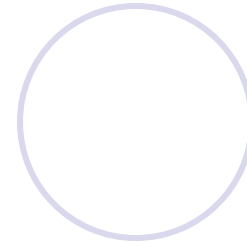
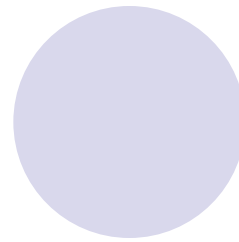
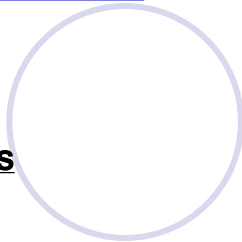
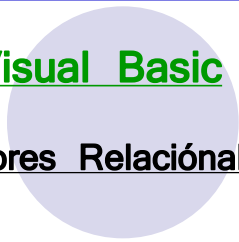
Operadores Relacionales



- Este conjunto de operadores nos permitirán evaluar expresiones y obtener dos resultados posibles: *True*, si la expresión es cierta o devuelve un valor distinto de cero, o *False*, si la expresión es falsa o devuelve cero. Se utilizan principalmente en estructuras de decisión, en las que, dependiendo del valor obtenido, se toma un camino u otro.
- Por medio de estos operadores podemos comparar números, cadenas y fechas. En el caso de los números, será su valor el que intervenga directamente en la relación.
- Si comparamos dos cadenas, en realidad se comparan los códigos de cada uno de los caracteres que las componen, de tal forma que “Mañana” no será igual a “mañana”, ya que el código de la “M” no es el mismo que el de la “m”. Al trabajar con dos fechas se utilizará la representación interna de las variables tipo **Date**, para saber si una fecha es igual, anterior o posterior a otra. Hay que tener en cuenta, que una variable del tipo **Date**, además de la fecha también puede contener la hora, por lo que al realizar una comparación puede que no obtengamos el resultado esperado.

Visual Basic

Operadores Relacionales



- Por ejemplo, dos variables Date pueden contener la fecha del día de hoy, pero si una tiene una hora y la otra no, o tiene una hora distinta, las variables no cumplirán la relación de igualdad.
- Es posible utilizar en la misma expresión operadores aritméticos y relacionales, caso éste en el que antes de que se evalúe una expresión relacional, previamente se realizarán los cálculos aritméticos necesarios para que los operandos que se van a comparar sean sólo dos.

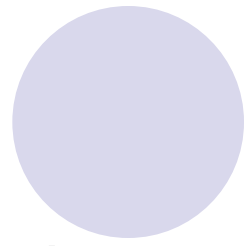
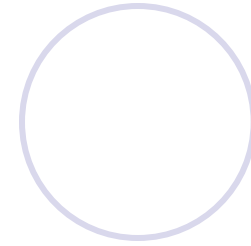
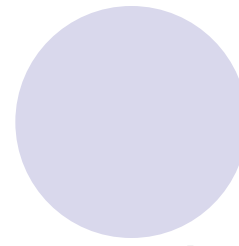
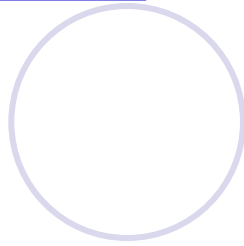
Visual BasicOperadores

- N1 **operador** N2

Tipo	Operador	Devuelve True si	Devuelve False si
Lógico	Not	N1 = False	N1 = True
	And	N1 = True y N2 = True	En cualquier otro caso
	Or	N1 = True o E2 = True	N1 = False y N2 = False
	Xor	N1 = True y N2 = False o N1 = False y N2 = True	N1 = True y N2 = True o N1 = False y N2 = False
	Eqv	N1 es igual a N2	N1 es distinto de N2
	Imp	N1 = True y N2 = True o N1 = False y N2 = True o N1 = False y N2 = False	N1 = True y N2 = False

Visual Basic

Operadores Lógicos



- En ocasiones, una expresión relacional no está compuesta tan sólo de dos operandos y un operador de comparación, sino que son necesarios varios operadores y más operandos. En estos casos hay que unir las distintas subexpresiones por medio de algún elemento, que fije a su vez unas reglas entre cada una de las relaciones.
- Entre otras, ésta es la finalidad de los operadores lógicos. Los operadores lógicos, a excepción del operador **Not**, toma dos expresiones, normalmente relacionales, que, como sabemos, pueden devolver cada una de ellas *True* o *False*. Con esta pareja de valores, el operador realiza otra operación, en este caso lógica, para obtener un solo valor.
- El operador **Not** sólo toma un parámetro, y lo que hace es devolver el valor inverso al devuelto por la expresión.

Visual Basic

Ejercicio 2

- Mostrar la representación binaria (utilizando 8 bits) de los números del 0 al 255.

Form1								
0 ->	00000000	00000001	00000010	00000011	00000100	00000101	00000110	00000111
8 ->	00001000	00001001	00001010	00001011	00001100	00001101	00001110	00001111
16 ->	00010000	00010001	00010010	00010011	00010100	00010101	00010110	00010111
24 ->	00011000	00011001	00011010	00011011	00011100	00011101	00011110	00011111
32 ->	00100000	00100001	00100010	00100011	00100100	00100101	00100110	00100111
40 ->	00101000	00101001	00101010	00101011	00101100	00101101	00101110	00101111
48 ->	00110000	00110001	00110010	00110011	00110100	00110101	00110110	00110111
56 ->	00111000	00111001	00111010	00111011	00111100	00111101	00111110	00111111
64 ->	01000000	01000001	01000010	01000011	01000100	01000101	01000110	01000111
72 ->	01001000	01001001	01001010	01001011	01001100	01001101	01001110	01001111
80 ->	01010000	01010001	01010010	01010011	01010100	01010101	01010110	01010111
88 ->	01011000	01011001	01011010	01011011	01011100	01011101	01011110	01011111
96 ->	01100000	01100001	01100010	01100011	01100100	01100101	01100110	01100111
104 ->	01101000	01101001	01101010	01101011	01101100	01101101	01101110	01101111
112 ->	01110000	01110001	01110010	01110011	01110100	01110101	01110110	01110111
120 ->	01111000	01111001	01111010	01111011	01111100	01111101	01111110	01111111
128 ->	10000000	10000001	10000010	10000011	10000100	10000101	10000110	10000111
136 ->	10001000	10001001	10001010	10001011	10001100	10001101	10001110	10001111
144 ->	10010000	10010001	10010010	10010011	10010100	10010101	10010110	10010111
152 ->	10011000	10011001	10011010	10011011	10011100	10011101	10011110	10011111
160 ->	10100000	10100001	10100010	10100011	10100100	10100101	10100110	10100111
168 ->	10101000	10101001	10101010	10101011	10101100	10101101	10101110	10101111
176 ->	10110000	10110001	10110010	10110011	10110100	10110101	10110110	10110111
184 ->	10111000	10111001	10111010	10111011	10111100	10111101	10111110	10111111
192 ->	11000000	11000001	11000010	11000011	11000100	11000101	11000110	11000111
200 ->	11001000	11001001	11001010	11001011	11001100	11001101	11001110	11001111
208 ->	11010000	11010001	11010010	11010011	11010100	11010101	11010110	11010111
216 ->	11011000	11011001	11011010	11011011	11011100	11011101	11011110	11011111
224 ->	11100000	11100001	11100010	11100011	11100100	11100101	11100110	11100111
232 ->	11101000	11101001	11101010	11101011	11101100	11101101	11101110	11101111
240 ->	11110000	11110001	11110010	11110011	11110100	11110101	11110110	11110111
248 ->	11111000	11111001	11111010	11111011	11111100	11111101	11111110	11111111

Visual Basic

Ejercicio 2

Form1 Form

Alfabética | Por categorías

Palette	(Ninguno)
PaletteMode	0 - Halftone
Picture	(Ninguno)
RightToLeft	False
ScaleHeight	3090
ScaleLeft	0
ScaleMode	1 - Twip
ScaleTop	0
ScaleWidth	4680
ShowInTaskbar	True
StartPosition	3 - Windows Def
Tag	
Top	0
Visible	True
WhatsThisButton	False
WhatsThisHelp	False
Width	4800
WindowState	2 - Maximized

Propiedades - Form1

Form1 Form

Alfabética | Por categorías

(Nombre)	Form1
Appearance	1 - 3D
AutoRedraw	True
BackColor	<input type="checkbox"/> &H8000000F

Visual Basic

Ejercicio 2

Option Explicit

Private Sub Form_Click()

Dim I As Integer, J As Integer, K As Integer

For I = 0 To 31 ' 32 Filas.

Print I * 8; " -> ", ' Primer número de la Fila.

' Ir al próximo punto de tabulación.

For K = 0 To 7 ' por 8 Columnas.

For J = 7 To 0 Step -1

' Cada número tiene 8 bits.

' $I * 8 + K$ es el número a comprobar, de 0 a 255.

' 2^J nos dará un número cuyo bit J esté a 1

' y los demás a 0.

' El resultado de la expresión será VERDADERO

' si dicho bit está a 1 en $I * 8 + K$, y FALSO

' en caso contrario.

If $I * 8 + K$ And 2^J Then

Print "1";

Else

Print "0";

End If

Next

Print , ' Ir al próximo punto de tabulación.

Next

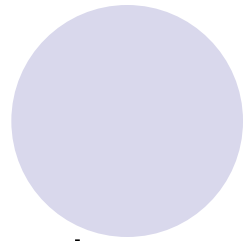
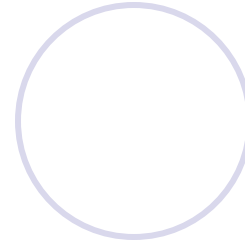
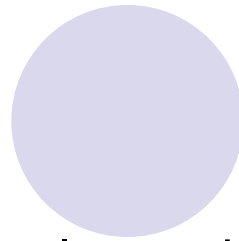
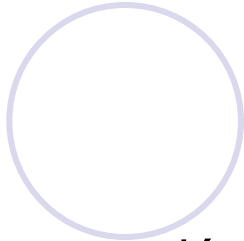
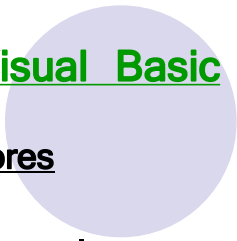
Print ' Ir a la siguiente línea.

Next

End Sub

Visual Basic

Operadores

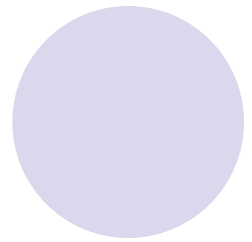
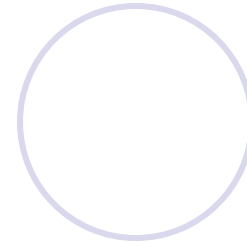
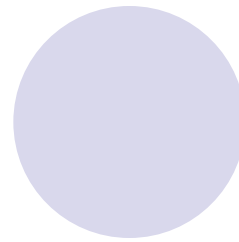
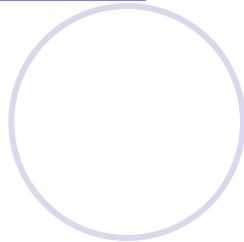
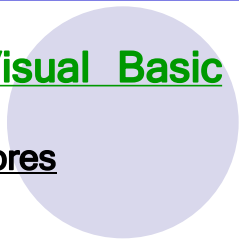


- Si al evaluar una expresión sucede que alguno de los operandos tiene un valor **Null**, el resultado es **Null**.
- Cuando en una expresión aritmética intervienen operandos de diferentes tipos, el resultado se expresa, generalmente, en la precisión del operando que la tiene más alta. El orden, de menor a mayor, según la precisión, es: **Integer**, **Long**, **Single**, **Double** y **Currency**.
- Los operadores relacionales comparan dos expresiones dando como 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, es preferible utilizar **&**. El resultado es de tipo **String** si ambas expresiones son de tipo **String**, en otro caso, el resultado es **Variant**.
- Los operadores lógicos podemos utilizarlos de dos formas: para obtener un resultado de tipo **Boolean** (True o False) una vez evaluadas las dos expresiones a **True** o **False**, o para realizar una operación lógica bit a bit entre dos expresiones numéricas, colocando el resultado en la variable que se especifique.

Visual BasicOperadores

- Cuando otros tipos de datos numéricos se convierten a **Boolean**, **0** (cero) pasa a ser **False**, mientras que todos los demás valores pasan a ser **True**. Cuando los valores **Boolean** se convierten a otros tipos, **False** pasa a ser **0**, mientras que **True** se convierte en **-1**.
- El operador **Like** se utiliza para comparar dos cadenas de caracteres. Su sintaxis es:
[resultado=] **expresión Like patrón**
- El resultado será **True** si la **expresión** coincide con alguna de las definidas en 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**.

Caracteres comodín del patrón	Se empareja en la expresión con
?	Un solo carácter
*	Cero o más caracteres
#	Un solo dígito (0 a 9)
[lista_caracteres]	Un solo carácter de los pertenecientes a la lista
[!lista_caracteres]	Un solo carácter de los no pertenecientes a la lista



- **Option Compare Binary** efectúa comparaciones de cadenas en función de un criterio de ordenación que se deriva de las representaciones binarias internas de los caracteres. El criterio de ordenación viene determinado por la página de códigos. En el ejemplo siguiente se muestra un criterio de ordenación binario típico.

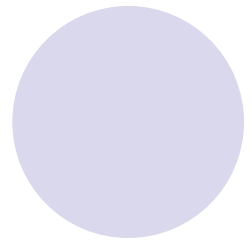
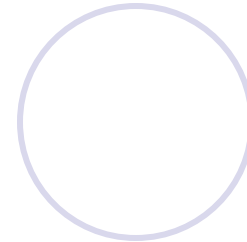
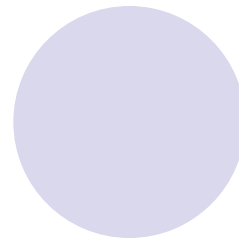
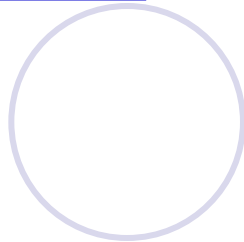
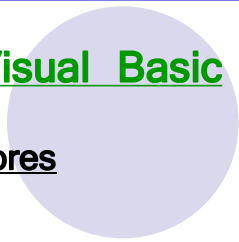
$A < B < E < Z < a < b < e < z < \grave{A} < \hat{E} < \emptyset < \grave{a} < \hat{e} < \emptyset$

- **Option Compare Text** efectúa comparaciones de cadenas en función de un criterio de ordenación textual que no distingue entre mayúsculas y minúsculas y que viene determinado por la configuración regional del sistema. Al establecer **Option Compare Text** y ordenar los caracteres en el ejemplo anterior, se aplica el criterio de ordenación de texto siguiente:

$(A=a) < (\grave{A}=\grave{a}) < (B=b) < (E=e) < (\hat{E}=\hat{e}) < (\emptyset=\emptyset) < (Z=z)$

Visual Basic

Operadores



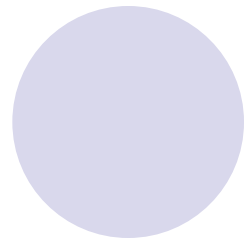
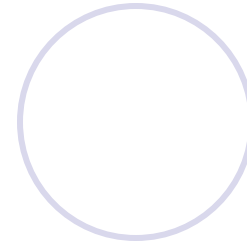
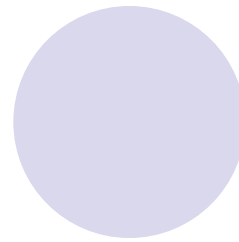
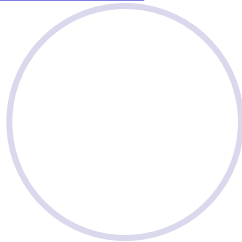
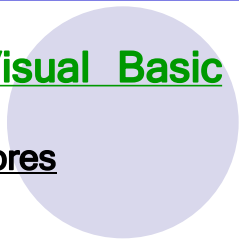
- En el siguiente ejemplo el resultado de la comparación será **True** cuando la cadena de caracteres *Cadena* coincida con una cadena de 3 caracteres, en la que el primero sea una 'a', el segundo, cualquier carácter que no sea 'i', 'j', 'k', 'l', 'm' o 'n', y el tercero un dígito de 0 a 9:

Dim Cadena As String

```
Private Sub Form_Load()  
    If Cadena Like "a[!i-n]#" Then  
        ' Acciones a ejecutar si el resultado es TRUE  
    End If  
End Sub
```

Visual Basic

Operadores



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

```
Dim A As Form           'Referencia la 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 dos **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).

Visual Basic

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 (:).

total = cantidad * precio: suma = suma + total

- 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):

```
Private Sub Form_Load()  
PagoMensual = CantidadPrest * _  
    (Interes / (1 - (1 / ((1 + Interes) ^ Meses))))  
End Sub
```

- La sentencia más común en Visual Basic es la sentencia de asignación:

variable = expresión

que indica que el valor que resulta de evaluar la ***expresión*** tiene que ser almacenado en la ***variable*** especificada.

Visual Basic

Sentencias

```
Dim Cont As Integer
Dim Interes As Double, Capital As Double
Dim TantoPorCiento As Single
Dim Mensaje As String
```

```
Cont = Cont + 1
Interes = Capital * TantoPorCiento / 100
Mensaje = "La Operación es correcta"
```

- Además, Visual Basic aporta otras muchas sentencias, que según su funcionalidad, se agrupan de la forma siguiente:

Declaración de variables y constantes	Manipulación de la fecha y de la hora
Control de flujo del programa	Manipulación de errores
Manipulación de matrices	Realización de gráficos
Entrada/Salida	Sentencias varias
Manipulación de procedimientos y funciones	

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

- El tipo de datos **Variant** permite almacenar todos los tipos de datos definidos en el sistema (enteros, fraccionarios en coma fija o variable, cadenas de caracteres, o datos relativos a la fecha y a la hora). Esto le define como un tipo de datos **genérico**. Cuando Visual Basic opera con tipos de datos **Variant**, **ejecuta automáticamente** las conversiones necesarias. No obstante, hay que tener en cuenta las siguientes consideraciones:
 - Cuando se ejecutan operaciones aritméticas con variables de tipo **Variant**, su contenido tiene que poderse convertir a un valor numérico.
 - Cuando se ejecutan operaciones aritméticas sobre una variable **Variant** que contiene un **Byte**, un **Integer**, un **Long** o un **Single** y el resultado excede del rango del tipo original, el tipo de la variable pasa a ser el necesario para contener el resultado.
 - Como el operador **+** puede ser utilizado para sumar valores y concatenar cadenas de caracteres, dependiendo del contenido de las variables, los resultados pueden ser inesperados. Por ello, para concatenar variables de tipo **Variant**, y en general para evitar ambigüedades, utilizaremos el operador **&** en lugar del operador **+**. El operador **&** ejecuta la concatenación sin tener en consideración qué contienen las variables.

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

- Cuando asignamos un valor numérico a una variable **Variant**, Visual Basic utiliza la representación más compacta para registrar el valor:

```
Dim Dato                ' Variable de tipo Variant por omisión.  
  
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). Si queremos conocer el tipo de dato que almacena una variable **Variant**, utilizaremos 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**.

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

```
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 esta iniciada (Empty).
            MsgBox "Vacio"
        Case 1      ' Si Tipo es 1, la variable Dato no tiene datos válidos (Null).
            MsgBox "Null"
        Case 2      ' Si Tipo es 2, la variable Dato se corresponde con un Entero.
            MsgBox "Entero: " & Dato
        Case 3      ' Si Tipo es 3, la variable Dato se corresponde con un Long.
            MsgBox "Long: " & Dato
        Case 4      ' Si Tipo es 4, la variable Dato se corresponde con un Single.
            MsgBox "Single: " & Dato
        Case 5      ' Si Tipo es 5, la variable Dato se corresponde con un Double.
            MsgBox "Double: " & Dato
        Case 6      ' Si Tipo es 6, la variable Dato se corresponde con un Currency.
            MsgBox "Currency: " & Dato
        Case 7      ' Si Tipo es 7, la variable Dato se corresponde con la Fecha/Hora.
            MsgBox "Fecha/Hora: " & Dato
        Case 8      ' Si Tipo es 8, la variable Dato se corresponde con una String.
            MsgBox "String: " & Dato
        Case 11     ' Si Tipo es 11, la variable Dato se corresponde con un Boolean.
            MsgBox "Boolean: " & Dato
        Case 17     ' Si Tipo es 17, la variable Dato se corresponde con un Byte.
            MsgBox "Byte: " & Dato
    End Select
End Sub
```

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

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

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

- En este ejemplo esperábamos resultado entero y no ha sido así. Para que así sea, debemos utilizar la función **CInt**, para convertir la variable **Variant** a un valor entero:

```
Dato = CInt(Dato) + 1111 'Valor Entero
```

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

- Las variable **Variant** también puede contener valores de tipo **Date** y, con ellos, puede ejecutar operaciones aritméticas y comparaciones:

```
Dim Dato                ' Variant por omisión.  
Dato = Now              ' Contenido: dd/mm/aa hh:mm:ss.  
Dato = Dato - 1         ' Menos 1 día.  
If Dato > #10/24/1999 3:00:00 AM# 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 de diversos formatos de fecha/hora, incluidos entre #:

```
Dim Dato ' Variant por omisión.  
Dato = #18-4-99 15:30#  
Dato = #18 Apr. 1999 3:30pm#  
Dato = #18-Apr-99#  
Dato = #18 April 1999#
```

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

- Para verificar si un determinado valor puede considerarse como uno del tipo **Date** (fecha/hora), debemos utilizar la función **IsDate**. Para convertir dicho valor en uno del tipo **Date**, debemos utilizar la función **CDate**. Por ejemplo, supongamos que el contenido de una caja de texto *Text1* es: 18 4 99. Las siguientes sentencias:

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

almacenan en la variable *Dato* el valor 18/04/99. Una alternativa a la función **CDate** es **Date Value**.

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

- Para saber si una variable **Variant** no ha sido iniciada, esto es, si está vacía, utilizaremos la función **IsEmpty**. El valor **Empty** (vacío) denota de una variable **Variant** que no ha sido iniciada (no se le ha asignado un valor inicial). Una variable **Variant** que contiene el valor **Empty** es 0 si se usa en un contexto numérico o una cadena de longitud cero ("") si se usa un contexto de cadenas.
- Para indicar que una variable **Variant** contiene un dato no válido, se utiliza el valor especial **Null**. Evidentemente **Null** no es lo mismo que **Empty**.
- Cualquier operación entre expresiones que contenga **Null** da como resultado **Null**. Esto puede ser útil cuando en las expresiones intervienen funciones escritas para que retornen **Null** cuando ocurra un error.

Visual Basic

Utilización del tipo Variant (Ampliación y recordatorio)

- Para verificar si el resultado final de una expresión es **Null**, debemos utilizar la función **IsNull**:

```
Dim R As Integer, S As Integer, T As Integer
Dim X As Variant

X = FDiv(R, S) + T ' Si ocurre un ERROR, FDiv retorna Null.
If IsNull(X) Then
    MsgBox "Dato NO válido."
Else
    X = CDb1(X)
End If
```

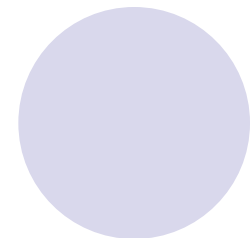
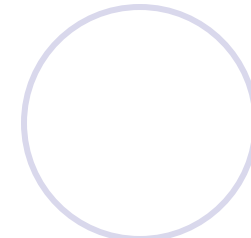
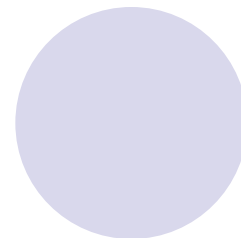
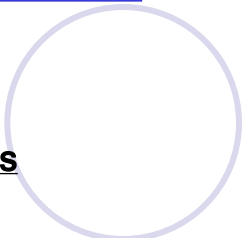
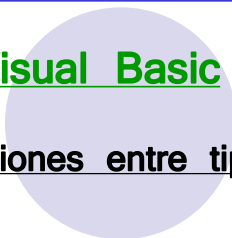
donde la función *FDiv* es de la forma:

```
Function FDiv(N As Integer, D As Integer) As Variant
    If D = 0 Then
        FDiv = Null
    Else
        FDiv = N / D
    End If
End Function
```

- El resultado devuelto por la función *FDiv* es de tipo **Variant**.

Visual Basic

Conversiones entre tipos



- En ocasiones ocurre que un dato que tenemos almacenado en una variable de un determinado tipo, lo necesitamos para realizar una operación en la que el tipo debe ser distinto.
- No siempre es posible realizar correctamente la conversión, caso en el que se genera un error durante la ejecución del programa. Esto ocurre, por ejemplo, cuando al realizar una conversión a un tipo numérico, éste se ve desbordado, o el tipo de dato original no contiene un número.

Visual Basic

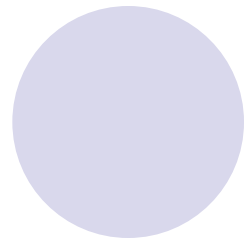
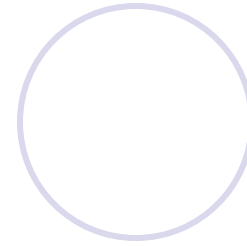
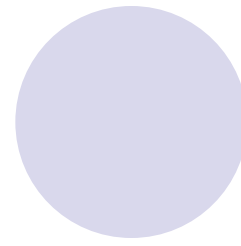
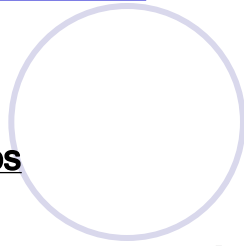
Conversiones entre tipos

Funciones para conversión de Datos

Función	Tipo que devuelve
CBool (<i>expresión</i>)	conversión a un valor de tipo Boolean
CByte (<i>expresión</i>)	conversión a un valor de tipo Byte
CCur (<i>expresión</i>)	conversión a un valor de tipo Currency
CDate (<i>expresión</i>)	conversión a un valor de tipo Date
CDbl (<i>expresión</i>)	conversión a un valor de tipo Double
CInt (<i>expresión</i>)	conversión a un valor de tipo Integer
CLng (<i>expresión</i>)	conversión a un valor de tipo Long
CSng (<i>expresión</i>)	conversión a un valor de tipo Single
CStr (<i>expresión</i>)	conversión a un valor de tipo String
CVar (<i>expresión</i>)	conversión a un valor de tipo Variant
CErr (<i>expresión</i>)	conversión a un valor de tipo Error

Visual Basic

Conversiones entre tipos



- **CInt** realiza una conversión a un número entero realizando un redondeo, no simplemente cortando la parte decimal, y además reconoce el carácter decimal y el separador de miles según la configuración internacional que tengamos fijada.
- Si almacenamos en una variable de tipo **Byte** el valor 65, por ejemplo, y utilizamos la función **CStr** para convertirlo a cadena, obtendremos una cadena de dos caracteres, "65", y no la letra "A", que sería el carácter con el código 65. Para obtener el código de cualquier carácter y viceversa, disponemos de las funciones **Chr** y **Asc**. La primera toma como parámetro un entero entre 0 y 255, y devuelve el carácter correspondiente, mientras que la segunda toma un solo carácter y devuelve el código.
- **AscW** (toma un carácter para valores comprendidos entre 0 y 65535).
- **ChrW** (toma un parámetro entre -32768 y 65535).

Visual Basic

Ejercicio 3

- Dibujar los primeros 255 elementos (del 32 al 255) de la tabla ASCII.

Form1

32 =	33 =!	34 ="	35 =#	36 =\$	37 =%	38 =&	39 ='
40 =[41 =]	42 =*	43 =+	44 =,	45 =-	46 =.	47 =/
48 =0	49 =1	50 =2	51 =3	52 =4	53 =5	54 =6	55 =7
56 =8	57 =9	58 =:	59 =;	60 =<	61 ==	62 =>	63 =?
64 =@	65 =A	66 =B	67 =C	68 =D	69 =E	70 =F	71 =G
72 =H	73 =I	74 =J	75 =K	76 =L	77 =M	78 =N	79 =O
80 =P	81 =Q	82 =R	83 =S	84 =T	85 =U	86 =V	87 =W
88 =X	89 =Y	90 =Z	91 =[92 =\	93 =]	94 =^	95 =_
96 =`	97 =a	98 =b	99 =c	100 =d	101 =e	102 =f	103 =g
104 =h	105 =i	106 =j	107 =k	108 =l	109 =m	110 =n	111 =o
112 =p	113 =q	114 =r	115 =s	116 =t	117 =u	118 =v	119 =w
120 =x	121 =y	122 =z	123 ={	124 =	125 =}	126 =~	127 =
128 =€	129 =	130 =!	131 =!	132 =!	133 =!	134 =!	135 =!
136 =!	137 =!	138 =!	139 =!	140 =!	141 =!	142 =!	143 =!
144 =!	145 ='	146 ='	147 =!	148 =!	149 =!	150 =!	151 =!
152 =!	153 =!	154 =!	155 =!	156 =!	157 =!	158 =!	159 =!
160 =	161 =j	162 =ç	163 =£	164 =¤	165 =¥	166 =!	167 =§
168 ='	169 =©	170 =ª	171 =«	172 =¬	173 =·	174 =®	175 =
176 =°	177 =±	178 =²	179 =³	180 =´	181 =µ	182 =¶	183 =
184 =	185 =¹	186 =ª	187 =»	188 =¼	189 =½	190 =¾	191 =¿
192 =À	193 =Á	194 =Â	195 =Ã	196 =Ä	197 =Å	198 =Æ	199 =Ç
200 =È	201 =É	202 =Ê	203 =Ë	204 =Ì	205 =Í	206 =Î	207 =Ï
208 =Ð	209 =Ñ	210 =Ò	211 =Ó	212 =Ô	213 =Õ	214 =Ö	215 =×
216 =Ø	217 =Ù	218 =Ú	219 =Û	220 =Ü	221 =Ý	222 =Þ	223 =ß
224 =à	225 =á	226 =â	227 =ã	228 =ä	229 =å	230 =æ	231 =ç
232 =è	233 =é	234 =ê	235 =ë	236 =ì	237 =í	238 =î	239 =ï
240 =ð	241 =ñ	242 =ò	243 =ó	244 =ô	245 =õ	246 =ö	247 =÷
248 =ø	249 =ù	250 =ú	251 =û	252 =ü	253 =ý	254 =þ	255 =ÿ

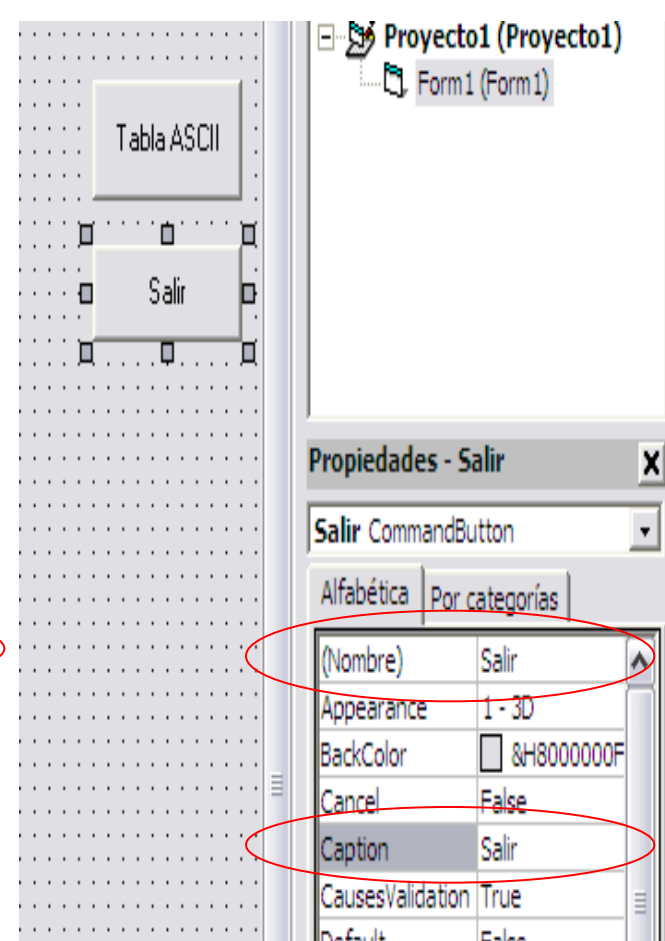
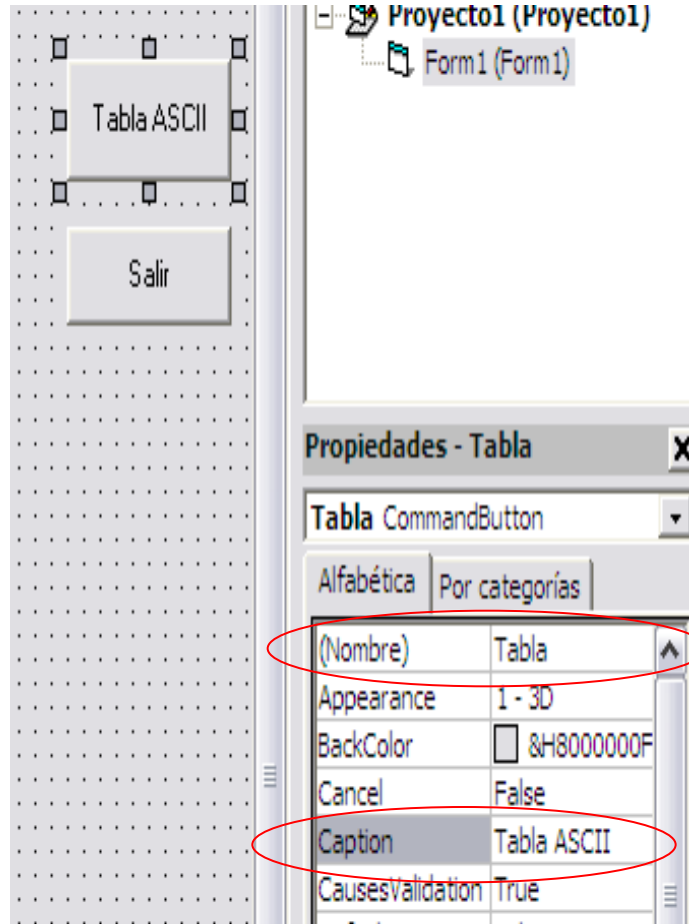
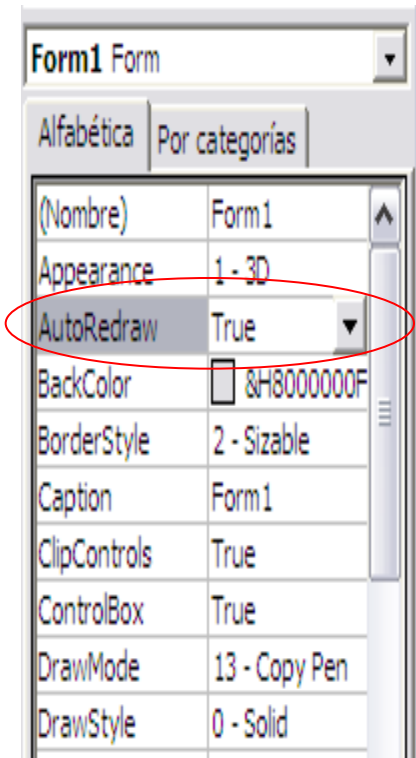
Tabla ASCII

Salir

Visual Basic

Ejercicio 3

- Dibujar los primeros 255 elementos (del 32 al 255) de la tabla ASCII.**



Ejercicio 3

- Dibujar los primeros 255 elementos (del 32 al 255) de la tabla ASCII.

Option Explicit

```
Private Sub Salir_Click()  
    End  
End Sub
```

```
Private Sub Tabla_Click()  
    Dim I As Byte, J As Byte  
    For I = 4 To 31      ' Desde el 32 al 255  
        For J = 0 To 7  
            Print I * 8 + J; "="; Chr(I * 8 + J),  
        Next  
        Print  
    Next  
End Sub
```

Visual Basic

Ejercicio 33

- Vamos a realizar una pequeña calculadora que realice las siguientes operaciones: + (Suma), - (Resta), * (Multiplicación), \ (División Entera), ^ (Potencia) y Mod (Módulo). Pulsaremos Salir para terminar la aplicación.**

The screenshot shows a Visual Basic application window titled "Form1". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main area of the form is light gray and contains the following elements:

- Two input fields for numbers, labeled "Número 1:" and "Número 2:", positioned at the top.
- A set of six operation buttons arranged in two rows of three:
 - Row 1: Addition (+), Subtraction (-), and Multiplication (*).
 - Row 2: Integer Division (\), Exponentiation (^), and Modulus (Mod).
- A "Resultado:" label followed by an output text box at the bottom left.
- A large "Salir" (Exit) button at the bottom right.

Visual Basic

Ejercicio 33

Form1

Número 1:

Número 2:

Resultado:

Propiedades - Nu...

Numero1 Label

Alfabética | Por categorías

(Nombre)	Numero1
Alignment	1 - Right Justify
Appearance	1 - 3D
AutoSize	False
BackColor	&H80000000
BackStyle	1 - Opaque
BorderStyle	0 - None
Caption	Número 1:

Propiedades - Nu...

Numero2 Label

Alfabética | Por categorías

(Nombre)	Numero2
Alignment	1 - Right Justify
Appearance	1 - 3D
AutoSize	False
BackColor	&H80000000
BackStyle	1 - Opaque
BorderStyle	0 - None
Caption	Número 2:

Propiedades - EtiquetaR...

EtiquetaResultado Label

Alfabética | Por categorías

(Nombre)	EtiquetaResultado
Alignment	1 - Right Justify
Appearance	1 - 3D
AutoSize	False
BackColor	&H80000000F&
BackStyle	1 - Opaque
BorderStyle	0 - None
Caption	Resultado:

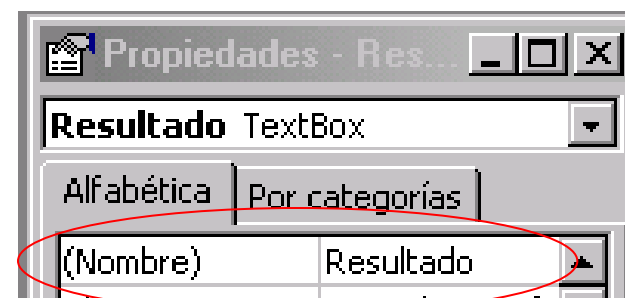
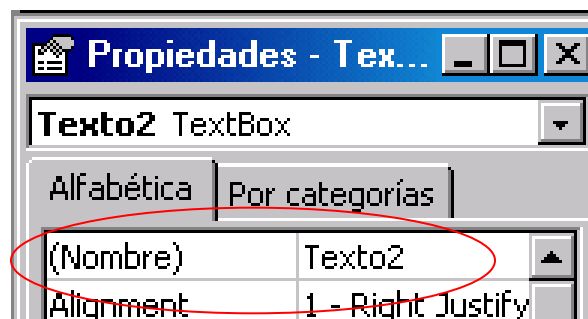
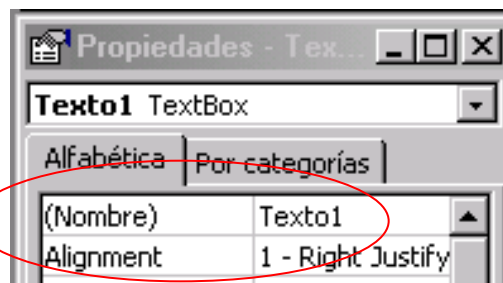
Visual Basic

Ejercicio 33

Número 1:

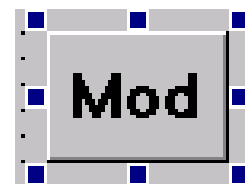
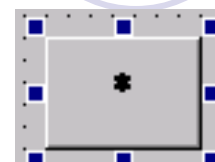
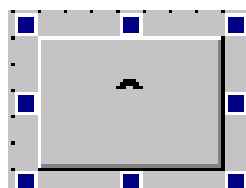
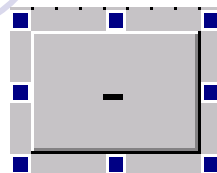
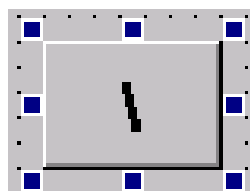
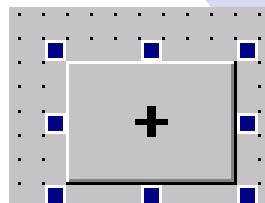
Número 2:

Resultado:



Visual Basic

Ejercicio 33



Visual Basic

Ejercicio 33

(General)

```
Dim Num1 As Integer  
Dim Num2 As Integer  
Dim ResultadoT As Long
```

```
Private Sub Division_Click()  
    Num1 = CInt(Texto1.Text)  
    Num2 = CInt(Texto2.Text)  
    ResultadoT = Num1 \ Num2  
    Resultado.Text = CStr(ResultadoT)  
End Sub
```

```
Private Sub Modulo_Click()  
    Num1 = CInt(Texto1.Text)  
    Num2 = CInt(Texto2.Text)  
    ResultadoT = Num1 Mod Num2  
    Resultado.Text = CStr(ResultadoT)  
End Sub
```

```
Private Sub Potencia_Click()  
    Num1 = CInt(Texto1.Text)  
    Num2 = CInt(Texto2.Text)  
    ResultadoT = Num1 ^ Num2  
    Resultado.Text = CStr(ResultadoT)  
End Sub
```

Visual Basic

Ejercicio 33

```
Private Sub Producto_Click()  
    Num1 = CInt(Texto1.Text)  
    Num2 = CInt(Texto2.Text)  
    ResultadoT = Num1 * Num2  
    Resultado.Text = CStr(ResultadoT)  
End Sub
```

```
Private Sub Resta_Click()  
    Num1 = CInt(Texto1.Text)  
    Num2 = CInt(Texto2.Text)  
    ResultadoT = Num1 - Num2  
    Resultado.Text = CStr(ResultadoT)  
End Sub
```

```
Private Sub Salir_Click()  
    End  
End Sub
```

```
Private Sub Suma_Click()  
    Num1 = CInt(Texto1.Text)  
    Num2 = CInt(Texto2.Text)  
    ResultadoT = Num1 + Num2  
    Resultado.Text = CStr(ResultadoT)  
End Sub
```

Ejercicio 4

- Vamos a pasar el texto “Texto de Prueba” a Arial, Courier, + Tamaño, - Tamaño, Negrita, Cursiva, Tachada y Subrayada.**

The screenshot shows a Visual Basic form titled "Form1". On the left, there is a text box containing the text "Texto de Prueba". To the right of the text box is a collection of buttons for text formatting. The buttons are arranged in two columns. The first column contains: "Arial", "Courier", "+ Tamaño", and "- Tamaño". The second column contains: "Negrita", "Cursiva", "Tachada", "Subrayada", and "Salir". The "Arial" button is currently selected, indicated by a dotted border.

Formato	Botón
Fuente	Arial
Fuente	Courier
Tamaño	+ Tamaño
Tamaño	- Tamaño
Estilo	Negrita
Estilo	Cursiva
Estilo	Tachada
Estilo	Subrayada
Acción	Salir

Visual Basic

Ejercicio 4

Form1 Form

Alfabetica Por categorias

Palette	(Ninguno)
PaletteMode	0 - Halftone
Picture	(Ninguno)
RightToLeft	False
ScaleHeight	11010
ScaleLeft	0
ScaleMode	1 - Twip
ScaleTop	0
ScaleWidth	15240
ShowInTaskbar	True
StartPosition	3 - Windows Defau
Tag	
Top	0
Visible	True
WhatsThisButton	False
WhatsThisHelp	False
Width	15360
WindowState	2 - Maximized

Texto de Prueba

Arial Negrita

Courier Cursiva

+ Tamaño Iachada

- Tamaño Subrayada

Salir

Proyecto1 (Proyecto1)

Form1 (Form1)

Propiedades - Text1

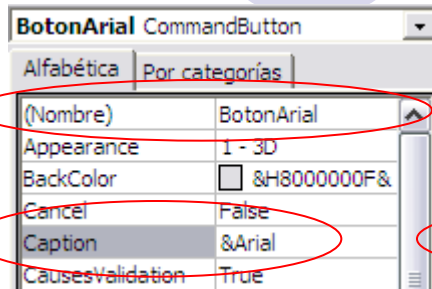
Text1 TextBox

Alfabetica Por categorias

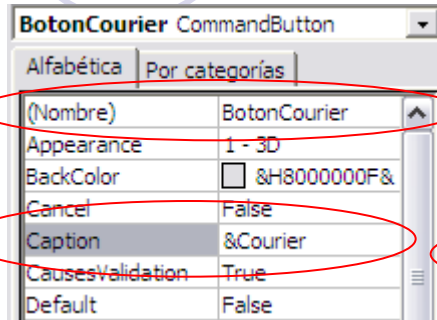
MaxLength	0
MouseIcon	(Ninguno)
MousePointer	0 - Default
MultiLine	False
OLEDragMode	0 - Manual
OLEDropMode	0 - None
PasswordChar	
RightToLeft	False
ScrollBars	0 - None
TabIndex	0
TabStop	True
Tag	
Text	Texto de Prueba
ToolTipText	
Top	240
Visible	True

Visual Basic

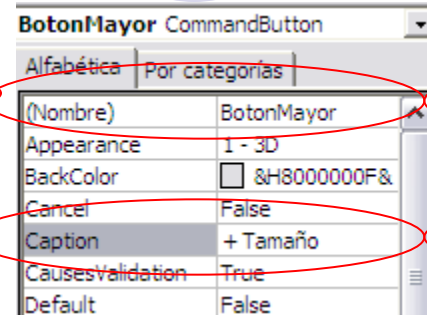
Ejercicio 4



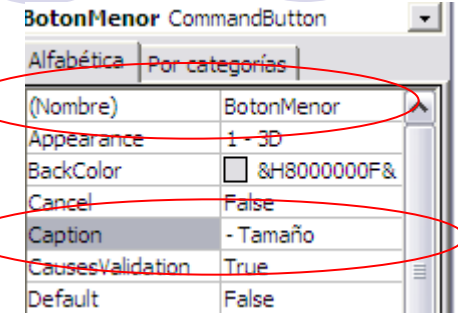
BotonArial CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonArial
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	&Arial
CausesValidation	True



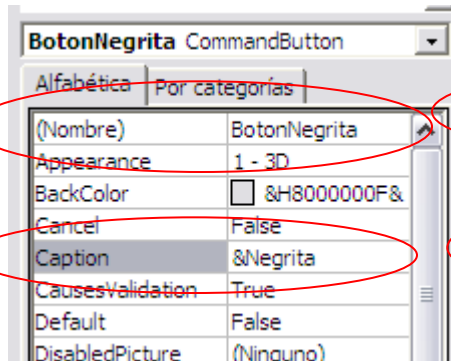
BotonCourier CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonCourier
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	&Courier
CausesValidation	True
Default	False



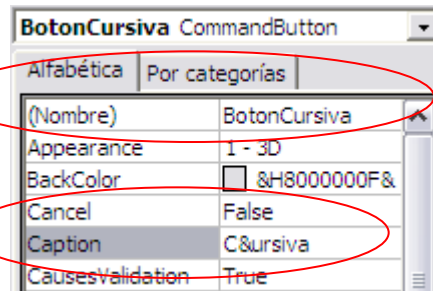
BotonMayor CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonMayor
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	+ Tamaño
CausesValidation	True
Default	False



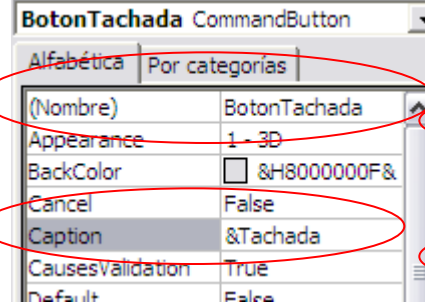
BotonMenor CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonMenor
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	- Tamaño
CausesValidation	True
Default	False



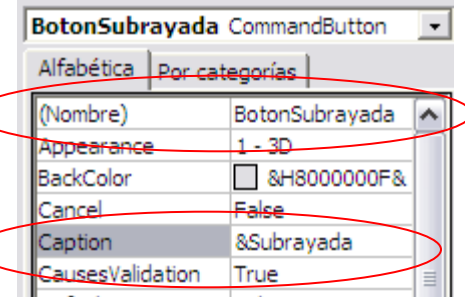
BotonNegrita CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonNegrita
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	&Negrita
CausesValidation	True
Default	False
DisabledPicture	(Ninguno)



BotonCursiva CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonCursiva
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	C&ursiva
CausesValidation	True

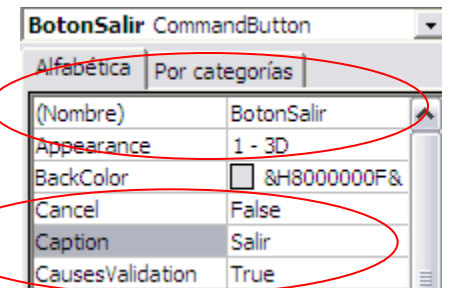


BotonTachada CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonTachada
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	&Tachada
CausesValidation	True
Default	False



BotonSubrayada CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonSubrayada
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	&Subrayada
CausesValidation	True

La selección de la opción se realiza pulsando **Alt** + la **tecla subrayada**



BotonSalir CommandButton	
Alfabetica	Por categorias
(Nombre)	BotonSalir
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H8000000F&
Cancel	False
Caption	Salir
CausesValidation	True

Visual Basic

Ejercicio 4

Option Explicit

```
Private Sub BotonArial_Click()
```

```
    Text1.FontName = "Arial"
```

```
End Sub
```

```
Private Sub BotonCourier_Click()
```

```
    Text1.FontName = "Courier"
```

```
End Sub
```

```
Private Sub BotonMayor_Click()
```

```
    Text1.FontSize = Text1.FontSize + 4
```

```
End Sub
```

```
Private Sub BotonMenor_Click()
```

```
    Text1.FontSize = Text1.FontSize - 4
```

```
End Sub
```

```
Private Sub BotonNegrita_Click()
```

```
    Text1.FontBold = Not Text1.FontBold
```

```
End Sub
```

```
Private Sub BotonCursiva_Click()
```

```
    Text1.FontItalic = Not Text1.FontItalic
```

```
End Sub
```

```
Private Sub BotonTachada_Click()
```

```
    Text1.FontStrikethru = Not Text1.FontStrikethru
```

```
End Sub
```

```
Private Sub BotonSubrayada_Click()
```

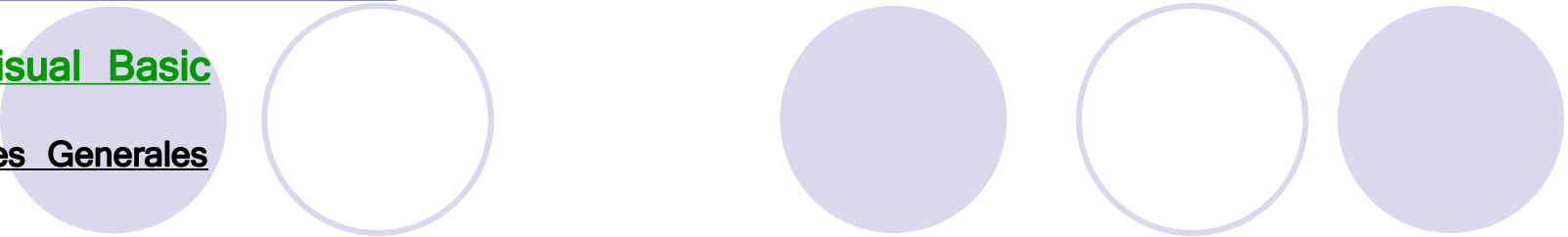
```
    Text1.FontUnderline = Not Text1.FontUnderline
```

```
End Sub
```


```
Private Sub BotonSalir_Click()
```

```
    End
```

```
End Sub
```



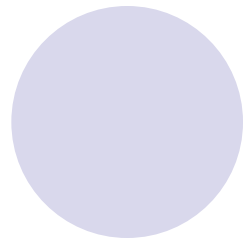
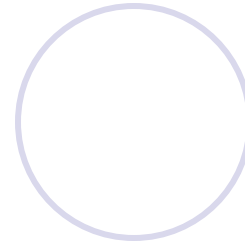
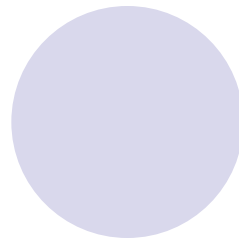
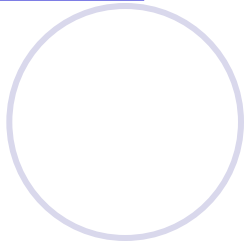
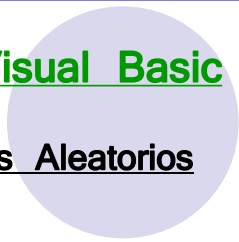
Abs	Devuelve el valor absoluto de un número dado
Sgn	Indica el signo de un número (-1 negativo, 1 positivo, 0 si 0)
Sqr	Halla la raíz cuadrada de un número dado
Exp	Calcula el número elevado al exponente indicado
Log	Halla el logaritmo natural del número dado



Sin	Seno de un ángulo (radianes)
Cos	Coseno de un ángulo (radianes)
Tan	Tangente de un ángulo (radianes)
Atn	Arco de una tangente (radianes)

Visual Basic

Números Aleatorios



- La primera de ellas, **Randomize**, **inicializa el generador aleatorio**, bien a partir del número pasado como parámetro o a partir del valor devuelto por la función **Timer**.
- Cada vez que deseemos obtener un número aleatorio usaremos la función **Rnd**, con la que obtendremos un numero decimal comprendido entre 0 y 1.
- Por ejemplo: la sentencia:

Print CInt (Rnd * 49 + 1)

devolverá siempre un número entero aleatorio comprendido entre 1 y 49.

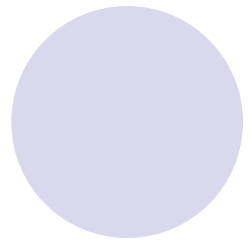
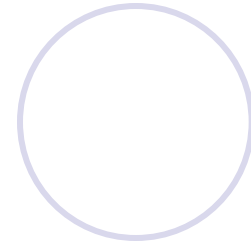
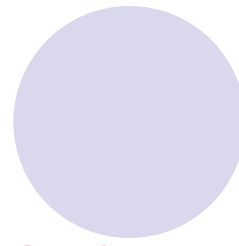
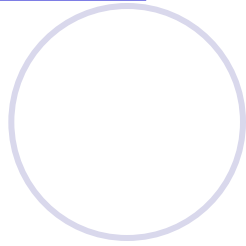
Visual Basic

Trabajo con Cadenas

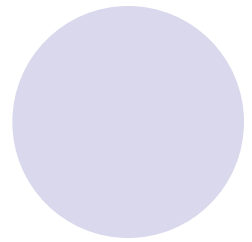
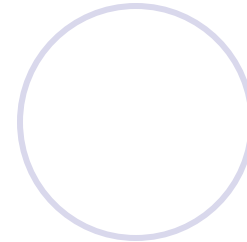
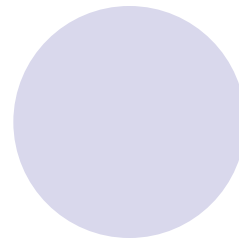
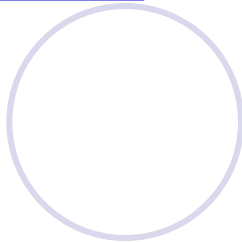
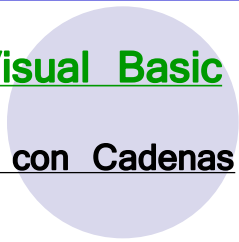
Left(<i>cadena</i> , <i>nº_caracteres_a_extraer</i>)	Extrae el número de caracteres indicado en <i>nº_caracteres_a_extraer</i> de la cadena <i>cadena</i> contando de izquierda a derecha
Right(<i>cadena</i> , <i>nº_caracteres_a_extraer</i>)	Extrae el número de caracteres indicado en <i>nº_caracteres_a_extraer</i> de la cadena <i>cadena</i> contando de derecha a izquierda
Mid(<i>cadena</i> , <i>carácter_de_inicio</i> , <i>nº_caracteres_a_extraer</i>)	Extra el número de caracteres indicado en <i>nº_caracteres_a_extraer</i> de la cadena <i>cadena</i> indicando el carácter desde el que queremos comenzar a extraer en <i>carácter_de_inicio</i> y realizando la extracción de izquierda a derecha
LTrim(<i>cadena</i>)	Elimina espacios al principio de una <i>cadena</i>
RTrim(<i>cadena</i>)	Elimina espacios al final de una <i>cadena</i>
Trim(<i>cadena</i>)	Elimina espacios tanto al principio como al final de la <i>cadena</i>
LCase(<i>cadena</i>)	Convierte las MAYÚSCULAS de la <i>cadena</i> en minúsculas
UCase(<i>cadena</i>)	Convierte las minúsculas de la <i>cadena</i> en MAYÚSCULAS
StrConv(<i>cadena</i> , <i>opción</i>)	Convierte una <i>cadena</i> según una <i>opción</i>
Space(<i>número</i>)	Genera una cadena de espacios. <i>número</i> indica el número de espacios que va a tener la cadena
String(<i>número</i> , <i>carácter</i>)	Genera una cadena de un cierto carácter indicado. <i>carácter</i> va a ser el carácter del que va a estar compuesta la cadena y que va a estar repetido un número <i>número</i> de veces
Len (<i>cadena</i>)	Devuelve la longitud de la <i>cadena</i>
InStr(<i>posición</i> , <i>cadena1</i> , <i>cadena2</i>)	Busca una cadena, <i>cadena2</i> , dentro de otra cadena, <i>cadena1</i> , a partir del carácter de la <i>cadena1</i> que se encuentra en la posición <i>posición</i> . Si <i>cadena1</i> no está en <i>cadena2</i> , InStr devuelve 0

Visual Basic

Trabajo con Cadenas



- En cuanto a las opciones de la función **StrConv**, podemos destacar las siguientes:
 - cadena = prácticas de Visual Basic
 - Resultado = StrConv(cadena, **vbUpperCase**). Resultado = PRÁCTICAS DE VISUAL BASIC
 - Resultado = StrConv(cadena, **vbLowerCase**). Resultado = prácticas de visual basic
 - Resultado = StrConv(cadena, **vbProperCase**). Resultado = Prácticas De Visual Basic



- Una variable de tipo **String** permite almacenar una cadena de caracteres y manipularla utilizando las funciones proporcionadas por Visual Basic:

```
Private Sub Command1_Click()  
    Dim Nombre as String  
    Nombre = "Pepito Pérez"  
    Nombre = Left(Nombre, 4)  
    MsgBox Nombre  
End Sub
```

- Después de ejecutar este procedimiento, el contenido de *Nombre* es "*Pepi*". La variable *Nombre* está definida como una cadena de caracteres de longitud variable. Para declarar *Nombre* como una cadena de caracteres de longitud fija, por ejemplo, de 60 caracteres, hay que hacer:

```
Dim Nombre As String * 60  
Nombre = "Pepito Pérez"
```

- Si asignamos a *Nombre* menos de 60 caracteres, se completa con espacios en blanco hasta 60 caracteres.

Visual Basic

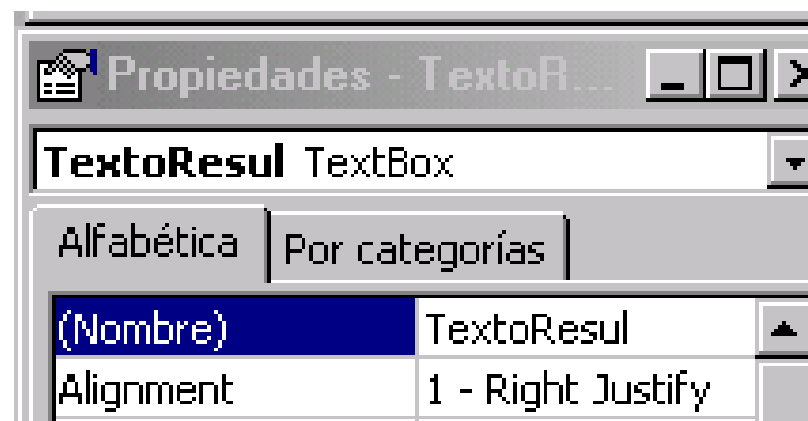
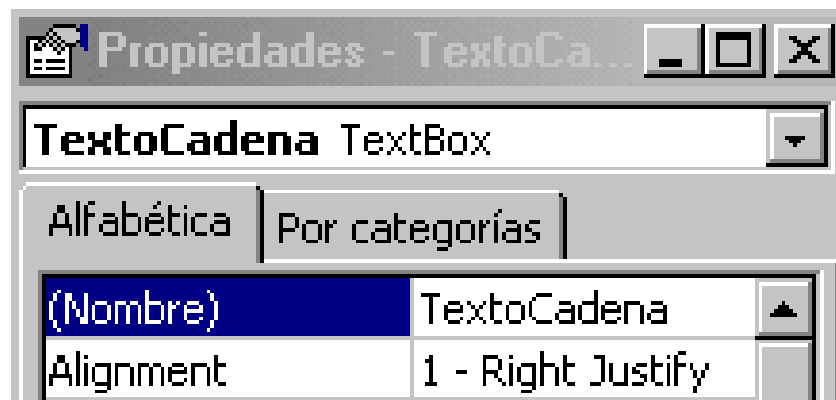
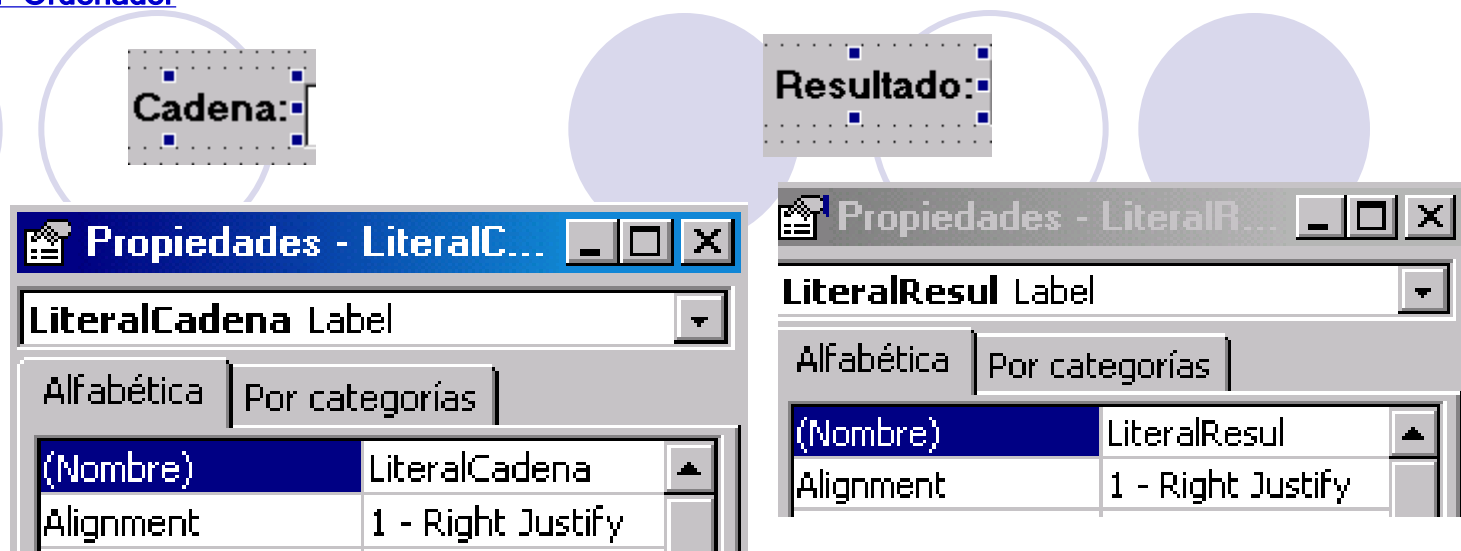
Ejercicio 44

- **Vamos a trabajar con la cadena que escribamos en “Cadena” cortando los 4 primeros caracteres de la Izquierda (*Izquierda*), los 4 primeros de la Derecha (*Derecha*), los 4 caracteres del centro a partir del tercero (*Centro*), la vamos a pasar a MAYÚSCULAS (*MAYUS*) y a minúsculas (*minus*). Pulsando Salir acabamos.**

The screenshot shows a Visual Basic form titled "Form1" with a light gray background. At the top, there is a label "Cadena:" followed by a text box containing the string "Emerita Augusta". Below this, there are three buttons arranged horizontally: "Izquierda", "Centro", and "Derecha". The "Izquierda" button is highlighted with a dashed border. Below these buttons are three more buttons arranged horizontally: "MAYUS", "minus", and "Salir". At the bottom, there is a label "Resultado:" followed by a text box containing the string "Emer".

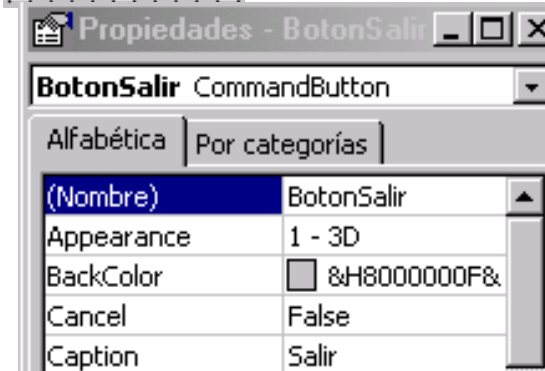
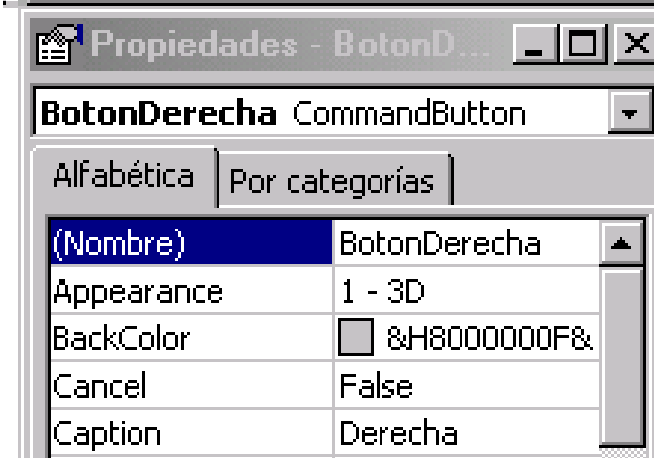
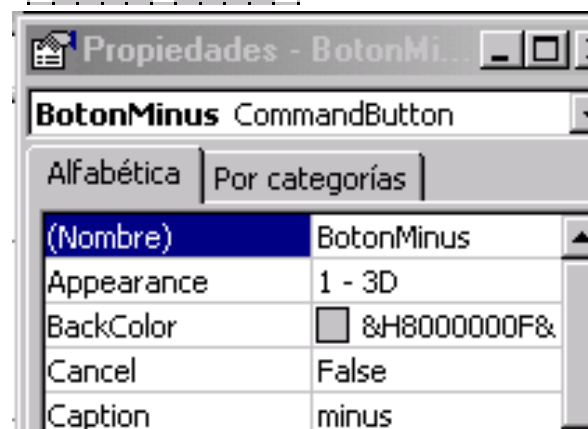
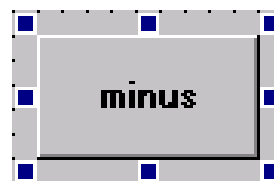
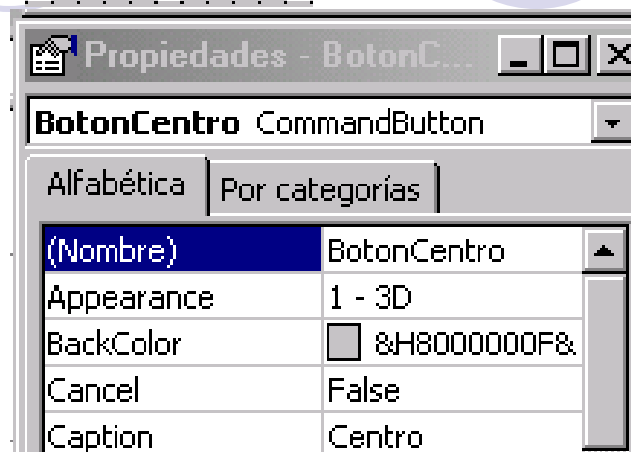
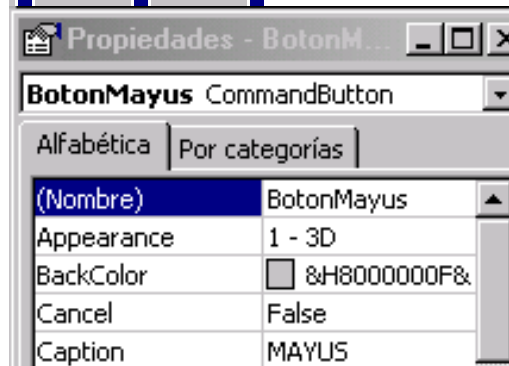
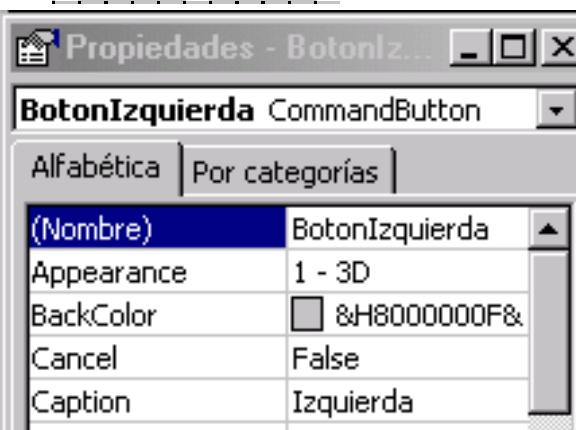
Visual Basic

Ejercicio 44



Visual Basic

Ejercicio 44



Visual Basic

Ejercicio 44

```
Private Sub BotonCentro_Click()  
    TextoResul.Text = Mid(TextoCadena.Text, 3, 4)  
End Sub
```

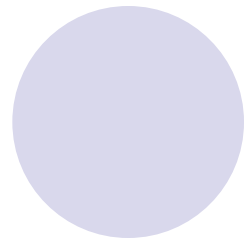
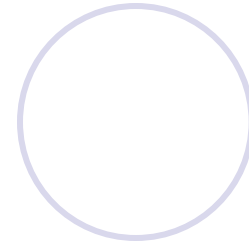
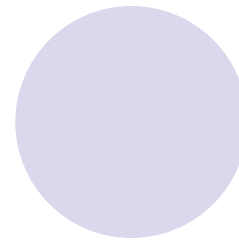
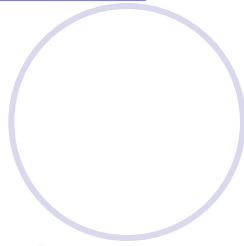
```
Private Sub BotonDerecha_Click()  
    TextoResul.Text = Right(TextoCadena.Text, 4)  
End Sub
```

```
Private Sub BotonIzquierda_Click()  
    TextoResul.Text = Left(TextoCadena.Text, 4)  
End Sub
```

```
Private Sub BotonMayus_Click()  
    TextoResul.Text = UCase(TextoCadena.Text)  
End Sub
```

```
Private Sub BotonMinus_Click()  
    TextoResul.Text = LCase(TextoCadena.Text)  
End Sub
```

```
Private Sub BotonSalir_Click()  
    End  
End Sub
```



- En un módulo estándar hay que declarar las cadenas de caracteres de longitud fija, como **Private** o **Public**. En un formulario o en una clase hay que declararlas **Private**.
- Si el contenido de una cadena de caracteres coincide con un valor numérico, se puede asignar la cadena de caracteres a una variable numérica. También es posible asignar un valor numérico a una cadena de caracteres:

Dim X As Integer, Y As Single

Dim Cadena As String

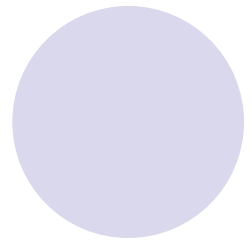
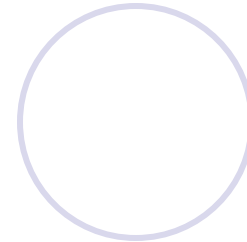
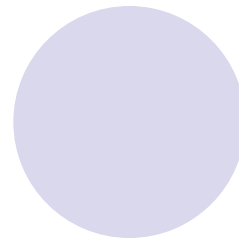
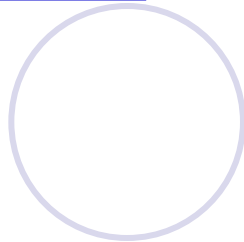
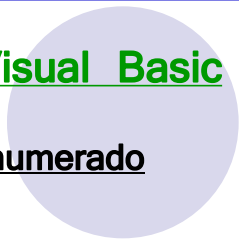
Y = 50.65

Cadena = Y ‘ O también, Cadena = “50.65”

X = Cadena ‘ El valor de X es 51.

Visual Basic

Tipo Enumerado



- La declaración de un tipo Enumerado (**Enum**) es simplemente una lista de valores que pueden ser tomados por una variable de ese tipo. Los valores de un tipo enumerado se representarán con identificadores, que serán las constantes del nuevo tipo:

```
Public Enum DiasSemana
```

```
Lunes
```

```
Martes
```

```
Miércoles
```

```
Jueves
```

```
Viernes
```

```
Sábado
```

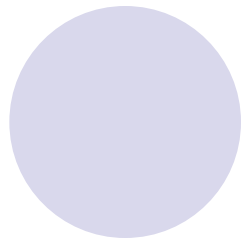
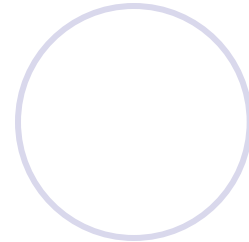
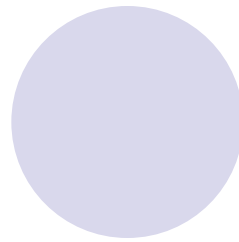
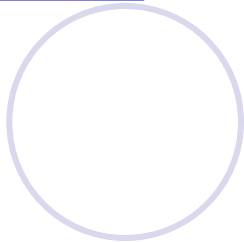
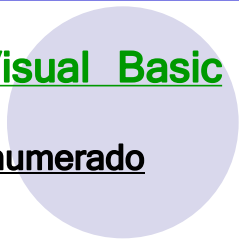
```
Domingo
```

```
End Enum
```

```
Dim Hoy As DiasSemana
```

Visual Basic

Tipo Enumerado



- En el ejemplo anterior hemos declarado la variable *Hoy* del tipo enumerado *DiasSemana*. Esta variable puede tomar cualquier valor de los especificados, de *Lunes* a *Domingo*. Las constantes son de tipo **Long** y sus valores por omisión son: 0, 1, 2, 3,... Según esto, el valor de *Lunes* es 0, el de *Martes* es 1, el de *Miércoles* es 2, y así sucesivamente:

Hoy = Domingo

sería 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:

Visual Basic

Tipo Enumerado

Public Enum DíasLaborables

Sábado

Domingo = 0

Lunes

Martes

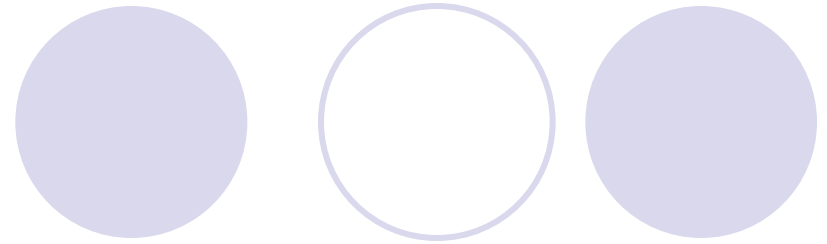
Miércoles

Jueves

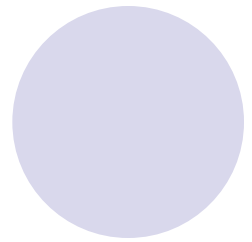
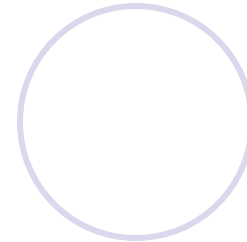
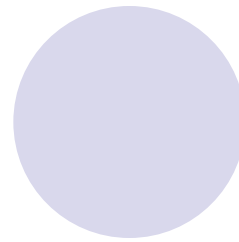
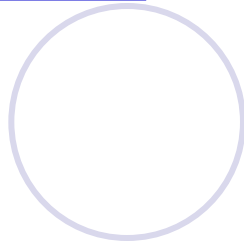
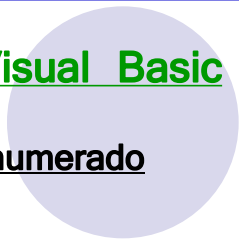
Viernes

NoVálido = -1

End Enum



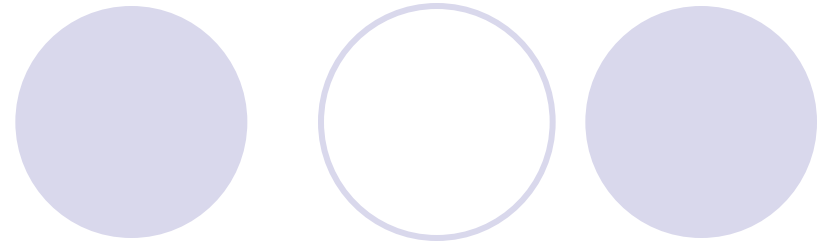
- Este ejemplo declara un tipo enumerado llamado *DíasLaborables*. Los valores asociados con cada una de las constantes son: *Sábado* = 0, *Domingo* = 0, *Lunes* = 1, *Martes* = 2, *Miércoles* = 3, *Jueves* = 4, *Viernes* = 5, *NoVálido* = -1.



- 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, las siguientes constantes reciben un valor superior en una unidad al de la 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 hagamos referencia a una constante individual, debemos calificar el nombre de la constante mediante su enumeración:
`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 y en comparaciones sí se pueden utilizar los identificadores del tipo enumerado.

Visual Basic

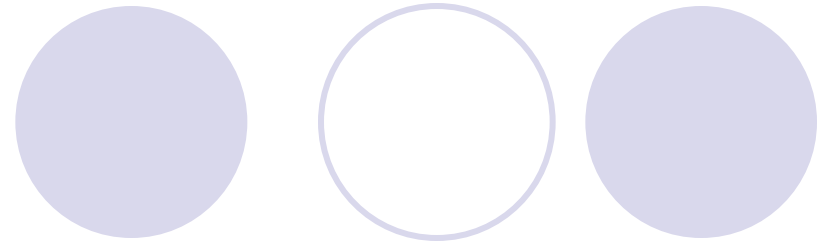
Sentencias de Control – IF...THEN...ELSE



- Estructura:
 - a) **If** *condición* **Then** *sentencia(s)* [**Else** *sentencia(s)*]
 - b) **If** *condición* **Then**
 sentencia(s)
 [Else
 sentencias]
 End If

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Ejemplos:

If $a > 3$ **Then** $b = 7$ **Else** $b = 0$

If $a > 3$ **Then**

$b = 7$

$c = 8$

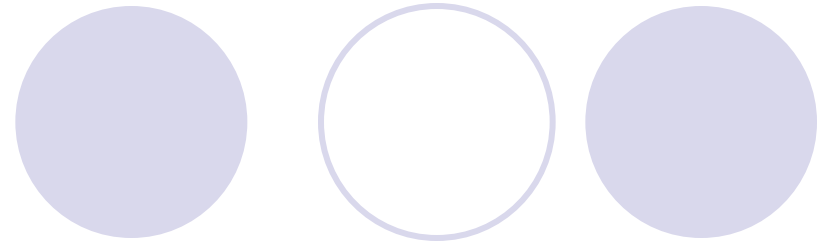
Else

$b = 0$

End If

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Estructura:

c) **If** *condición-1* **Then**

sentencia(s)-1

[Elseif *condición-2* **Then**

sentencias-2]

.

[Else

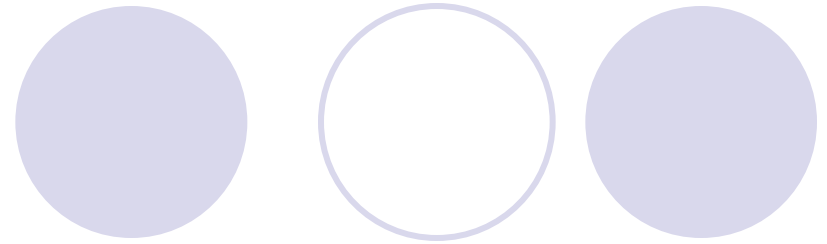
sentencias-n]

End If

d) *Resultado = If (condición, parte_verdadera, parte_falsa)*

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Ejemplos:

If $c > 100$ **Then**

$Resu = c * p * 0.6$ ' Si $c > 100$

Elseif $c \geq 25$ **Then**

$Resu = c * p * 0.8$ ' Si $c \geq 25$

Elseif $c \geq 10$ **Then**

$Resu = c * p * 0.9$ ' Si $c \geq 10$

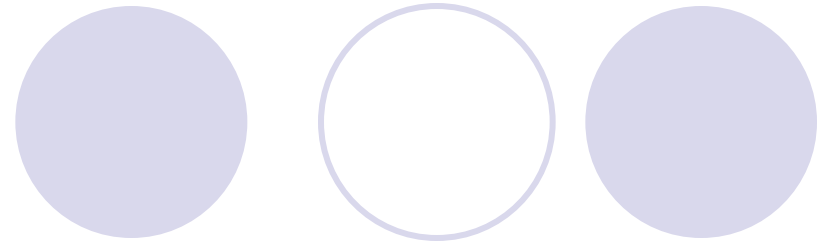
Else

$Resu = c * p$ ' Si **c** NO es ni > 100 , ni ≥ 25 , ni ≥ 10

End If

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Ejemplos:

If $A = 1$ **Then**

If $B = 5$ **Then**

$C = A + B$ ‘ $A = 1$ y $B = 5$

Else

$C = A - B$ ‘ $A = 1$ y $B \neq 5$

End If

Elseif $A = 3$ **Then**

$C = A * B$ ‘ $A = 3$ y $B = ?$

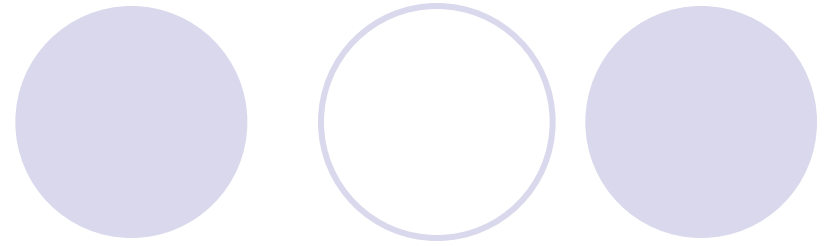
Else

$C = A / B$ ‘ $A = ?$ y $B = ?$

End If

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Ejemplos:

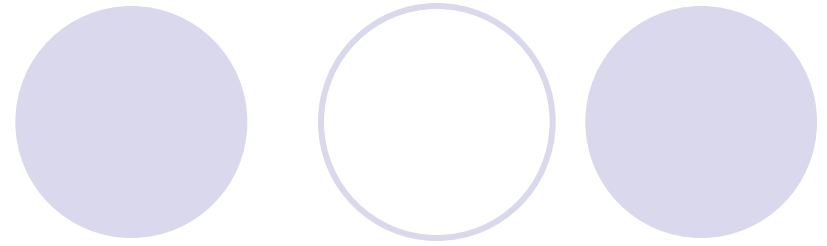
If $B = 5$ **Then** $A = 1$ **Else** $A = 2$

es equivalente a:

$A = \mathbf{IIf}$ (B=5, 1, 2)

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Supongamos ahora la siguiente estructura:

```
If B = 5 Then
```

```
    A = 1
```

```
Elseif B <= 0 Then
```

```
    A = -1
```

```
Elseif B > 5 Then
```

```
    A = 0
```

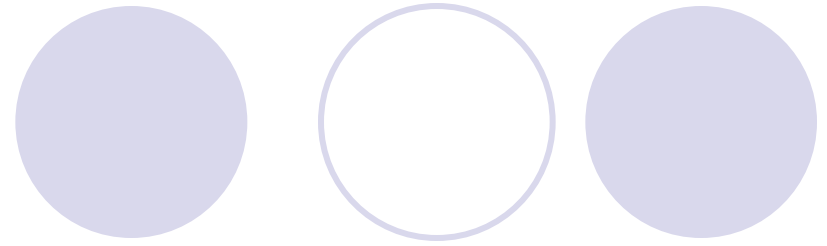
```
Endif
```

Toda esta secuencia de decisiones se puede reducir a tan sólo una línea, por medio de la función **Switch**. Esta función puede tomar dos o más parámetros, **siempre por parejas**. El primer elemento de cada pareja es la expresión a evaluar, mientras que el segundo corresponde al valor a devolver en caso de que la expresión sea cierta. La primera expresión que se cumpla devolverá el valor correspondiente:

```
A = Switch (B = 5, 1, B <= 0, -1, B > 5, 0)
```

Visual Basic

Sentencias de Control – IF...THEN...ELSE



- Supongamos ahora la siguiente estructura:

```
If B = 1 Then
    A = 5
Elseif B = 2 Then
    A = 7
Elseif B = 3 Then
    A = 11
Endif
```

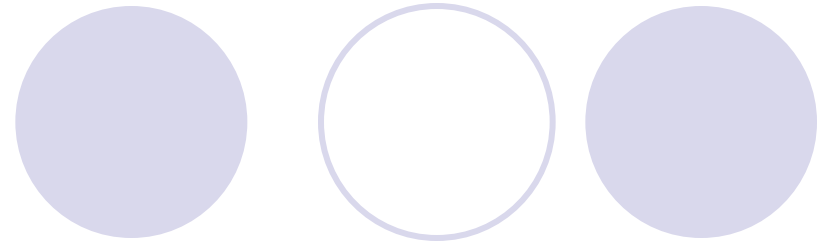
Toda esta secuencia de decisiones se puede reducir a tan sólo una línea, por medio de la función **Choose**. Esta función toma un primer parámetro (B, en este caso) que **actuará como índice**, y después, tantos valores como sea necesario; de tal forma que dependiendo del valor del índice se devuelva uno u otro:

```
A = Choose (B, 5, 7, 11)
```

Hay que tener en cuenta que si no es posible devolver un valor tanto con **Switch** como con **Chosee**, se devolverá el valor **Null** (que tendrá que ir sobre un **Variant**).

Visual Basic

Sentencias de Control – SELECT CASE



• Estructura:

Select Case *expresión_a_evaluar*

Case *opción_1*

[sentencias_1]

[Case *opción_2*

[sentencias_2]]

[Case *opción_3*

[sentencias_3]]

.....

[Case Else

[sentencias_n]]

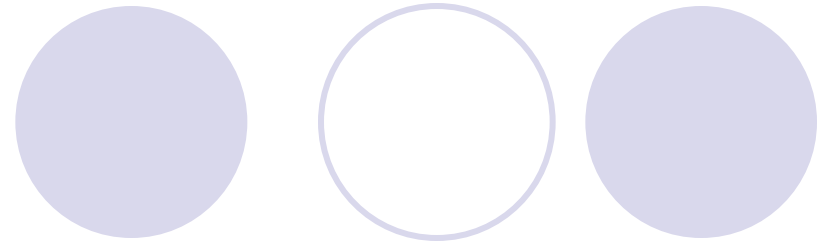
End Select

- *expresión* [, *expresión*]....
- *expresión* **To** *expresión*
- **Is** *operador_de_relación* *expresión*
- combinación de las anteriores separadas por comas

donde *expresión_a_evaluar* es una expresión numérica o alfanumérica, y *opción_1*, *opción_2*, *opción_3*,... representan una lista que puede tener cualquiera de las formas siguientes:

Visual Basic

Sentencias de Control – **SELECT CASE**



• Ejemplo:

Select Case B

Case Is < X

Print “El valor de B es menor que el de X”

B = 100

Case 3

Print “B = 3”

Case X **To** 20

Print “B = X o B = X+1 o B = X+2 o B = X+3 o ... o B = X+20”

B = 20

Case 3, X

Print “B = 3 o B = X”

B = 3

Case -1, X **To** 5

Print “B = -1 o B = X o B = X+1 o B = X+2 o ... o B = X+5”

B = -1

Case “si”, “SI”

Print “B = si o B = SI”

Case Is >= 10

Print “B >= 10”

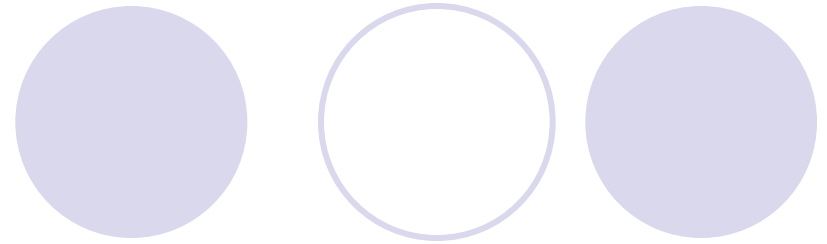
Case Else

Print “Valor NO valido para B”

End Select

Visual Basic

Sentencias de Control – SELECT CASE



- Ejemplo:

Select Case B

Case 1

Text1.Text = “1”

Case 2, 3

Text1.Text = “2 o 3”

Case 4 To 9

Text1.Text = “De 4 a 9”

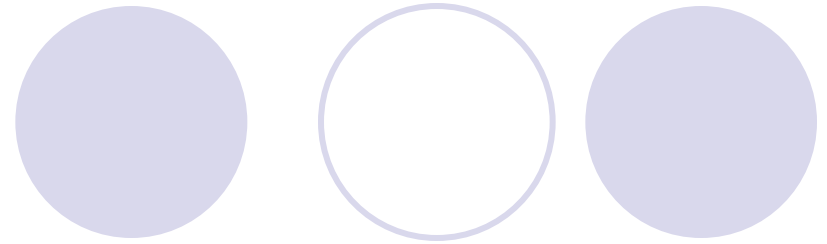
Case Else

Text1.Text = “B<1 o B>9”

End Select

Visual Basic

Sentencias de Control – **SELECT CASE**



- **Ejemplo:**

Select Case B

Case 5

Print “B tiene el valor óptimo”

A = 10

Case Is > 0 And B < 5

Print “B está por debajo del valor óptimo”

A = 3

Case Is > 5

Print “B esá por encima del valor óptimo”

A = 7

Case Else

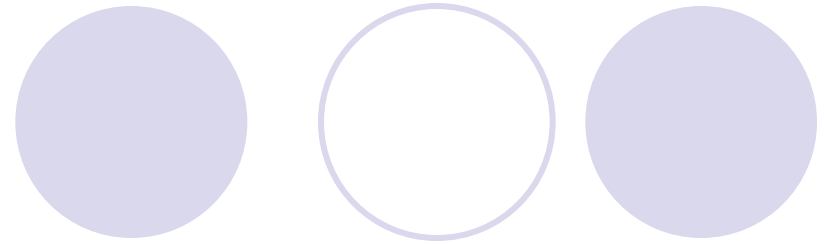
Print “Valor NO valido para B”

A = 0

End Select

Visual Basic

Sentencias de Control – DO...WHILE



- Estructura:

Do While *condición*

[*sentencias*]

[**Exit Do**]

[*sentencias*]

Loop

‘**Exit Do** permite salir del bucle Do...While antes de
‘que éste acabe.

- Ejemplo:

Do While $I \leq 99$ ‘ **Hacer mientras** $I \leq 99$

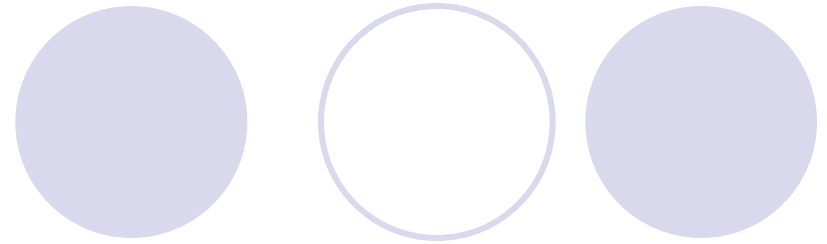
Suma = Suma + 1

$I = I + 2$

Loop

Visual Basic

Sentencias de Control – DO...UNTIL



- Estructura:

Do

[*sentencias*]

[**Exit Do**]

[*sentencias*]

Loop Until *condición*

‘**Exit Do** permite salir del bucle Do...Until antes de
‘que éste acabe.

- Ejemplo:

Do

‘ **Hacer hasta** que $I < 1$

Suma = Suma + 1

$I = I - 2$

Loop Until $I < 1$

Visual Basic

Sentencias de Control – FOR...NEXT

- Estructura:

For *variable = expresión1 To expresión2* [**Step** *expresión3*]

[sentencias]

 [**Exit For**]

[sentencias]

Next [*variable[,variable]...*]

‘**Exit For** permite salir del bucle For...Next antes de
‘que éste acabe.

- Ejemplo:

For *l = 1 To 99 Step 2*

 Suma = Suma + *l*

Next *l*

‘ Para *l = 1, 3, 5, 7, ...* hasta 99

For *l = 99 To 1 Step - 2*

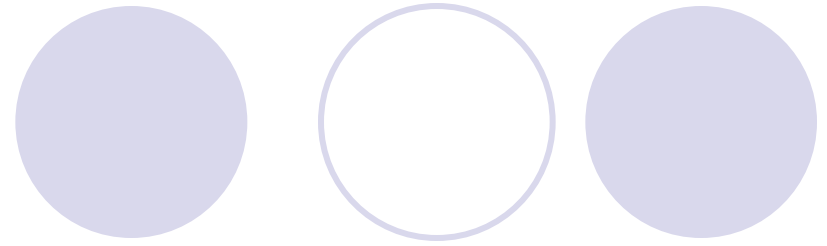
 Suma = Suma + *l*

Next *l*

‘ Para *l = 99, 97, 95, 93, ...* hasta 1

Visual Basic

Sentencias de Control – FOR...NEXT



- Ejemplo:

```
For H = 1 To 4 Step 0.5    ‘ Para H = 1, 1.5, 2,..., 3.5, 4
    Print H
Next H
```


Visual Basic

Ejercicio 5

- **Vamos a cambiar el color de texto variando sus componentes Rojo, Verde y Azul.**

The screenshot shows a Visual Basic form titled "Form1". On the left, there is a text box containing the text "Color de Texto" in a yellow, cursive font. To the right of the text box is a control panel with nine buttons arranged in a 3x3 grid. The top row of buttons is labeled "Ro+", "Ve+", and "Az+". The middle row of buttons displays the current values for the color components: "255" (in red), "255" (in green), and "0" (in blue). The bottom row of buttons is labeled "Ro-", "Ve-", and "Az-". Below this grid is a single button labeled "Salir".

Control	Rojo (R)	Verde (G)	Azul (B)
Increment (+)	Ro+	Ve+	Az+
Value	255	255	0
Decrement (-)	Ro-	Ve-	Az-
Action	Salir		

Visual Basic

Ejercicio 5

Propiedades - Form1

Form1 Form

Alfabetica Por categorias

(Nombre)	Form1
Appearance	1 - 3D
AutoRedraw	True
BackColor	&H00F0E0E0&

Propiedades - Text1

Text1 TextBox

Alfabetica Por categorias

(Nombre)	Text1
Alignment	2 - Center
Appearance	1 - 3D

Text1 TextBox

Alfabetica Por categorias

MaxLength	0
MouseIcon	(Ninguno)
MousePointer	0 - Default
MultiLine	False
OLEDragMode	0 - Manual
OLEDropMode	0 - None
PasswordChar	
RightToLeft	False
ScrollBars	0 - None
TabIndex	0
TabStop	True
Tag	
Text	Color de Texto
ToolTipText	
Top	600
Visible	True
WhatsThisHelpID	0
Width	3495

Propiedades - Form1

Form1 Form

Alfabetica Por categorias

Palette	(Ninguno)
PaletteMode	0 - Halftone
Picture	(Ninguno)
RightToLeft	False
ScaleHeight	11010
ScaleLeft	0
ScaleMode	1 - Twip
ScaleTop	0
ScaleWidth	15240
ShowInTaskbar	True
StartPosition	3 - Windows Default
Tag	
Top	0
Visible	True
WhatsThisButton	False
WhatsThisHelp	False
Width	15360
WindowState	2 - Maximized

Fuente

Fuente: Bat

Estilo de fuente: Negrita

Tamaño: 28

Efectos

☐ Tachado

☐ Subrayado

Ejemplo

AaBbYy

Alfabeto: Occidental

Aceptar

Cancelar

Text1 TextBox

Alfabetica Por categorias

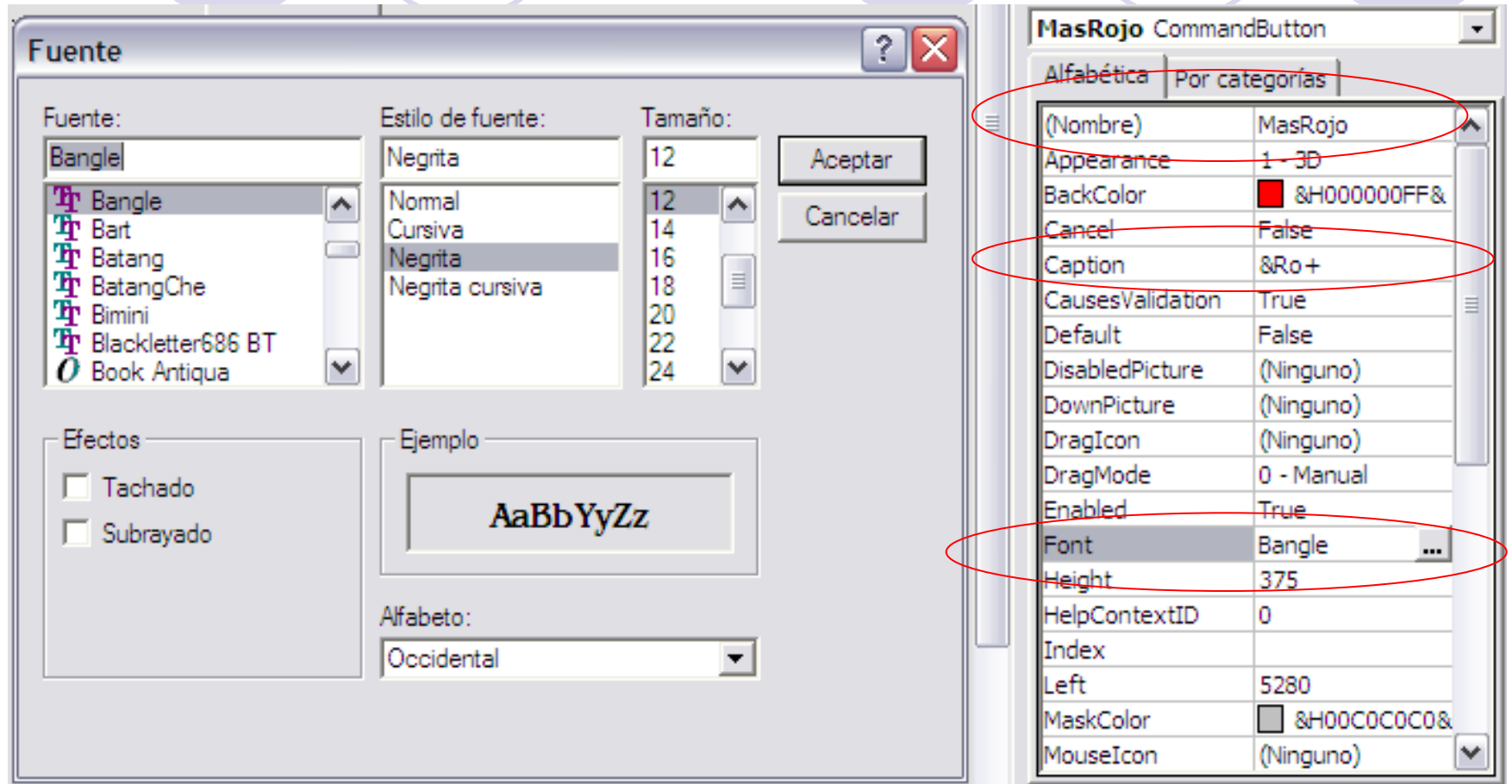
(Nombre)	Text1
Alignment	2 - Center
Appearance	1 - 3D
BackColor	&H80000005&
BorderStyle	1 - Fixed Single
CausesValidation	True
DataField	
DataFormat	
DataMember	
DataSource	
DragIcon	(Ninguno)
DragMode	0 - Manual
Enabled	True
Font	Bart
ForeColor	&H80000008&
Height	855
HelpContextID	0
HideSelection	True
Index	

Font

Devuelve Posición del formulario

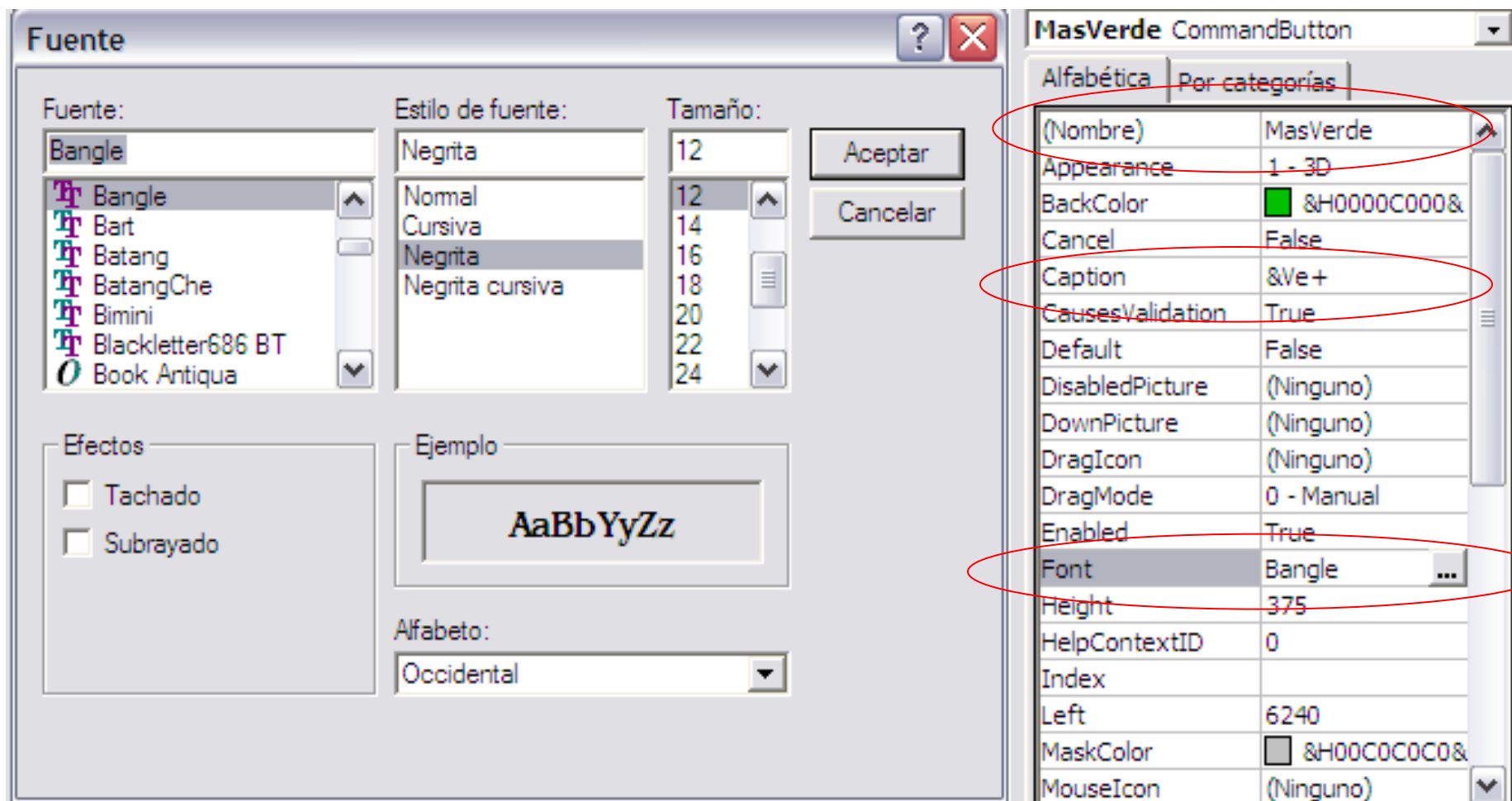
Visual Basic

Ejercicio 5



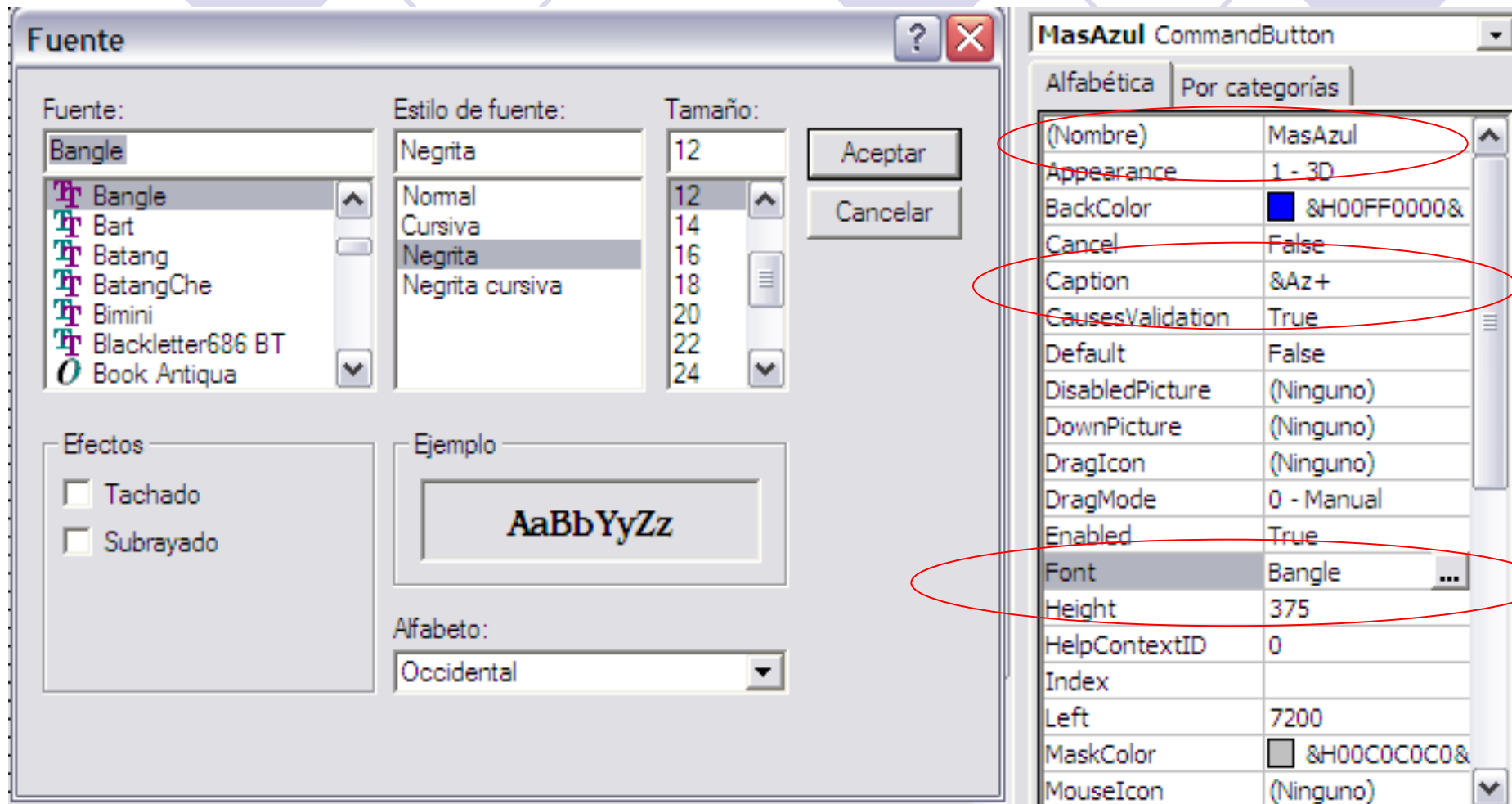
Visual Basic

Ejercicio 5



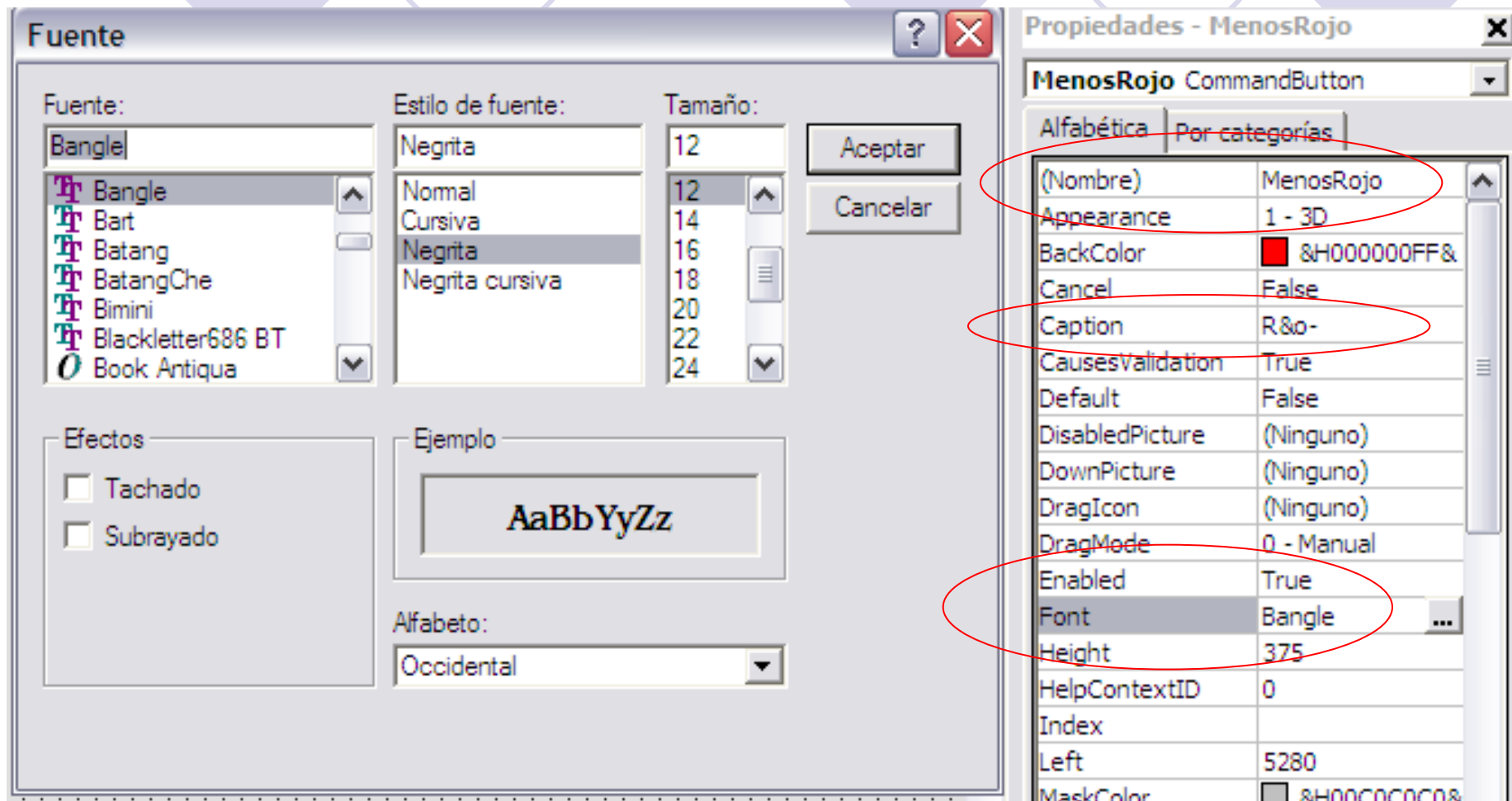
Visual Basic

Ejercicio 5



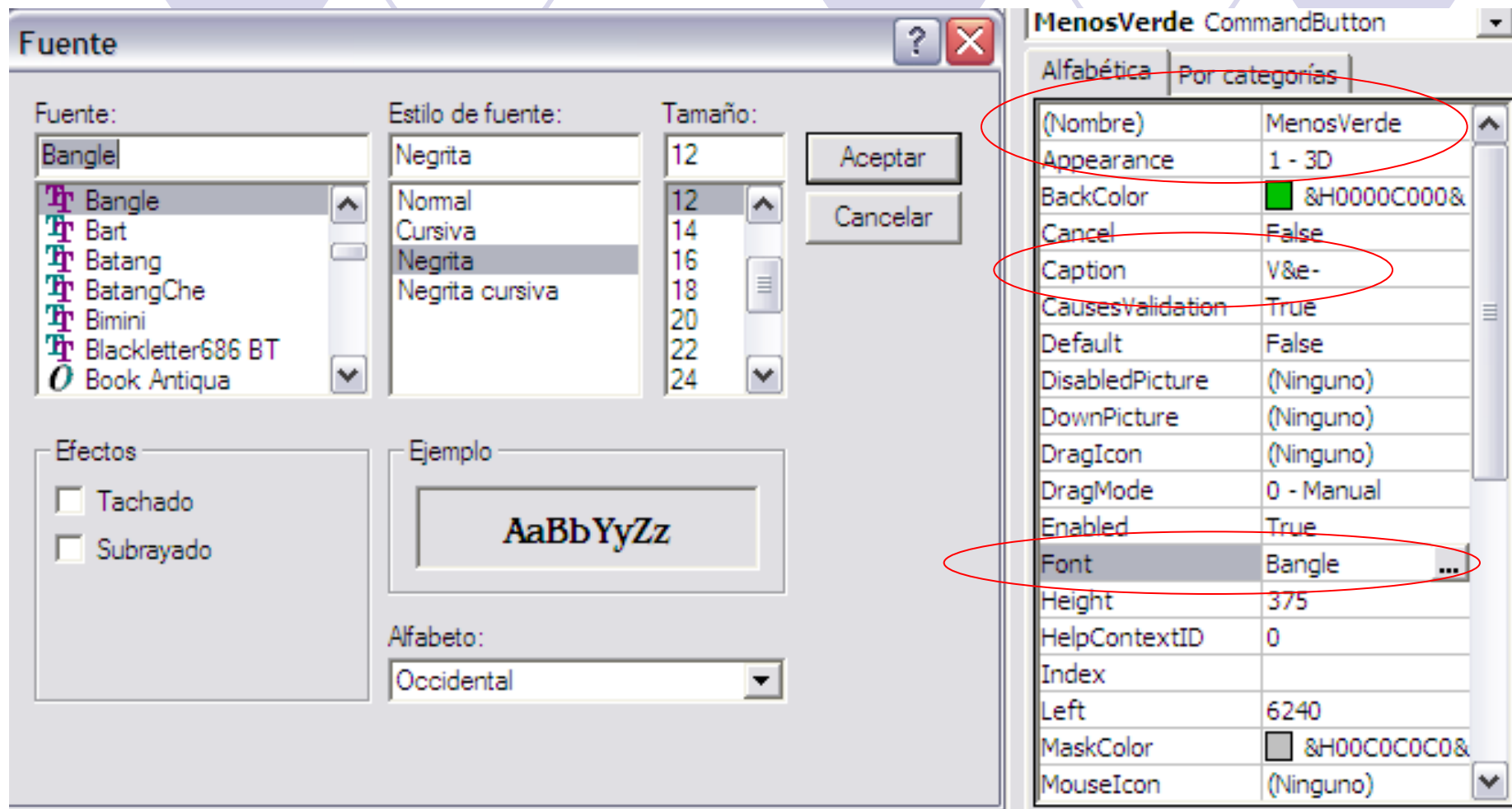
Visual Basic

Ejercicio 5



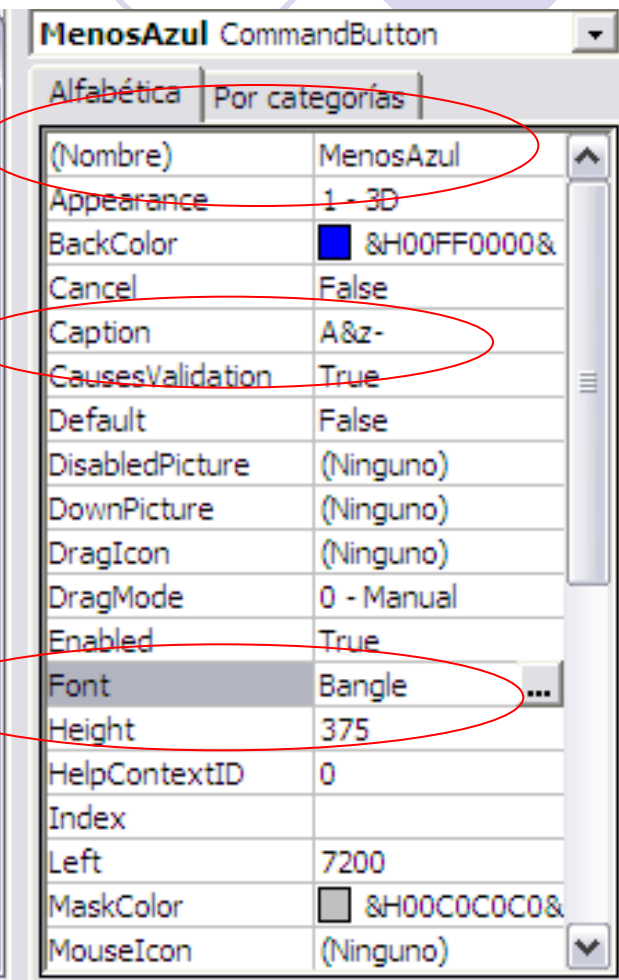
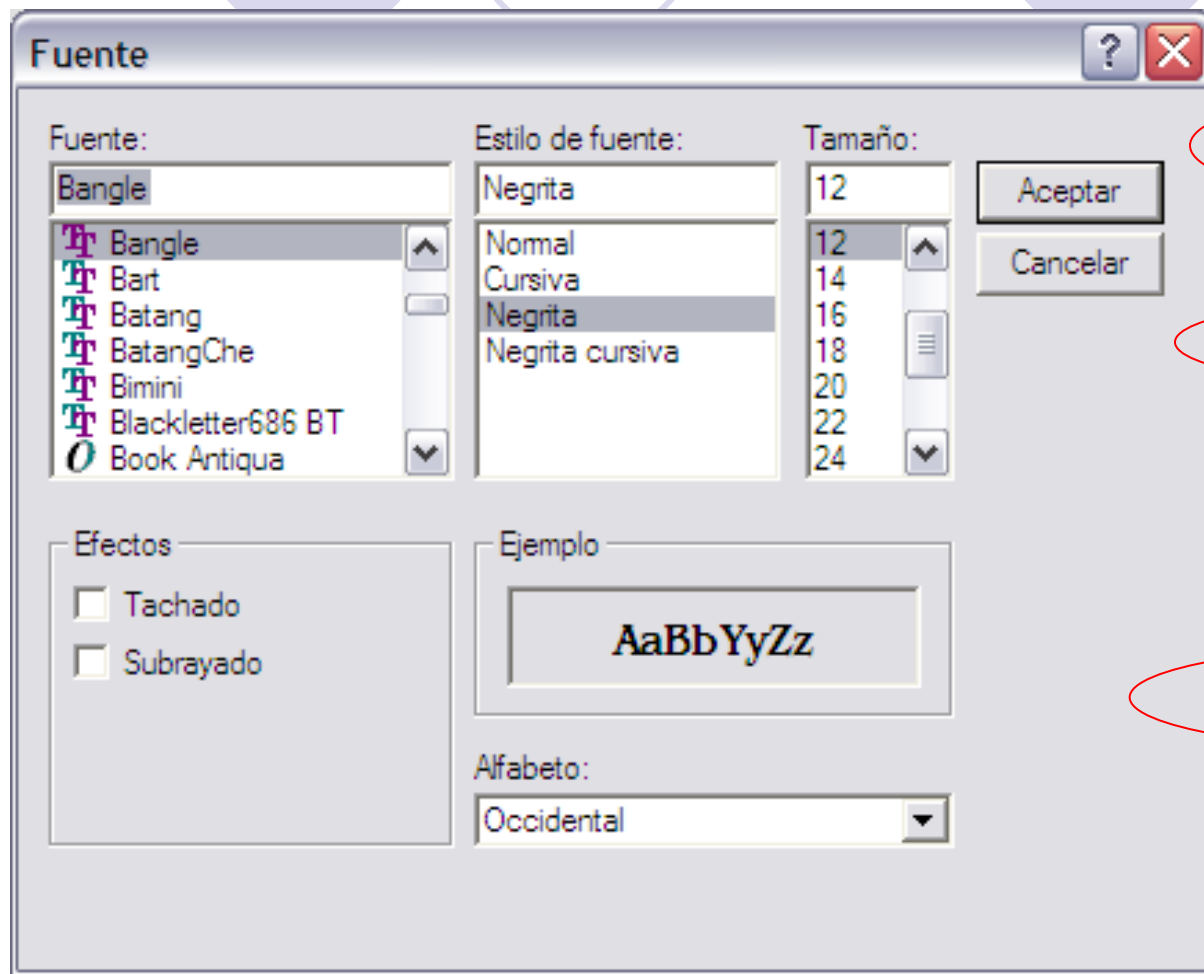
Visual Basic

Ejercicio 5



Visual Basic

Ejercicio 5



Visual Basic

Ejercicio 5

TextoRojo TextBox

Alfabética Por categorías

(Nombre)	TextoRojo
Alignment	2 - Center
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H80000005&
BorderStyle	1 - Fixed Single
CausesValidation	True
DataField	
DataFormat	
DataMember	
DataSource	
DragIcon	(Ninguno)
DragMode	0 - Manual
Enabled	True
Font	MS Sans Serif
ForeColor	<input type="checkbox"/> &H000000F
Height	375

TextoVerde TextBox

Alfabética Por categorías

(Nombre)	TextoVerde
Alignment	2 - Center
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H80000005&
BorderStyle	1 - Fixed Single
CausesValidation	True
DataField	
DataFormat	
DataMember	
DataSource	
DragIcon	(Ninguno)
DragMode	0 - Manual
Enabled	True
Font	MS Sans Serif
ForeColor	<input type="checkbox"/> &H0000C0C
Height	375

Propiedades - TextoAzul

TextoAzul TextBox

Alfabética Por categorías

(Nombre)	TextoAzul
Alignment	2 - Center
Appearance	1 - 3D
BackColor	<input type="checkbox"/> &H80000005&
BorderStyle	1 - Fixed Single
CausesValidation	True
DataField	
DataFormat	
DataMember	
DataSource	
DragIcon	(Ninguno)
DragMode	0 - Manual
Enabled	True
Font	MS Sans Serif
ForeColor	<input type="checkbox"/> &H00FF000
Height	375

TextoRojo TextBox

Alfabética Por categorías

MaxLength	0
MouseIcon	(Ninguno)
MousePointer	0 - Default
MultiLine	False
OLEDragMode	0 - Manual
OLEDropMode	0 - None
PasswordChar	
RightToLeft	False
ScrollBars	0 - None
TabIndex	1
TabStop	True
Tag	
Text	0
ToolTipText	

Propiedades - TextoVerde

TextoVerde TextBox

Alfabética Por categorías

MaxLength	0
MouseIcon	(Ninguno)
MousePointer	0 - Default
MultiLine	False
OLEDragMode	0 - Manual
OLEDropMode	0 - None
PasswordChar	
RightToLeft	False
ScrollBars	0 - None
TabIndex	4
TabStop	True
Tag	
Text	0
ToolTipText	

Propiedades - TextoAzul

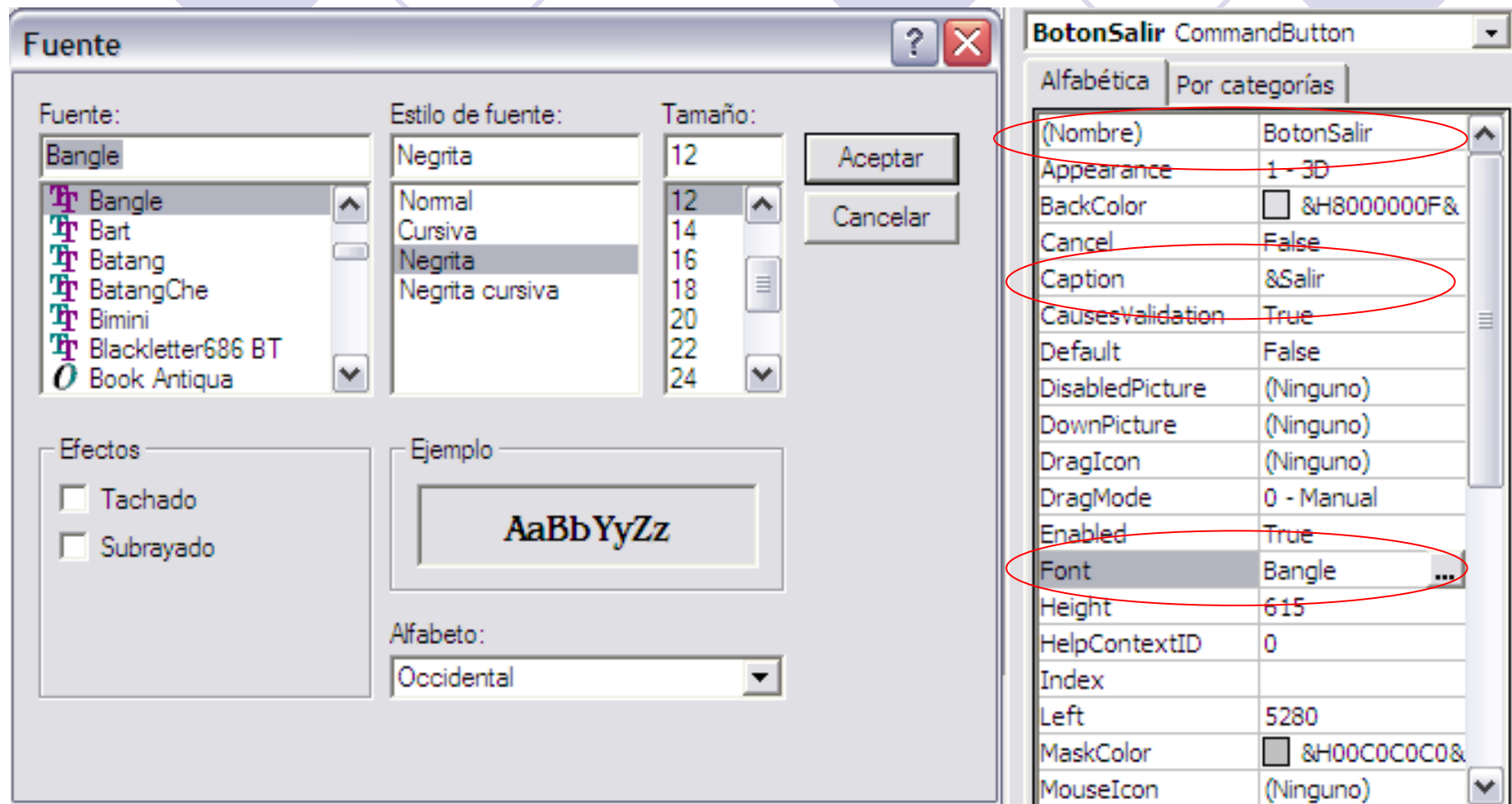
TextoAzul TextBox

Alfabética Por categorías

MaxLength	0
MouseIcon	(Ninguno)
MousePointer	0 - Default
MultiLine	False
OLEDragMode	0 - Manual
OLEDropMode	0 - None
PasswordChar	
RightToLeft	False
ScrollBars	0 - None
TabIndex	7
TabStop	True
Tag	
Text	0
ToolTipText	

Visual Basic

Ejercicio 5



```
Option Explicit
Dim Rojo As Integer, Verde As Integer, Azul As Integer
```

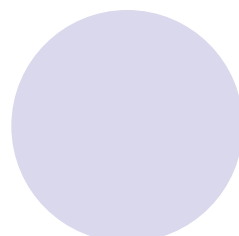
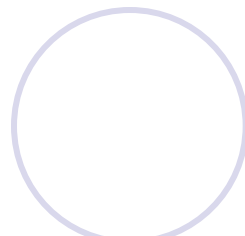
```
Private Sub BotonSalir_Click()
    End
End Sub
```


```
Private Sub MasRojo_Click()
    If (Rojo < 255) Then
        Rojo = Rojo + 1
        Text1.ForeColor = RGB(Rojo, Verde, Azul)
        TextoRojo.Text = CStr(Rojo)
    End If
End Sub
```

```
Private Sub MenosRojo_Click()
    If (Rojo > 0) Then
        Rojo = Rojo - 1
        Text1.ForeColor = RGB(Rojo, Verde, Azul)
        TextoRojo.Text = CStr(Rojo)
    End If
End Sub
```

```
Private Sub MasVerde_Click()
    If (Verde < 255) Then
        Verde = Verde + 1
        Text1.ForeColor = RGB(Rojo, Verde, Azul)
        TextoVerde.Text = CStr(Verde)
    End If
End Sub
```

```
Private Sub MenosVerde_Click()
    If (Verde > 0) Then
        Verde = Verde - 1
        Text1.ForeColor = RGB(Rojo, Verde, Azul)
        TextoVerde.Text = CStr(Verde)
    End If
```





```
Private Sub MasAzul_Click()  
    If (Azul < 255) Then  
        Azul = Azul + 1  
        Text1.ForeColor = RGB(Rojo, Verde, Azul)  
        TextoAzul.Text = CStr(Azul)  
    End If  
End Sub
```

```
Private Sub MenosAzul_Click()  
    If (Azul > 0) Then  
        Azul = Azul - 1  
        Text1.ForeColor = RGB(Rojo, Verde, Azul)  
        TextoAzul.Text = CStr(Azul)  
    End If  
End Sub
```

Visual Basic

Ejercicio 6

- **Vamos a cambiar el color de texto a Negro, Azul, Azul Celeste, Verde, Violeta, Rojo, Blanco y Amarillo.**

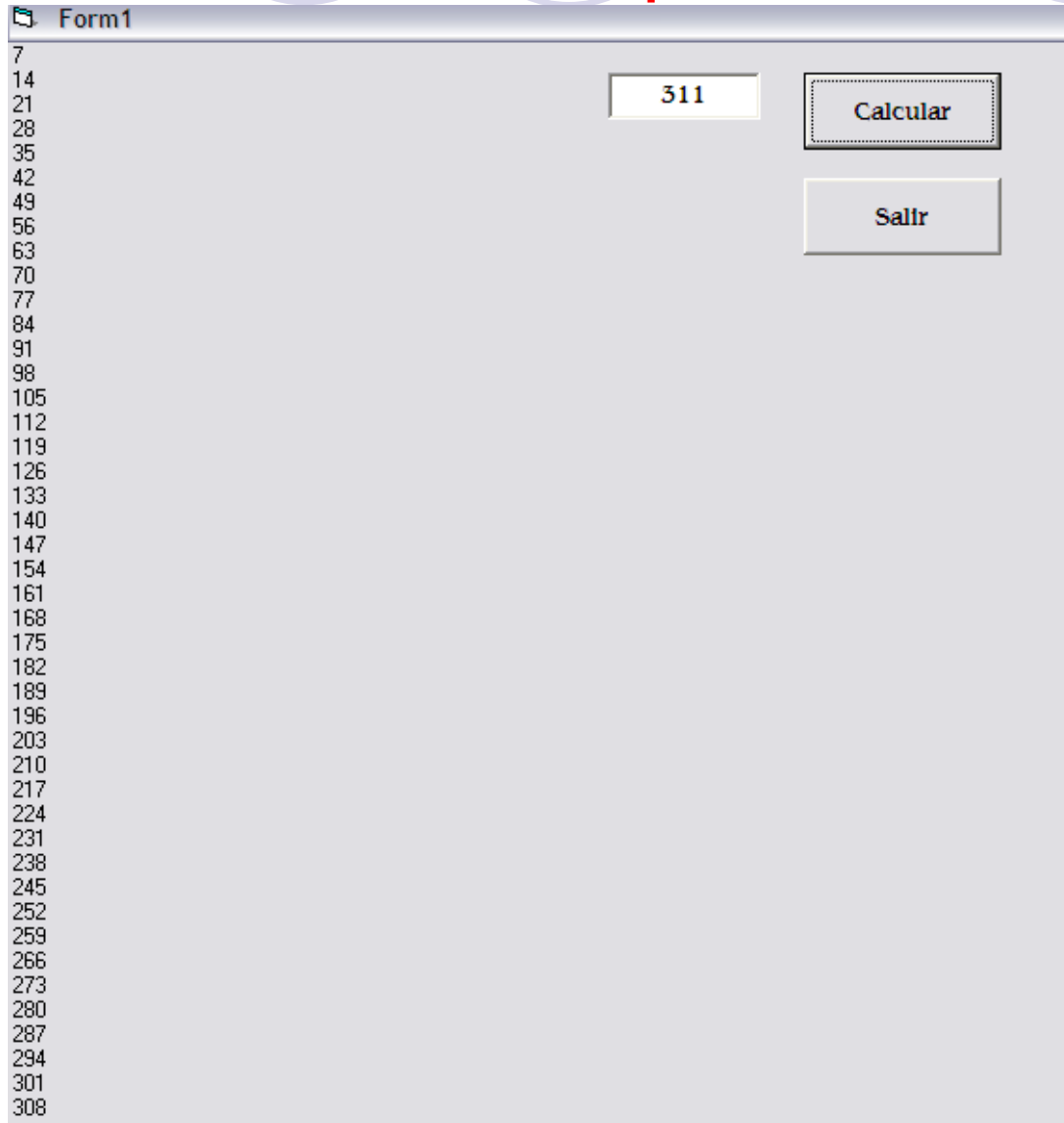
The image shows a screenshot of a Visual Basic application window titled "Form1". Inside the window, there is a text box on the left containing the text "Texto de Prueba" in a bright green color. To the right of the text box is a grid of buttons for selecting different text colors. The buttons are arranged in two columns. The first column contains buttons for "Negro", "Azul", "Azul Celeste", "Verde", and "Salir". The second column contains buttons for "Violeta", "Rojo", "Blanco", and "Amarillo". The "Verde" button is currently selected, indicated by a dotted border around it. The text "Texto de Prueba" is currently green, matching the selected "Verde" button.

Negro	Violeta
Azul	Rojo
Azul Celeste	Blanco
Verde	Amarillo
Salir	

Visual Basic

Ejercicio 7

- **Obtener los divisores por 7 desde 1 hasta un valor introducido.**



Form1

7
14
21
28
35
42
49
56
63
70
77
84
91
98
105
112
119
126
133
140
147
154
161
168
175
182
189
196
203
210
217
224
231
238
245
252
259
266
273
280
287
294
301
308

311

Calcular

Salir

Visual Basic

Ejercicio 8

- **Vamos a ir introduciendo números. Cuando pulsemos “*Calcular*” si el número es par se sumará al valor que hay en el recuadro de la derecha y, si es impar, al de la izquierda. El proceso acaba cuando metemos el valor 77.**

The screenshot shows a Visual Basic form titled "Form1". The form has a light gray background. In the center, there is a rectangular box containing the number "77". To the right of this box is a button labeled "Calcular". Below the "77" box, there are two more rectangular boxes, one containing the number "8" on the left and one containing the number "9" on the right.

Visual Basic

Ejercicio 9

- **Vamos a jugar a las “*máquinas tragaperras*”. Se trata de obtener, de forma aleatoria, el mismo número en los 4 cuadros de texto. Si lo conseguimos, imprimiremos el literal: “HAS GANADO”.**

Form1

HAS GANADO

9	9	9	9
Pulsa	Pulsa	Pulsa	Pulsa
Salir			