

Tema 2: Elementos del lenguaje

- ☐ Identificadores.
- ☐ Estructura de un programa .NET.
- ☐ Tipos de datos.
- ☐ Estructuras.
- ☐ Literales.
- ☐ Declaración de variables.
- ☐ Constantes y enumeraciones.
- ☐ Operadores y expresiones.
- ☐ Conversión de tipos.
- ☐ Instrucciones, líneas y bloques.

Identificadores

- ❑ Caracteres.
 - Utiliza caracteres Unicode de 16 bits.
- ❑ Identificadores.
 - De 1 a 16.386 caracteres Unicode.
 - ✓ Caracteres alfabéticos, numéricos o el carácter de subrayado.
 - Debe empezar por un carácter alfabético o el subrayado.
 - ✓ Si comienza por el subrayado debe tener al menos otro carácter alfabético o numérico.
- ❑ Comentarios.
 - Cualquier texto que aparezca después del apóstrofo (') es ignorado por el compilador.
 - Sentencia REM.
 - Visual Studio permite añadir comentarios XML en el editor de código utilizando tres apóstrofes (""").

Identificadores (II)

❑ Palabras reservadas. No se pueden utilizar como identificador.

AddHandler	AddressOf	Alias	And	AndAlso	As	Boolean	ByRef
Byte	ByVal	Call	Case	Catch	CBool	CByte	CChar
CDate	CDec	CDbl	Char	Clnt	Class	CLng	CObj
Const	Continue	CSByte	CShort	CSng	CStr	CType	CUInt
CULng	CUShort	Date	Decimal	Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each	Else	Elseif	End	EndIf
Enum	Erase	Error	Event	Exit	False	Finally	For
Friend	Function	Get	GetType	Global	GoSub	GoTo	Handles
If	Implements	Imports	In	Inherits	Integer	Interface	Is
IsNot	Let	Lib	Like	Long	Loop	Me	Mod
Module	MustInherit	MustOverride	MyBase	MyClass	Namespace	Narrowing	New
Next	Not	Nothing	NotInheritable	NotOverridable	Object	Of	On
Operator	Option	Optional	Or	OrElse	Overloads	Overridable	Overrides
ParamArray	Partial	Private	Property	Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume	Return	SByte	Select	Set
Shadows	Shared	Short	Single	Static	Step	Stop	String
Structure	Sub	SyncLock	Then	Throw	To	True	Try
TryCast	TypeOf	Variant	Wend	UInteger	ULong	UShort	Using
When	While	Widening	With	WithEvents	WriteOnly	Xor	#Const
#Else	#Elseif	#End	#If	-	&	&=	*
*=	/	/=	\	\=	^	^=	+
+=	=	-=					

Estructura de un programa VB .NET

- ❑ Una aplicación VB.NET se almacena en uno o más archivos de proyecto.
 - Cada proyecto consta de uno o más archivos de código (módulos) que se compilan para crear aplicaciones.

- ❑ Las categorías de instrucciones dentro de cada módulo deben seguir este orden:
 1. Instrucciones Option.
 2. Instrucciones Imports.
 3. Instrucciones Namespace.
 4. Declaraciones de módulos o clases.

Estructura de un programa VB.NET (II)

☐ Instrucciones Option.

- Establecen las reglas base del código que aparece en el archivo.
- Option Explicit, Option Compare, Option Strict.

☐ Instrucciones Imports.

- Facilitan el empleo de los espacios de nombres y clases dentro del código del archivo.
- Permiten evitar la referencia por el nombre cualificado.

```
Imports System.Text
...
Dim sb1 As New System.Text.StringBuilder(20) 'Nombre cualificado
Dim sb2 As New StringBuilder(30)
```

Estructura de un programa VB.NET (III)

☐ Declaraciones.

- Todo el código ejecutable (declaraciones, métodos, procedimientos, funciones) debe estar contenido en una clase o un módulo.

✓ La clase o módulo puede estar dentro de un espacio de nombres.

☐ La mayoría de las veces el módulo o clase deberá tener un método main.

- Es el punto de entrada de la aplicación.

Sub Main()

Sub Main(ByVal CmdArgs() As String)

Function Main() As Integer

Function Main(ByVal CmdArgs() As String) As Integer

Tipos de datos

❑ Tres categorías.

● Tipos de valores.

- ✓ Tipos primitivos (tipos de datos predefinidos o tipos valor integrados).
 - Numéricos, reales, lógicos, caracteres.
- ✓ Enumeraciones.
- ✓ Estructuras.

● Tipos de referencia.

- ✓ Cadenas, arrays, clases, módulos estándar, interfaces y delegados.

● Tipo Object.

- ✓ Alias de la clase System.Object.
- ✓ De ella descienden todos los tipos.
- ✓ Puede contener cualquier tipo de dato.

Tipos de datos (II)

❑ Almacenamiento en memoria.

- Los tipos de valores se almacenan en la pila.
 - ✓ Se crean y reservan en tiempo de compilación.
 - ✓ Su acceso es directo.
- Los tipos de referencia se almacenan en el montículo.
 - ✓ Son dinámicos, se guardan en tiempo de ejecución.
 - ✓ El acceso se hace a través de una referencia.
 - Cuando la referencia se pierde no se puede acceder al dato.
- La asignación a un dato de referencia copia la referencia, no su contenido.

Tipo de dato: Entero.

- ☐ SByte.
 - Corresponde al tipo del CLR System.SByte.
 - Ocupa 1 byte.
 - Puede tomar los valores del rango -128 a 127.
- ☐ Byte.
 - Corresponde al tipo del CLR System.Byte.
 - Ocupa 1 byte.
 - Puede tomar los valores del rango 0 a 255.
- ☐ UShort.
 - Corresponde al tipo del CLR System.UInt16.
 - Ocupa 2 bytes.
 - Puede tomar los valores del rango 0 a 65.536.
- ☐ Short.
 - Corresponde al tipo del CLR System.Int16.
 - Ocupa 2 bytes.
 - Puede tomar los valores del rango -32768 a 32767.

Tipo de dato: Entero (II).

☐ UInteger.

- Corresponde al tipo del CLR System.Int32.
- Ocupa 4 bytes.
- Puede tomar los valores del rango 0 a 4.294.967.295

☐ Integer.

- Corresponde al tipo del CLR System.Int32.
- Ocupa 4 bytes.
- Puede tomar los valores del rango -2.147.483.648 a -2.147.483.647

☐ ULong.

- Corresponde al tipo del CLR System.Int64.
- Ocupa 8 bytes.
- Puede tomar los valores del rango 0 a 18.446.744.073.709.551.615.

☐ Long.

- Corresponde al tipo del CLR System.Int64.
- Ocupa 8 bytes.
- Puede tomar los valores del rango -9.223.372.036.854.775.808 a
9.223.372.036.854.775.807

Tipo de dato: Real.

- ☐ **Single** (numero en coma flotante de simple precisión).
 - Corresponde al tipo del CLR System.Single.
 - Ocupa 4 bytes.
 - Puede tomar los valores del rango

Negativos de -3.4028235E+38 a -1.401298E-45
Positivos de 1.401298E-45 a 3.4028235E+38

- ☐ **Double** (numero en coma flotante de doble precisión).
 - Corresponde al tipo del CLR System.Double.
 - Ocupa 8 bytes.
 - Puede tomar los valores:

Negativos de -1.79769313486231570E+308 a -4.94065645841246544E-324
Positivos de 4.94065645841246544E-324 a 1.79769313486231570E+308

- ☐ **Decimal**.
 - Corresponde al tipo del CLR System.Decimal.
 - Ocupa 16 bytes.
 - Puede tomar los valores:

de 0 a +/-79,228,162,514,264,337,593,543,950,335 sin decimales;
de 0 a +/-7.9228162514264337593543950335 con 28 lugares a la derecha del decimal.

Tipos de datos: Boolean y Char

☐ El tipo de dato Boolean.

- Corresponde al tipo del CLR System.Boolean.
- Ocupa 2 bytes.
- Puede tomar el valor True o False.

☐ El tipo de dato Char.

- Corresponde al tipo del CLR System.Char.
- Ocupa 2 bytes (un carácter Unicode de 16 bits).
- No es compatible con el dato String.
- No se puede utilizar como un dato numérico.
 - ✓ Funciones Asc(), AscW(), Chr() y ChrW().
- Métodos estáticos
Char.IsControl(), Char.IsDigit(), Char.IsLetter(), Char.IsDigitOrLetter(),
Char.IsLower(), Char.IsNumber(), Char.IsPunctuation(), Char.IsSymbol(),
Char.Upper(), Char.IsWhiteSpace().

Tipo de dato: Date

☐ Características.

- Corresponde a la clase System.DateTime del CLR
- Ocupa 8 bytes (un entero largo).
- Puede tomar el valor de fechas y horas desde el 1 de enero del año 1 a las 00:00:00 hasta el 31 de diciembre del año 9.999 a las 11:59:59.

☐ Constructor: System.DateTime(año,mes,día).

☐ Permite el acceso a los diferentes elementos de una fecha.

Propiedad	Devuelve...	Propiedad	Devuelve...
Day	El día	Hour	Horas
Month	El mes	Minute	Minutos
Year	El año	Second	Segundo
DayOfWeek	El día de la semana	Millisecon	Milisegundos
DayOfYear	El día del año (0 para el domingo)	Today	La fecha actual
TimeOfDay	La hora del día	Now	La fecha y hora actual
Ticks	Número de <i>ticks</i> de una fecha		

Tipos de datos: el tipo Date (II)

☐ Aritmética de fechas.

- AddDays(n).
- AddMonths(n).
- AddYears(n).
- AddHours(n).
- AddMinutes(n).
- AddSeconds(n).

☐ n puede ser un número entero o fraccionario, positivo o negativo.

La clase String

- ❑ Características.
 - Corresponde a la clase System.String del CLR.
 - Permite almacenar de 0 a 2.000 mill. de caracteres Unicode.
- ❑ No es un dato valor ni un dato primitivo, sino una clase.
 - Es una referencia que apunta a una zona del montículo.
 - Ocupará aproximadamente 4 bytes de la referencia más el doble del número de caracteres.
- ❑ Al ser una clase dispone de una serie de propiedades y métodos para la creación y manipulación de cadenas de texto.
- ❑ Por ser una clase en la declaración de tipo de variable se instancia a través de un constructor.

La clase String (II)

☐ Constructores.

- `String(Char,Integer)`; `String(Char())`; etc.

☐ Algunas propiedades públicas.

- `Length`. Devuelve el número de caracteres.
- `Chars(Integer)`. Devuelve el carácter de la posición `Integer` en base 0.

☐ Algunos métodos públicos (contenidos en la clase `String` del CLR).

- `Replace`, `Split`, `Unicode`, `ToString`, `ToLower`, `ToUpper`, `Trim`, `TrimEnd`, `TrimStart` ...

☐ Existen funciones de cadena exclusivas de VB.NET, no de .NET Framework. Al no estar incluidas en la biblioteca de .NET Framework es preferible utilizar los métodos públicos.

El tipo de dato Object

- ❑ Todos los tipos derivan de la clase System.Object, por lo que con un dato de tipo Object se puede referenciar cualquier tipo de dato.
- ❑ Ocupa 4 bytes.
 - Lo que ocupa un puntero.
- ❑ Todos los datos heredan los métodos de Object.
 - Los métodos se pueden sobrecargar para dotarlos de características especiales.
 - ✓ Método Equals().
 - ✓ Método ToString().
- ❑ Al trabajar con referencias de objetos, algunas veces deberemos convertir el objeto genérico al tipo de objeto concreto del CLR

Estructuras

- ❑ Tipos de datos definidos por el usuario.
- ❑ Pueden contener cualquier tipo de dato (valores o referencias) y métodos para manipularlos.
- ❑ Declaración:

```
[Modificador de acceso]Structure NombreTipo
    declaración de variables del tipo
    [declaración de métodos]
End Structure

Structure Empleado
    Public DNI As String
    Public Nombre As String
    Public SueldoBruto As Decimal
    Public Retenciones As Decimal
    Public Function SueldoNeto() As Decimal
        Return SueldoBruto - Retenciones
    End Function
End Structure
```

- ❑ Se accede a los miembros mediante el punto.

Literales

- ☐ Booleanos.
 - True y False.
- ☐ Enteros.
 - Dígitos y, opcionalmente el signo -.
 - El prefijo &H indica un valor hexadecimal. El prefijo &O indica un valor octal.
 - Pueden incluir un sufijo para indicar el tipo de entero: S,I,L,US, ,UI o UL.
- ☐ Reales.
 - Dígitos y opcionalmente el punto decimal y el signo.
 - Formato en coma fija o coma flotante.
 - Pueden incluir un sufijo para indicar el tipo de real: D, F o R

Literales (II)

☐ Fecha.

- Fecha en formato mm/dd/aaaa encerrada entre almohadillas (#).

☐ Literales de cadena y carácter.

- Secuencia de 0 o más caracteres Unicode entre comillas.
- Pueden incluir comillas mediante el carácter de escape “.
- Los literales de tipo carácter tienen el sufijo c.

☐ El literal Nothing.

- Inicializa cualquier dato con su valor por omisión.
- Cuando se le asigna a una variable de tipo objeto o a una clase, se elimina la referencia.

Declaraciones de variables

❑ Declaración implícita y explícita.

- Instrucción Option Explicit [{On | Off}].

❑ Declaración de variables a nivel de módulo: variables locales.

- [Static] Dim *ident* [As [New] *TipoDato*] [=*exprInic*]
 - ✓ Su accesibilidad es dentro del procedimiento.
 - ✓ El modificador Static determina el tiempo de vida de la variable.
 - ✓ El tipo de dato no es obligatorio. Por omisión es de tipo Object.
 - ✓ Para variables de tipos de referencia se puede utilizar la palabra reservada New para crear una nueva instancia de la clase.
 - ✓ Admiten una expresión de inicialización del tipo de dato adecuado.

Declaraciones de variables (II)

❑ Declaración a nivel de módulo o clase.

```
[{Public | Protected | Friend |  
Protected Friend | Private } [Static |Shared]  
Dim ident [As [New] tipoDato] [=exprInic]
```

- Public, Protected, Friend, Protected Friend y Private son los modificadores de acceso.
 - ✓ Public permite acceder a la variable desde otro proyecto o desde cualquier ensamblado del proyecto.
 - ✓ Friend permite el acceso desde cualquier parte del mismo ensamblado.
 - ✓ Private permite acceder sólo desde dentro del ámbito donde ha sido declarada.
 - ✓ Protected, Protected Friend y Shared se utilizan para la POO.
- Si se utiliza cualquier de estos modificadores se puede omitir la palabra Dim.

Constantes y enumeraciones

❑ Constantes.

```
{ { Public | Protected | Friend | Protected Friend  
|  
Private } } Const identificador [ As tipoDatos ] = ExprInic
```

❑ Enumeraciones.

- Proporcionan una forma cómoda de trabajar con constantes relacionadas.
- Declaración:

```
{ { Public | Protected | Friend | Protected Friend  
|  
Private } } Enum nombreEnumeración [As tipoDato]  
nombreMiembro [= exprInic]  
...  
End Enum
```

Constantes y enumeraciones (II)

❑ Enumeraciones (*continuación*).

- El valor de las constantes *nombreMiembro* es de algún tipo entero.
 - ✓ Por omisión son de tipo Integer y comienzan a contar desde 0.
 - ✓ Para cambiar el tipo de dato se puede utilizar la cláusula *As tipoDato*.
 - ✓ La *exprInic* permite asignar a la constante enumerada un valor distinto.

```
Public Enum DiaSemana
    Domingo 'La constante DiaSemana.Domingo vadrá 0
    Lunes
    Martes
    Miércoles
    Jueves
    Viernes
    Sábado 'La constante DiaSemana.Sábado vadrá 6
End Enum
```


Operadores y expresiones

☐ Operadores aritméticos.

- ^, exponenciación.
 - ✓ Los operandos se convierten a Double y el resultado es Double.
- *, +, -.
 - ✓ El resultado depende de los operandos y será del tipo del operando de mayor intervalo (Byte, Short, Integer, Long, Single, Double y Decimal).
 - ✓ El signo + se puede utilizar para la suma o para la concatenación.
- División entera (\) y módulo (Mod).
 - ✓ Los operandos se convierten a entero y el resultado es entero.

☐ Operadores de concatenación.

- + y &.

Operadores y expresiones (II)

❑ Operadores de asignación.

- =, +=, -=, *=, /=, \=, ^=, &=.

Operadores de relación.

- =, <>, >=, <=, <, >.

● Comparaciones de cadenas.

- ✓ Comparación binaria y comparación de texto:

- Instrucción Option Compare {Binary | Text}.

- ✓ La función StrComp.

- StrComp(String1,String2[, {Binary|Text}]).

- Devuelve 0, 1 o -1 según la cadena 1 sea igual, mayor o menor que la cadena 2.

- El tercer argumento permite especificar el tipo de comparación.

- ✓ Método Compare(String1,String2[,Boolean]).

- ✓ Método CompareTo(String1).

- Ambas devuelven un valor igual, menor o mayor que 0, según la cadena sea igual, menor o mayor que la otra.

- Si el tercer argumento de Compare es True ignora mayúsculas y minúsculas.

Operadores y expresiones (III)

❑ El operador Like.

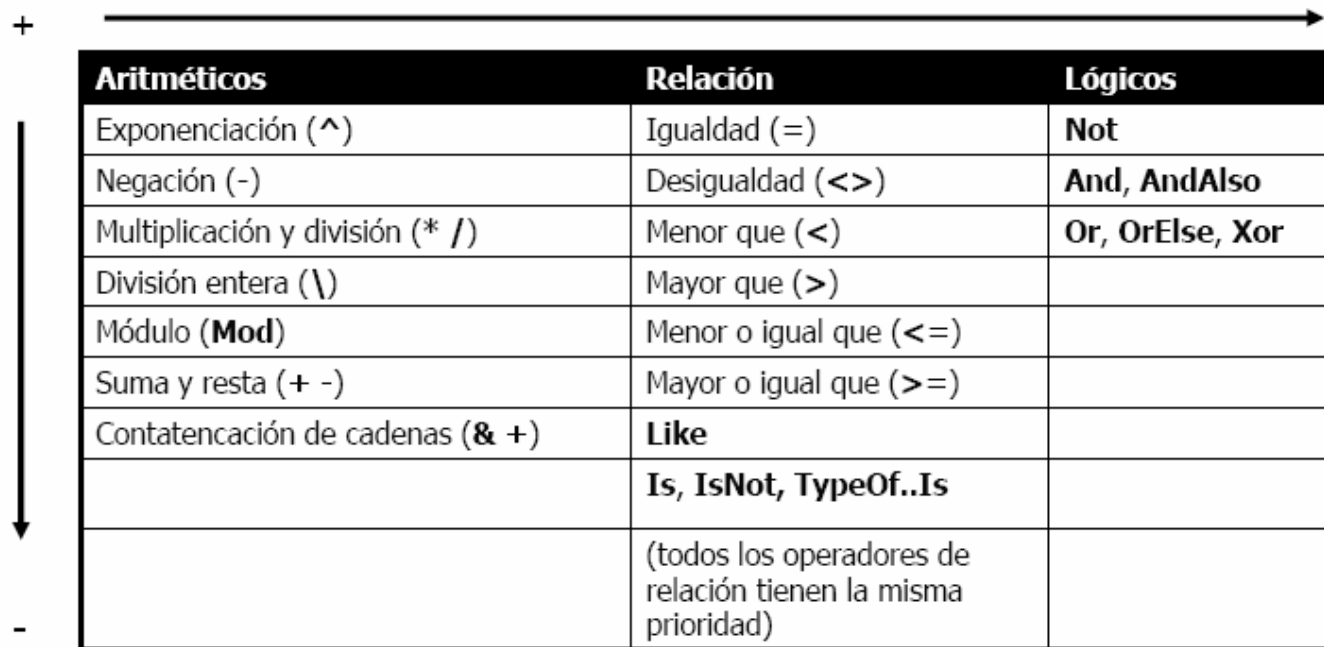
- Compara una cadena con un patrón.
 - ✓ *expresiónCadena* Like *patrón*
- patrón es una cadena que puede contener:
 - ✓ *, sustituye a 0 o más caracteres ("hola" Like "ho*").
 - ✓ ?, sustituye a 1 carácter ("hola" Like "ho?a").
 - ✓ #, sustituye a un dígito ("1234A" Like "####*").
 - ✓ [caracteres], sustituye a alguno de los caracteres ("hola" Like "hol[aeiou]").
 - ✓ [car1-car2], sustituye a alguno de los caracteres del rango ("A3" Like "A[0-5]").
 - ✓ [!caracteres], sustituye a cualquier carácter no incluido en la lista.

❑ Operador Is.

- Compara si dos referencias de objetos apuntan a la misma instancia.
- No compara el contenido. Para eso habría que sobrecargar el método Equals.

Operadores y expresiones (IV)

- ☐ Operador IsNot.
 - Compara si dos referencias de objetos no apuntan a la misma instancia.
- ☐ Operadores lógicos.
 - Not, And, Or, AndAlso, OrElse, Xor.
- ☐ Prioridad de los operadores.



Aritméticos	Relación	Lógicos
Exponenciación (^)	Igualdad (=)	Not
Negación (-)	Desigualdad (<>)	And, AndAlso
Multiplicación y división (* /)	Menor que (<)	Or, OrElse, Xor
División entera (\)	Mayor que (>)	
Módulo (Mod)	Menor o igual que (<=)	
Suma y resta (+ -)	Mayor o igual que (>=)	
Concatenación de cadenas (& +)	Like	
	Is, IsNot, TypeOf..Is	
	(todos los operadores de relación tienen la misma prioridad)	

Conversión de tipos

- ❑ Conversiones de ampliación y restricción.
 - Conversiones de ampliación: el tipo de dato receptor tiene más ocupación en memoria.
 - Conversiones de restricción: el tipo de dato receptor tiene menos ocupación en memoria.
 - ✓ Posible pérdida de precisión.
- ❑ Conversión implícita y conversión explícita.
 - Por omisión los tipos de datos se convierten siempre que sea necesario.
 - Instrucción Option Strict {On | Off}.
 - ✓ Restringe la conversión a conversiones de ampliación.
 - ✓ En caso de realizar conversiones de restricción produce errores en tiempo de compilación

Conversión de tipos

❑ Funciones de conversión.

[illegible]

Conversiones de tipos (III)

❑ Función CType(*expresión, tipo*).

- Permite convertir una expresión a un tipo de datos, objeto, estructura, clase o interfaz.

```
Dim num1 As Long  
Dim num2 As Single  
num1 = 1000  
num2 = CType(num1, Single) 'num2 será 1000.0
```

❑ Función DirectCast(*expresión, tipo*).

- Funciona de forma similar a CType.
- Requiere que el tipo especificado sea el mismo que el de la expresión o de un tipo heredado.
- Es más rápida de CType.
- Devuelve una excepción si no se puede realizar la conversión.

❑ Función TryCast(*expresión, tipo*).

- Devuelve Nothing si la conversión no se puede realizar.

Entrada/salida

❑ Entrada/salida por consola.

- `System.Console.ReadLine()`.
- `System.Console.Write()`.
- `System.Console.WriteLine()`.
 - ✓ Escribe una representación en formato cadena del argumento, seguida de un salto de línea.
- `WriteLine()` permite la salida de varios datos, concatenándolos para crear un objeto de tipo cadena:
`Console.WriteLine("Hola" & 23 & date.now())`
- `WriteLine(cadenaFormato, objetos)`, escribe el objeto u objetos según la cadena de formato.
 - ✓ La cadena de formato puede llevar secuencias {nº argumento} para indicar la posición que debe llevar el objeto.
`Console.WriteLine("Nombre:{1} Edad:{0}", 24, "Ana López")`

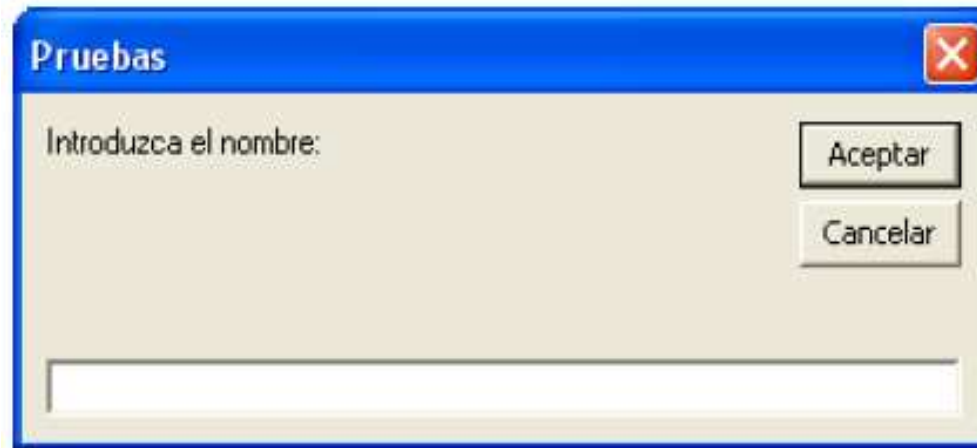
Función InputBox

❑ La entrada de datos simples de Windows se puede hacer con la función InputBox de Visual Basic.

- Muestra un cuadro de diálogo con una caja de texto en la que el usuario puede introducir información.

`InputBox(mensaje[, título][, valorOmisión])`

- Devuelve un dato de tipo String.
- En general se desaconseja su uso como elemento de la interfaz de entrada de datos.



La clase MessageBox

- ❑ Se puede utilizar para la salida de mensajes en un entorno de ventanas.
- ❑ .NET Framework dispone de la clase MessageBox que se utiliza para mostrar cuadros de diálogo por pantalla.
 - Está en el espacio de nombres y en el ensamblado System.Windows.Forms.
 - ✓ En aplicaciones de consola es necesario agregar esa referencia (con aplicaciones para Windows no).
- ❑ El método estático Show muestra el cuadro de diálogo.
 - `MessageBox.Show(texto[, título [, botones [, icono[, botónDefecto]]])`
 - Devuelve un dato de tipo DialogResult:
 - ✓ Puede tomar los valores DialogResult.Abort, DialogResult.Cancel, DialogResult.Ignore, DialogResult.No, DialogResult.OK, DialogResult.Retry, DialogResult.Yes.

La clase MessageBox (II)

- ❑ *botones* es un miembro de la enumeración MessageBoxButtons.

Miembro	Descripción
AbortRetryIgnore	El cuadro de mensaje contiene los botones Abort, Retry y Ignore.
OK	El cuadro de mensaje contiene un botón OK.
OKCancel	El cuadro de mensaje contiene los botones OK y Cancel.
RetryCancel	El cuadro de mensaje contiene los botones Retry y Cancel.
YesNo	El cuadro de mensaje contiene los botones Yes y No.
YesNoCancel	El cuadro de mensaje contiene los botones Yes, No y Cancel.

- ❑ *botónDefecto* es un miembro de la enumeración MessageBoxDefaultButtons.

Miembro	Descripción
Button1	El primer botón del cuadro de mensaje es el botón predeterminado.
Button2	El segundo botón del cuadro de mensaje es el botón predeterminado.
Button3	El tercer botón del cuadro de mensaje es el botón predeterminado.

La clase MessageBox (III)

 *icono* es un miembro de la enumeración MessageBoxIcon.

Miembro	Descripción
Asterisk	El cuadro de mensaje contiene un símbolo que consiste en un letra 'i' minúscula en un círculo.
Error	El cuadro de mensaje contiene un símbolo que consiste en una X blanca en un círculo con fondo rojo.
Exclamation	El cuadro de mensaje contiene un símbolo que consiste en un signo de exclamación en un triángulo con fondo amarillo.
Hand	El cuadro de mensaje contiene un símbolo que consiste en una X blanca en un círculo con fondo rojo.
Information	El cuadro de mensaje contiene un símbolo que consiste en un letra 'i' minúscula en un círculo.
None	El cuadro de mensaje no contiene ningún símbolo.
Question	El cuadro de mensaje contiene un símbolo que consiste en un signo de interrogación en un círculo.
Stop	El cuadro de mensaje contiene un símbolo que consiste en una X blanca en un círculo con fondo rojo.
Warning	El cuadro de mensaje contiene un símbolo que consiste en un signo de exclamación en un triángulo con fondo amarillo.

La clase MessageBox (IV)



```
If MessageBox.Show("¿Desea actualizar el archivo?", _  
    "Prueba de MessageBox", _  
    MessageBoxButtons.YesNo, _  
    MessageBoxIcon.Question, _  
    MessageBoxDefaultButton.Button2) = DialogResult.Yes Then  
    'Acciones del botón "Si"  
Else  
    'Acciones del botón "No"  
End If
```

Instrucciones, líneas y bloques

- ❑ VB no tiene formato libre de línea de código.
 - El fin de línea forma parte de la sintaxis del lenguaje.
 - ✓ La siguiente instrucción no sería válida
`System.Console.WriteLine ("Hola" & " "`
`& "qué tal")`
 - Es posible extender una instrucción más de una línea de texto utilizando el carácter de continuación de línea (_).
 - ✓ Debe ir precedido de un espacio en blanco.
 - ✓ La siguiente instrucción sería válida
`System.Console.WriteLine("Hola" & " " _`
`& "qué tal")`
 - En una línea pueden ir varias instrucciones utilizando el carácter separador de instrucciones (:).
`a = 3 : b = 4 : c = 6`

Instrucciones, líneas y bloques (II)

❑ Un grupo de líneas de código forman un **bloque de código**

- Los bloques de código se agrupan en una o más líneas *etiquetadas*, aunque la etiqueta es opcional.

- ✓ Una etiqueta está formada de un número o un identificador seguido de 2 puntos.

- Se pueden utilizar para referenciar zonas del código (instrucciones On Error...Goto)