

1. Studio Net

1.1 Introducción.

Aplicación Windows, formularios.
Consola, no necesita formularios.
Clase, para crear una clase
ASP, aplicación de Vis. Stud. ASP
WEB, pagina WEB

Estos son los tipos de aplicaciones que se pueden crear en el entorno de programación de Studio Net. Esta plataforma integra los lenguajes de programación C++, C#, Java, y Visual Basic. Se puede desarrollar un programa en cualquiera de esos lenguajes de programación y que éste esté formado por código escrito en todos esos lenguajes.

Dicho código puede compartir datos, clases y módulos. La ejecución posterior de dicho programa se realiza con un ejecutable que es un resultado intermedio entre el objeto y el del procesador, ya que la compilación da como resultado un archivo que es interpretado por una máquina virtual, que en función del S. O. puede ser cambiada, lo que en teoría sería la solución para la portabilidad de una aplicación de uno a otro S. O.

Una de las novedades es la posibilidad de escribir aplicaciones que tengan como elemento de salida la consola del ordenador, y como entrada el teclado, sin necesidad de diseñar o crear formularios.

Es decir usar la ventana del Sistema.
En esas aplicaciones es posible utilizar la instrucción

```
Console.WriteLine("Mes de {0} ,su índice es {1}", Meses(X), X)
```

Donde {0} indica en el sitio en el que se insertará el contenido de la primera de las variables que se indican al final.

```
Dato = Console.ReadLine
```

Esta es la instrucción para la entrada de datos desde la consola usando el teclado.

Cuando creamos la aplicación, un nuevo proyecto, hemos de declarar en las propiedades del proyecto, cual es el elemento de arranque de la aplicación, en el caso de la de consola que es la que vamos a utilizar en los ejemplos del principio, hemos de indicar que es el Sub Main, en propiedades del proyecto.

Para la ejecución del programa conviene pulsar <Ctrl> + <F5>, de esa forma se ejecuta y se genera una pausa que permite ver el contenido de la ventana con el resultado.

1.2 Cambios

Visual Basic .NET cambia la forma de ofrecer varios elementos del lenguaje, principalmente para la interoperabilidad con Common Language Runtime. Se cambia el nombre y la clasificación de muchos elementos de Visual Basic 6.0, y se combinan con otros elementos de programación para Visual Basic .NET.

Ya no se admiten varios elementos, puesto que Common Language Runtime incluye funcionalidad que los hace innecesarios.

Para obtener información adicional sobre los cambios realizados en Visual Basic, incluidos los del entorno integrado de desarrollo (IDE), de funcionalidad Web, de proyectos, de formularios, de constantes y de los métodos **Circle**, **Line** y **Pset**, conviene ver:

Introducción a Visual Basic .NET para usuarios veteranos de Visual Basic.

Hay una tabla donde se muestran los elementos de programación que han cambiado y sus sustitutos.

Visual Basic 6.0	Visual Basic .NET	Ubicación del espacio de nombres Clase o la biblioteca de tiempo de ejecución
Date (Función) Date (Instrucción)	Now, Today	DateAndTime
Debug.Assert (Método)	Assert, Fail (Métodos)	System.Diagnostics (Espacio de nombres) Debug (Clase)
Debug.Print(Método)	Write, Writelf, WriteLine y WriteLinelf (Métodos)	System.Diagnostics (Espacio de nombres), Debug (Clase)
DoEvents(Función)	DoEvents Método	System.Windows.Form (Espacio de nombres) Application (Clase)
Move (Método)	SetBounds	
Now (Función)	Now(Propiedad)	
Option Base	No se admite.	
Time (función) Time (Instrucción)	TimeOfDay (Propiedad) DateTime (Estructura)	DateAndTime

1.3 A tener en cuenta.

La utilización de DefXXX en Studio Net no es posible.

El tipo de datos Variant desaparece, a cambio tenemos el Objeto que cumple su función.

No se puede usar

```
Dim Cadena as String * 20
```

Hay que usar

```
<VbFixedString(20)> Public Cadena as String
```

y solo dentro de una estructura de registro.

Las propiedades del tipo DataField, DataChanged, DataMember, DataFormat, DataSource han desaparecido en los objetos.

En VB6 se usaba un solo formulario MDI en una aplicación, y todos los demás formularios eran MDIChildren. Ahora en una aplicación pueden haber uno o varios formularios MDI, y en los formularios Children hay que indicar en la propiedad MdiParent el formulario MDI que lo contendrá.

Cuando se cierra el formulario MDI, se cierran todos sus formularios dependientes.

Todo lo referente a Scale no se puede usar, ScaleMode, ScaleTop, ScaleWidth, ScaleHeight, ScaleLeft, ScaleX, ScaleY.

Los métodos CurrentX, CurrentY, DrawMode, DrawStyle, DrawWidth no se pueden usar.

Ahora hay dos objetos, uno que se denomina Pen, al que se le asignan las propiedades para crear una línea, color, ancho etc., y otro Graphics, que realiza el dibujo de la forma con las características antes definidas.

El objeto PictureBox, ha perdido muchas de sus funcionalidades, pero a cambio han aparecido objetos directamente relacionados con la impresión, que aunque algo extraña, ha salido beneficiada.

2. Operadores.

2.1 Introducción.

En todo lenguaje de programación hay que manejar datos, información.

Los datos sirven para hacer operaciones de diverso tipo, para hacer estas operaciones es necesario disponer de operadores que indiquen el tipo de operación que deseamos hacer.

¿Cuáles?, por no alargarse demasiado podríamos resumirlo con lo de los habituales.

Hay alguno nuevo como es el AndAlso, OrElse.

2.2 Aritméticos

Operación	Operador
Suma	+
Resta	-
Multiplicación	*
División	/
División entera	\
Eleva potencia	^
Resto división	Mod

2.3 Lógicos.

AndAlso no evalúa el operador de la derecha en una expresión cuando la expresión de la izquierda ya es falso, o dicho de otro modo es más rápido que el AND tradicional.

OrElse, es igual pero al revés, si la expresión de la izquierda ya es cierta no evalúa la de la derecha.

Operación	Operador
Conjunción, Y	And
Disyunción, O	Or
Negación, No	Not
Xor	Xor
Y También	AndAlso
O Sino	OrElse

2.4 Relacionales.

Operación	Operador
Mayor	>
Menor	<
Igual	=
Mayor igual	>=
Menor igual	<=
Distinto	<>

2.5 Concatenación. &

El operador de concatenación es el carácter “&”, se dispone también del método Concat, en las variables de tipo cadena.

2.6 Prioridad de evaluación de las expresiones.

El orden de evaluación de las expresiones es: primero las aritméticas, después las relacionales y finalmente las lógicas.

Los operadores relacionales tienen todos la misma prioridad; es decir, se evalúan de izquierda a derecha, en el orden en que aparecen. Los operadores lógicos y aritméticos se evalúan en el siguiente orden de prioridad

Aritméticos

Exponenciación (^)
Negación (-)
Multiplicación y división (*, /)
División de enteros (\)
Módulo aritmético (Mod)
Adición y substracción (+, -)

Relacionales

Igualdad (=)
Desigualdad (<>)
Menor que (<)
Mayor que (>)
Menor o igual que (<=)
Mayor o igual que (>=)

Operadores lógicos

Not
And, AndAlso
Or, OrElse
Xor

Cuando hay multiplicación y división en la misma expresión, cada operación se evalúa a medida que aparece, de izquierda a derecha. Del mismo modo, cuando se presentan adiciones y substracciones en una misma expresión, cada operación se evalúa tal como aparecen de izquierda a derecha. Es posible usar paréntesis para saltar el orden de preferencia y forzar que algunas partes de una expresión se evalúen antes que otras. Las operaciones entre paréntesis se realizan antes que las de fuera. Sin embargo, dentro de los paréntesis, la precedencia de los operadores se mantiene.

3. Datos.

3.1 Introducción.

Como consecuencia de la integración de todos los lenguajes en esta plataforma de programación y dado que por ejemplo en C, hay un juego de tipos de datos con alguna diferencia a los de V.B., se ha creado un juego de tipos de datos que es posible utilizar desde cualquiera de esos lenguajes de programación dentro de una clase denominada System, en la cual por ejemplo existen punteros y en VB no.

Esta clase System de la que dependen todos los tipos de la plataforma, permitiría por ejemplo desarrollar un lenguaje de programación e implementar desde la misma los tipos de datos que se desearan.

3.2 Tipos de datos

En Visual Basic podemos agrupar los tipos de datos en dos tipos genéricos.

Los que denominamos de Valores, datos información, y los de Referencias, objetos.

Los denominados de Valores agrupan a lo que son los datos.

Ambos tipos arrancan de una raíz común llamada Object.

3.2.1 Valores

Intrínsecos del lenguaje:

```
Byte, Int16, Int32, Integer, Long, Single, Double, Decimal, Boolean, Char, String, Object
```

Estructuras, o lo que conocemos como tipos de usuario.

```
Structure TipoArticulo
  Public Codigo As Int16
  Public Denom As String
  Public Cantidad As Int16
  Private Precio As Single
End Structure
```

Enumerados, parecido a los conjuntos de otros lenguajes.

```
Enum Semana
  Lunes
  Martes
  Miércoles
  Jueves
  Viernes
  Sábado
  Domingo
End Enum
```

Como podemos observar tenemos que definir el ámbito del dato en el tipo de usuario, que puede ser Private, Public, Friend.

Los tipos de datos Intrínsecos, son los nombrados, pero como Visual Basic, pertenece a la plataforma Visual Studio, en esta plataforma existen más tipos de datos, pero que no están definidos en VB.

A pesar de ello VB dispone de recursos para poderlos utilizar.

3.2.2 Utilización.

El uso de los tipos antes expuestos, puede quedar así:

```
Sub Main()
  Dim      Dias As Semana
  Private  Articulo As TipoArticulo
  Public  OtroArticulo As TipoArticulo
  Friend  OtroArticuloMas As TipoArticulo
End Sub
```

El conjunto anterior en una aplicación de consola

```
Module ModuloUno
    Structure TipoArticulo
        Private Codigo As Int16
        Private Denom As System.String
        Private Cantidad As Int16
        Private Precio As Single
    End Structure

    Sub Main()
        Dim Dias As Semana
        Dim Articulo As TipoArticulo
        Dim OtroArticulo As TipoArticulo
        Dim OtroArticuloMas As TipoArticulo
    End Sub
End Module
```

3.2.3 Estructuras, tipos de usuario.

Para las estructuras, un ejemplo puede ser este.

```
Sub Main()

    Dim Articulo As TipoArticulo
    Articulo.Codigo = 123
    Articulo.Cantidad = 10
    Articulo.Denom = "Mesa"
    ` Precio no puede usarse, es Private
    Console.Read()
End Sub
```

Si vemos el ejemplo, podemos observar que no hemos podido asignar valor al precio, y es que este está declarado como Private, y no está disponible, sin embargo los demás son Public y sí pueden ser utilizados. Y todo el conjunto queda como sigue, en una aplicación de Consola.

```
Module ModuloUno
    Structure TipoArticulo
        Public Codigo As Int16
        Public Denom As String
        Public Cantidad As Int16
        Private Precio As Single
    End Structure

    Sub Main()
        Dim Articulo As TipoArticulo

        Articulo.Codigo = 123
        Articulo.Cantidad = 10
        Articulo.Denom = "Mesa"
        Console.Read()
    End Sub
End Module
```

3.2.4 Enumerados.

Para los tipos enumerados, un ejemplo puede ser el que sigue.

```
Module ModuloUno
    Enum Semana
        Lunes
        Martes
        Miercoles
        Jueves
        Viernes
        Sabado
        Domingo
    End Enum

    Sub Main()
        Dim Dias As Semana
        Dias = Semana.Miercoles

        ' Obtenemos el índice del día de la semana
        Console.WriteLine(Dias)
        ' Obtenemos el nombre del día de la semana
        Console.WriteLine(Dias.ToString())
        Console.Read()
    End Sub
End Module
```

Declaramos la variable `Días` del tipo `Semana`.

En la ejecución le asignamos el día de la semana `Miércoles`, `Miércoles` tiene empezando por cero, el índice 2, por lo tanto cuando ejecutamos el ejemplo el valor de la primera línea será 2.

Las variables del tipo `Enum`, disponen del método denominado `.ToString`, que visualiza el elemento que corresponde en el conjunto al valor del índice que tiene la variable, como `Días` vale 2, al ejecutar `Dias.ToString`, se visualiza el valor `Miércoles`.

Pero también puede usarse como `Días = 3`, o lo que es lo mismo, `Días` se convierte en realidad en un índice y según la usemos nos devolverá, en este caso, `Jueves` o el valor 3.

Los valores que se asignen deben estar dentro de los límites del número de elementos del conjunto.

3.3 Declarar las variables con valor inicial.

Las variables pueden inicializarse en el momento de la declaración

```
Dim Suma As Int16 = 10
Dim Estado As Boolean = True
```

También podemos declarar varias variables del mismo tipo en la misma línea

```
Dim Suma, Resta = 5, Multipli As Int16 = 10
Dim Suma, Resta As Int16, Estado As Boolean = True
```

3.4 Ambito de uso de las variables.

Las variables pueden tener los siguientes tipos de ámbito.

```
Private
Public
Protected
Friend
```

Además podemos declarar variables dentro de una instrucción, podemos definir el ámbito de las variables en el siguiente orden.

Modulo, Clase, Formulario

Procedimiento, Función.

Dentro de un procedimiento o una función la declaración será siempre Privada, y es suficiente utilizar DIM.

En un módulo, clase o form, podrá ser Public o Private.

Private el ámbito es solo para ese módulo o form, o clase, Public, podrá ser vista desde otros módulos o formularios.

Crear un proyecto nuevo de tipo Consola.

En el siguiente ejemplo quedará más claro.

```
Module Module1
    Public NivModPub As Int16 = 10
    Private NivModPriv As Int16 = 20
    Friend NivModFri As Int16 = 30

    Sub EjemploUno()
        Dim NivSub As Int16 = 40
        Console.WriteLine("En procedimiento Ejemplo uno")
        Console.WriteLine("{0},{1},{2}", NivModPub, NivModPriv, NivSub)
        Console.WriteLine("{0},{1},{2} ", NivModPub, NivModPriv, NivModFri)
    End Sub

    Sub Main()
        EjemploUno()
        Console.WriteLine("En Main")
        Console.WriteLine("{0},{1},{2} ", NivModPub, NivModPriv, NivModFri)
        Console.Read()
    End Sub
End Module
```

Las variables declaradas en esta clase, a pesar de su tipo no pueden verse en el Módulo Module1.

```
Public Class Ejemplo

    Private NivClasePriv As Int16 = 50
    Public NivClasePub As Int16 = 60

End Class
```

3.5 ¿Por qué Private, Public, Friend en Visual Basic.?

Es un motivo de la nueva estructura a la que se ha llegado en esta nueva versión en la que se intenta llegar a cumplir la definición de una programación dirigida a objetos respetando las ventajas del uso de las clases por un lado, y de la integración que conlleva el uso de la plataforma Studio Net, que en realidad integra en un único código de programación los lenguajes actuales C, Java, y Visual Basic de Microsoft, de forma que el código que generan todos es utilizable por el resto, a base de la utilización de un código intermedio y la interpretación del mismo por una máquina virtual, lo cual permite generar código para Linux por ejemplo, si la máquina se escribe, claro.

Private y Public está claro su utilización, pero Friend es un híbrido que permite que desde el mismo programa pueda ser visto, utilizado por otros módulos, pero oculto para otros desarrollados fuera del mismo.

Todo esto queda justificado cuando se utilizan las clases.

```

Module Ejemplo
  Structure TipoArticulo

    Public Sub New(ByVal Cod As Int16, ByVal Den As String, _
                  ByVal Cant As Int16, ByVal Prec As Int16)

        Codigo = Cod
        Denom = Den
        Cantidad = Cant
        Precio = Prec
    End Sub

    Public Sub Muestra()
        Console.WriteLine("{0},{1},{2},{3}", Codigo, Denom, Cantidad, Precio)
    End Sub

    Private Codigo As Int16
    Private Denom As String
    Private Cantidad As Int16
    Private Precio As Single
  End Structure

  Sub Main()
    Dim Mesa As TipoArticulo
    Dim Silla As TipoArticulo = New TipoArticulo(123, "Silla", 12, 23)

    Silla.Muestra()
    Mesa.Muestra()
    Console.Read()
  End Sub
End Module

```

En el ejemplo vemos como se crea el método New en la estructura TipoArticulo, y los datos se convierten en Private, siendo el método New el que asigna datos a los componentes del tipo.

El método Muestra se encarga de visualizar el contenido de la estructura.

En el Main del ejemplo creamos dos variables de TipoArticulo, una la inicializamos con valores y la otra no.

Después usando el método Muestra de cada una de esas variables, objetos o estructuras podemos ver el contenido de cada una de ellas.

Para que funcione debe asignarse en propiedades de proyecto Sub Main como objeto inicial, y el proyecto es de tipo consola.

3.6 Algo más sobre las variables.

Si al nombre de una variable le colocamos un punto nos aparecerá una lista de métodos disponibles para esa variable.

Esos métodos actúan en función del tipo de las variables.

En el ejemplo que sigue a continuación, podemos observar como A es del tipo String, y para esa variable disponemos de la propiedad, .Length, que para una de tipo numérico no,

En los ejemplos solo hay uno de los usos de cada método, pero la mayoría tiene más de una posible sintaxis, por lo que a la hora de usarlos hay que ver si hay otra versión más adecuada a nuestras necesidades.

```

Module Ejemplo
  Sub Main()
    Dim A As String = 25

    Console.WriteLine(A.ToString)           ' "25"
    Console.WriteLine(Len(A.ToString))     ' 2
    Console.WriteLine(A.Length)            ' 2

  End Sub
End Module

```

Además hasta ahora cuando había que realizar alguna acción sobre una variable, debíamos utilizar una función para actuar sobre ella. Ahora hay que tener presente que una variable en realidad es un objeto más que se corresponde con una clase que le proporciona su operatividad, por lo tanto la misma variable, el mismo objeto en si, dispone de los métodos, antes funciones, que permite que se pueda actuar sobre el, o sobre otras.

En los ejemplos que hay en el capítulo de cadenas, vemos como en muchos casos se ha trabajado con la clase "STRINGS.", que puede ir entre corchetes, para actuar sobre las variables de cadena.

3.6.1 Clone

Se obtiene una copia de la variable clonada en la variable que recibe la asignación.

En el ejemplo vemos como B está declarada pero sin valor inicial, y luego recibe el valor de A, ejecutándose el CompareOrdinal después dando los mismos resultados, el carácter de la posición dos, "c", es mayor que el de la posición uno, "5".

Iguals, cero
Mayor, mayor que cero,
Menor, menor que cero.

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cc"
    Dim B As String
    B = A.Clone
    Console.WriteLine(A.Compare(A, B, False))           ' 0
    Console.WriteLine(B.CompareOrdinal(B.Chars(2), B.Chars(1))) ` 46 Mayor
    Console.WriteLine(A.CompareOrdinal(A.Chars(2), A.Chars(1))) ` 46 Mayor
  End Sub
End Module
```

3.6.2 Compare

Realiza la comparación de dos variables, el valor devuelto es

Iguals, cero
Mayor, mayor que cero,
Menor, menor que cero.

En el ejemplo el valor devuelto es C, porque son iguales, ya que se envía el valor True como argumento, que indica que se ignore las diferencias por mayúsculas y minúsculas

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cc"
    Dim B As String = "25cC"

    Console.WriteLine(" {0},{1}", A.Length, B.Length) ' 5 , 4
    Console.WriteLine(A.Compare(A, B, True))           ' 0
  End Sub
End Module
```

En el siguiente ejemplo devuelve negativo, porque se envía el valor False, lo que provoca el tener presente las diferencias entre mayúsculas y minúsculas.

```
Module Ejemplo
Sub Main()
    Dim A As String = "25cc"
    Dim B As String = "25cC"

    Console.WriteLine(" {0},{1}", A.Length, B.Length) ' 5 , 4
    Console.WriteLine(A.Compare(A, B,False)) ' -1
End Sub
End Module
```

Si cambiamos el tipo de las variables a Byte, el método desaparecería, y no sería utilizable, habría que usar CompareTo.

3.6.3 CompareOrdinal

Este método realiza una comparación del valor del código de Ascii del carácter indicado en las cadenas del argumento.

El valor devuelto por el método sigue el criterio habitual.

iguales, cero
mayor A que B, mayor que cero,
menor A que B, menor que cero

En el ejemplo vemos como CompareOrdinal da que el carácter de la posición dos, "c", es mayor que el de la posición uno, "5".

```
Module Ejemplo
Sub Main()
    Dim A As String = "25cc"
    Console.WriteLine(A.CompareOrdinal(A.Chars(2), A.Chars(1))) ' 46, mayor
    Console.ReadLine()
End Sub
End Module
```

3.6.4 CompareTo

Devuelve un valor menor igual o mayor que cero que indica el resultado de la comparación de A con B. En el ejemplo que hay a continuación el resultado es cero, lo que significa que son iguales.

iguales, cero
mayor A que B, mayor que cero,
menor A que B, menor que cero

```
Module Ejemplo
Sub Main()
    Dim A As Int16 = 25
    Dim B As Int16 = 25

    Console.WriteLine(A.CompareTo(B))
    Console.ReadLine()
End Sub
End Module
```

Sin embargo el ejemplo que sigue no se puede probar por que las variables no son del mismo tipo.

```
Module Ejemplo
  Sub Main()
    Dim A As Int32 = 25
    Dim B As Int16 = 25

    Console.WriteLine(A.CompareTo(B))
    Console.ReadLine()
  End Sub
End Module
```

En el siguiente ejemplo devuelve 23, lo que indica que A es mayor que B.

```
Module Ejemplo
  Sub Main()
    Dim A As Int16 = 25
    Dim B As Int16 = 2

    Console.WriteLine(A.CompareTo(B))
    Console.ReadLine()
  End Sub
End Module
```

Por lo que para el siguiente ejemplo, el resultado que es menor A que B, da -27, o sea menor que cero.

```
Module Ejemplo
  Sub Main()
    Dim A As Int16 = 25
    Dim B As Int16 = 52
    Console.WriteLine(A.CompareTo(B))
    Console.ReadLine()
  End Sub
End Module
```

3.6.5 Concat

Devuelve una cadena de caracteres formada por las variables que se pasan en el argumento, sin embargo esa cadena no se asigna a la variable con la que se ejecuta el método.

En el ejemplo podemos ver los resultados de varias combinaciones.

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cc"
    Dim B As String = "Abcd"

    Console.WriteLine(A.Concat(A))      ' 25cc
    Console.WriteLine(A.Concat(A, B))  ' 25ccAbcd
    Console.WriteLine(A.Concat(B))     ' Abcd
    Console.WriteLine(A.Concat(B, A))  ' Abcd25cc
    Console.WriteLine(A)                ' 25cc
    Console.WriteLine(B)                ' Abcd
    Console.ReadLine()
  End Sub
End Module
```

Podemos observar como después de varias combinaciones del método Concat al visualizar el valor de las variables A y B, estas conservan sus valores originales.

3.6.6 Copy

Este método realiza la copia del valor de la variable B en A, eso se ha hecho siempre como A = B, pero el caso es que los dos sistemas se pueden utilizar y dan el mismo resultado.

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cc"
    Dim B As String = "Abcd"

    Console.WriteLine("Valor de A, {0}", A) ' 25cc
    Console.WriteLine("Valor de B, {0}", B) ' Abcd
    A = [String].Copy(B)
    'A = B
    Console.WriteLine("Valor de A, {0}", A) ' Abcd
    Console.WriteLine("Valor de B, {0}", B) ' Abcd
    Console.ReadLine()
  End Sub
End Module
```

En el ejemplo se han hecho con los dos sistemas y los resultados no cambian.

Posiblemente en una programación avanzada con un uso más exhaustivo de clases sea necesario el uso de este método.

3.6.7 EndsWith, StartsWith

Comprueba si al final o al principio de una cadena existe la cadena que se facilita, devolviendo True o False.

```
Module Ejemplo
  Dim Cadena As String = "Contenido"
  Sub Main()
    Console.WriteLine(Cadena.StartsWith("Conte")) ' True
    Console.WriteLine(Cadena.StartsWith("conte")) ' False

    Console.WriteLine(Cadena.EndsWith("NIDO")) ' False
    Console.WriteLine(Cadena.EndsWith("nido")) ' True
    Console.ReadLine()
  End Sub
End Module
```

3.6.8 Equals

Se dispone de otro método que lo que hace es indicar si son o no iguales, pero han de serlo incluso en el tipo, pues el ejemplo que sigue da como resultado falso

```
Module Ejemplo
  Sub Main()
    Dim A As Int32 = 25
    Dim B As Int16 = 25

    Console.WriteLine(A.Equals(B))
  End Sub
End Module
```

Y el motivo es porque siendo iguales en valor, no lo son en el tipo.

El siguiente ejemplo si que da True, pues son iguales en valor y en tipo.

```
Module Ejemplo
  Sub Main()
    Dim A As Int16 = 25
    Dim B As Int16 = 25
    Console.WriteLine(A.Equals(B))
    Console.ReadLine()
  End Sub
End Module
```

3.6.9 GetType

Este método devuelve el tipo de la variable, indicando que pertenece a System.

```
Module Ejemplo
  Sub Main()
    Dim A As Int32 = 25
    Dim B As Int16 = 25
    Console.WriteLine(A.GetType)      \ System.Int32
    Console.WriteLine(B.GetType)     \ System.Int16
    Console.ReadLine()
  End Sub
End Module
```

3.6.10 GetTypeCode

En lugar de devolver el literal con el nombre del tipo, nos devuelve un código, lo que facilita su utilización.

```
Module Ejemplo
  Sub Main()
    Dim A As Int32 = 25
    Dim B As Long = 25
    Console.WriteLine(A.GetType)      \ System.Int21
    Console.WriteLine(A.GetTypeCode) \ 9
    Console.WriteLine(B.GetType)     \ System.Long
    Console.WriteLine(B.GetTypeCode) \ 11
    Console.ReadLine()
  End Sub
End Module
```

3.6.11 IndexOf, LastIndexOf

Busca el contenido de una cadena comenzando desde el principio o desde el final, devuelve la posición donde empieza esa cadena, la primera posición es cero.

El valor negativo significa que no se encuentra la cadena

```
Module Ejemplo
  Dim Cadena As String = "Contenido"

  Sub Main()
    Console.WriteLine(Cadena.IndexOf("Conte")) \ 0
    Console.WriteLine(Cadena.IndexOf("conte")) \ -1
    Console.WriteLine(Cadena.LastIndexOf("NIDO")) \ -1
    Console.WriteLine(Cadena.LastIndexOf("nido")) \ 5
    Console.ReadLine()
  End Sub
End Module
```

3.6.12 IndexOfAny, LastIndexOfAny.

Busca el contenido de una cadena comenzando desde el principio o desde el final, devuelve la posición donde empieza esa cadena, la primera posición es cero, pero localizando cualquier carácter de la cadena.

El valor negativo, significa que no se encuentra la cadena

Comparar los resultados de éste método con el anterior, para IndexOfAny, son similares, igual si se atiende al cambio, y para LastIndexOfAny, devuelve ocho, que es la primera coincidencia que se da.

```
Module Ejemplo
  Dim Cadena As String = "Contenido"

  Sub Main()
    Console.WriteLine(Cadena.IndexOfAny("onte"))      \ 1
    Console.WriteLine(Cadena.IndexOfAny("onte"))      \ 1
    Console.WriteLine(Cadena.LastIndexOfAny("NIDO"))  \ -1
    Console.WriteLine(Cadena.LastIndexOfAny("nido"))  \ 8
    Console.ReadLine()
  End Sub
End Module
```

3.6.13 Insert

Inserta una cadena en otra a partir de la posición que se indique.

```
Module Ejemplo
  Dim Cadena As String = "sufridores"

  Sub Main()
    Cadena = Cadena.Insert(0, "Hola informáticos ")
    Console.WriteLine(Cadena)
  End Sub
End Module
```

3.6.14 PadLeft, PadRight

Estos métodos rellenan con el carácter especificado por la derecha o por la izquierda hasta la longitud indicada en el argumento.

```
Module Ejemplo
  Sub Main()
    Dim X As Int16
    Dim A As Char = "."
    Dim Valor As String = "Hola mundo"
    Console.WriteLine("Visualizamos el contenido ")
    Valor = Valor.PadRight(20, A)
    Console.WriteLine(Valor.Length)
    Console.WriteLine("[{0}]", Valor)

    Valor = "Hola mundo"
    Valor = Valor.PadLeft(20, A)
    Console.WriteLine(Valor.Length)
    Console.WriteLine("[{0}]", Valor)
    Console.ReadLine()
  End Sub
End Module
```

3.6.15 Remove

Elimina un número de caracteres de una cadena a partir de una posición indicada.

```
Module Ejemplo
  Dim Cadena As String = "sufridores"
  Sub Main()
    Cadena = Cadena.Insert(0, "Hola informáticos ")
    Console.WriteLine(Cadena)
    Cadena = Cadena.Remove(18, 10)
    Console.WriteLine(Cadena)
    Console.ReadLine()
  End Sub
End Module
```

3.6.16 Replace

Busca en una cadena el contenido facilitado para sustituirlo por la misma, o eliminarlo si el argumento facilitado es un nulo como el ejemplo.

Si la cadena no existe no se hace ningún cambio en la cadena origen.

```
Module Ejemplo
  Dim Cadena As String = "sufridores"
  Sub Main()
    Cadena = Cadena.Insert(0, "Hola informáticos ")
    Console.WriteLine(Cadena)
    Cadena = Cadena.Replace("sufridores", "")
    Console.WriteLine(Cadena)
    Console.ReadLine()
  End Sub
End Module
```

3.6.17 SubString

Extrae una cadena de la cadena origen a partir de la posición indicada y de la longitud que se indica en el argumento.

```
Module Ejemplo
  Dim Cadena As String = "sufridores"
  Sub Main()
    Cadena = Cadena.Insert(0, "Hola informáticos ")
    Console.WriteLine(Cadena)
    Cadena = Cadena.Substring(0, 18)
    Console.WriteLine(Cadena) \ "Hola informáticos"
    Console.ReadLine()
  End Sub
End Module
```

3.6.18 ToCharArray

Copia los caracteres de la instancia en una matriz de caracteres Unicode.

3.6.19 ToString

Convierte el valor de la variable numérica en un valor de cadena de caracteres.

En el ejemplo podemos comprobar que la longitud, Len(), de ToString es de dos caracteres, para el valor de la variable A.

```
Module Ejemplo
Sub Main()
    Dim A As Byte = 25
    Console.WriteLine(A.ToString)           ' "25"
    Console.WriteLine(Len(A.ToString))     ' 2
    Console.ReadLine()
End Sub
End Module
```

3.6.20 ToLower, ToUpper

Convierte a minúscula o mayúsculas las letras de la cadena de caracteres.

```
Module Ejemplo
Sub Main()
    Dim A As String = "25cC "
    Console.WriteLine(A.ToString)           ' "25cC "
    Console.WriteLine(Len(A.ToString))     ' 5
    Console.WriteLine(A.Length)           ' 5

    Console.WriteLine(A.ToLower)           ' 25cc
    Console.WriteLine(A.ToUpper)          ' 25CC
    Console.ReadLine()
End Sub
End Module
```

3.6.21 Trim, TrimEnd, TrimStart

Elimina los blancos, o el carácter indicado en el argumento, por ambos lados de la variable de tipo String, o por el final o el principio.

En el ejemplo podemos ver el uso de la función Trim, y del Método.

```
Module Ejemplo
Sub Main()
    Dim A As String = "25cc"

    Console.WriteLine(A.ToString)           ' "25cc"
    Console.WriteLine(Len(A.ToString))     ' 4
    Console.WriteLine(A.Length)           ' 4
    Console.WriteLine(Trim(A))             ' "25cc"

    Console.WriteLine("[{0}]", A.TrimEnd("c")) ' "[25]"
    Console.WriteLine("[{0}]", A.TrimStart("2")) ' "[5cc]"
    Console.WriteLine("[{0}]", Trim(A))     ' "[25cc]"
    Console.ReadLine()
End Sub
End Module
```

3.7 Y además hay propiedades.

3.7.1 Chars

Devuelve el carácter de la cadena indicado en el argumento.

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cC"
    Console.WriteLine(A.Chars(0)) ' 2
    Console.WriteLine(A.Chars(1)) ' 5
    Console.WriteLine(A.Chars(2)) ' c
    Console.WriteLine(A.Chars(3)) ' C
    Console.ReadLine()
  End Sub
End Module
```

3.7.2 Empty

Devuelve el contenido de la variable a nulo, longitud 0.

En el ejemplo podemos ver como pasa de longitud 5 a 0.

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cC "
    Console.WriteLine(A.Empty) ' ""
    Console.WriteLine(Len(A.Empty)) ' 0
    Console.ReadLine()
  End Sub
End Module
```

3.7.3 Length

Devuelve el número de caracteres de la cadena.

En el ejemplo podemos ver su uso, y el de la función Len()

```
Module Ejemplo
  Sub Main()
    Dim A As String = "25cc"
    Console.WriteLine(Len(A.ToString)) ' 4
    Console.WriteLine(A.Length) ' 4
    Console.ReadLine()
  End Sub
End Module
```

3.7.4 MaxValue, MinValue

MaxValue y MinValue que nos devuelven los rangos en los que pueden tomar valores cada tipo de variable.

En el ejemplo vemos el valor para una variable del tipo Byte.

```
Module Ejemplo
  Sub Main()
    Dim A As Byte = 25
    Console.WriteLine(A.MinValue) ' 0
    Console.WriteLine(A.MaxValue) ' 255
    Console.ReadLine()
  End Sub
End Module
```