

1.2.3 Declaración dinámica.

Para cuando la dimensión del array depende de la ejecución del programa.

```
Module Ejemplo
  Sub Main()

    Dim Valor As String
    ` Declaración e inicialización
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                           "Abril", "Mayo", "Junio", _
                           "Julio", "Agosto", "Septiembre", _
                           "Octubre", "Noviembre", "Diciembre"}

    ` Declaración dinámica
    Dim OtrosMeses() As String

    ` En esta línea nos dará Nothing, no hay asignación todavía
    Console.WriteLine("Tipo del vector {0} ", TypeName(OtrosMeses))

    ` Asignación de dimensión
    ReDim OtrosMeses(Meses.GetUpperBound(0))

    ` Aquí el resultado será String
    Console.WriteLine("Tipo del vector {0} ", TypeName(OtrosMeses))

    Console.WriteLine("Visualizamos el contenido ")

    ` Copia de meses sobre otros meses
    Meses.CopyTo(OtrosMeses, 0)

    ` Visualización del vector
    For Each Valor In OtrosMeses
      Console.WriteLine(Valor)
    Next

    Console.WriteLine("Elementos del vector {0} ", OtrosMeses.GetLength(0))
    Console.WriteLine("Primer índice {0} ", OtrosMeses.GetLowerBound(0))
    Console.WriteLine("Ultimo índice {0} ", OtrosMeses.GetUpperBound(0))
    Console.WriteLine("Visualizamos el contenido ")
    Console.ReadLine()
  End Sub
End Module
```

1.3 Recorrido del vector.

Para recorrer el vector podemos usar cualquiera de los dos sistemas, el de recorrerlo como una colección.

```
Module Ejemplo
  Sub Main()
    ` Declaración e inicialización del vector con los
    ` nombres de los meses, de tipo string
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                           "Abril", "Mayo", "Junio", _
                           "Julio", "Agosto", "Septiembre", _
                           "Octubre", "Noviembre", "Diciembre"}

    ` Declaramos la variable del mismo tipo para poder
    ` recorrer el vector
    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido")
  End Sub
End Module
```

```

    ' Visualizamos el vector
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next
    Console.ReadLine()
End Sub
End Module

```

Pero también se puede seguir haciendo lo clásico, con la función Ubound, aunque ahora también hay un método del objeto Meses, que se llama GetUpperBound.

Y también podemos utilizar los siguientes métodos, en lugar del X=0 to 11, o to Ubound(meses)

```

For X=Meses.GetLowerBound(0) To Meses.GetUpperBound(0)

```

```

Module Ejemplo
Sub Main()
    ' Declaración e inicialización del vector con los
    ' nombres de los meses, de tipo string
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    Dim X As Int16
    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido")

    ' Mostramos lo valores
    For X = 0 To UBound(Meses)
        Console.WriteLine(Meses(X).ToString)
    Next

    ' Mostramos lo valores
    For X = Meses.GetLowerBound(0) To Meses.GetUpperBound(0)
        Console.WriteLine(Meses(X).ToString)
    Next
    Console.ReadLine()
End Sub
End Module

```

1.4 Clase Array

Esta clase dispone de los siguientes métodos

BinarySearch	Clear
Copy	CreateInstance
IndexOf	LastIndexOf
Reverse	Sort

Cada uno de estos métodos a su vez están sobrecargados, OverLoad, de tal forma que se pueden utilizar con varios tipos distintos de argumentos en la utilización.

1.5 Clase NuestroArray

Además a esto hemos de añadir que el Vector definido por nosotros, como tal dispone de los siguientes métodos:

BinarySearch	Clear
Clone	Copy
CopyTo	CreateInstance
GetEnumerator	GetLength
GetLongLength	GetLowerBound
GetType	GetUpperBound
GetValue	IndexOf
Initialize	LastIndexOf
Rank	Reverse
SetValue	Sort

Y el Vector, o Array, con el que estamos trabajando dispone también de las siguientes propiedades:

IsFixedSize	IsReadOnly
IsSynchronized	Length
LongLength	Rank
SyncRoot	

1.6 Métodos de arrays.

Vamos a ver la utilidad de cada uno de los métodos existentes en cada una de las dos clases.

1.6.1 Clear

Siguiendo con el ejemplo anterior, el método Clear realiza el vaciado del contenido del vector, pero no del número de elementos, que seguirá siendo el mismo

```
Module Ejemplo
Sub Main()

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido antes ")

    ' Visualizamos el vector
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next
    Console.WriteLine("Elementos del vector {0} ", Meses.Length)

    ' Vaciado del vector
    Meses.Clear(Meses, 0, Meses.Length)

    Console.WriteLine("Visualizamos el contenido después ")
End Sub
```

```

Console.WriteLine("Elementos del vector {0} ", Meses.length)

' Visualizamos el vector
For Each Valor In Meses
    Console.WriteLine(" [{0}]", Valor)
Next
Console.ReadLine()
End Sub
End Module

```

1.6.2 Clone

Este método lo que hace es generar una copia de un array ya existente.

```

Module Ejemplo
Sub Main()

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    Dim OtrosMeses() As String
    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido ")
    Console.WriteLine("Elementos del vector {0} ", Meses.GetUpperBound(0))
    ' Visualizamos el vector
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next
    ' Clonamos el vector
    OtrosMeses = Meses.Clone()
    Console.WriteLine("Elementos vector {0} ", OtrosMeses.GetUpperBound(0))
    Console.WriteLine("Visualizamos el contenido ")

    ' Visualizamos el vector
    For Each Valor In OtrosMeses
        Console.WriteLine(Valor)
    Next
    Console.ReadLine()
End Sub
End Module

```

1.6.3 ConstrainedCopy

Copia el número de elementos indicado de un array de origen a un array de destino, en la dimensión que se indica, y del número de elementos que se indique.

```

Module Ejemplo
Sub Main()
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    Dim Mesesitos(3) As String
    Dim Dato As String
    Dim DimOrigen As Integer = 0
    Dim DimDestino As Integer = 0
    Dim Cuantos As Integer = 2 ' probar con 2 y 4

    ' Contenido vector origen

```

```

For Each Dato In Meses
    Console.WriteLine(Dato)
Next

' Copia contenido de un vector en otro
Array.ConstrainedCopy(Meses, DimOrigen, Mesesitos, DimDestino, Cuantos)

' Contenido vector destino
Console.WriteLine("Visualizar el vector de destino")
For Each Dato In Mesesitos
    Console.WriteLine("Mes de {0} ", Dato)
Next
Console.ReadLine()
End Sub
End Module

```

1.6.4 Copy

A diferencia de Clone, Copy hace un duplicado del vector, solo del número de elementos que se indica en el argumento.

```

' Copia solo el número de elementos indicado,
' OtrosMeses.GetUpperBound(0) vale 5, que es el número de elementos
' que tiene
' Meses es el origen,
' OtrosMeses es el destino
Meses.Copy(Meses, OtrosMeses, OtrosMeses.GetUpperBound(0))
Module Ejemplo
Sub Main()
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    ' Le damos cinco elementos
    Dim OtrosMeses(5) As String
    Dim Valor As String
    Console.WriteLine("Visualizamos el contenido ")
    Console.WriteLine("Elementos del vector {0} ", Meses.GetUpperBound(0))
    ' Visualizamos el vector
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next

    ' Copia solo el número de elementos indicado,
    ' OtrosMeses.GetUpperBound(0) vale 5, que es el número de elementos
    ' que tiene
    ' Meses es el origen, OtrosMeses es el destino
    Meses.Copy(Meses, OtrosMeses, OtrosMeses.GetUpperBound(0))

    Console.WriteLine("Elementos vector {0} ", OtrosMeses.GetUpperBound(0))

    Console.WriteLine("Visualizamos el contenido ")
    ' Visualizamos el vector
    For Each Valor In OtrosMeses
        Console.WriteLine(Valor)
    Next
    Console.ReadLine()
End Sub
End Module

```

1.6.5 CopyTo

Realiza la copia del Vector de origen sobre el de destino, colocando los datos a partir de la posición que se indica, se ha de controlar que hay espacio en el vector de destino.

```
' Se copia el vector Meses completo y se coloca
' a partir de la sexta posición en el vector de
' destino OtrosMeses
Meses.CopyTo(OtrosMeses, 6)
Module Ejemplo
Sub Main()

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    ' Le damos 20 elementos
    Dim OtrosMeses(20) As String
    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido ")
    Console.WriteLine("Elementos del vector {0} ", Meses.GetUpperBound(0))
    ' Visualizamos el vector
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next
    ' Se copia el vector Meses completo y se coloca
    ' a partir de la sexta posición en el vector de
    ' destino OtrosMeses
    Meses.CopyTo(OtrosMeses, 6)

    Console.WriteLine("Elementos          del          vector          {0}          ",
OtrosMeses.GetUpperBound(0))
    Console.WriteLine("Visualizamos el contenido ")
    ' Visualizamos el vector
    For Each Valor In OtrosMeses
        Console.WriteLine("[ {0} ]", Valor)
    Next
    Console.ReadLine()
End Sub
End Module
```

1.6.6 CreateInstance

Crema un array a partir de la información existente en el array de origen.

Observar que en este ejemplo está activado Option Strict a Off, sin comentarios.

```
Option Strict Off
Module Ejemplo
Sub Main()
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    Dim Dato As String
    ' Contenido vector origen
    For Each Dato In Meses
        Console.WriteLine(Dato)
    Next
    ' Crear el vector con createinstance
    ' Dim Otro = Array.CreateInstance(Meses.GetType, Meses.Length)
```

```

` El formato anterior, aunque parece lógico no sirve.
Dim Otro = Array.CreateInstance(GetType(String), Meses.Length)
` Cargarlo con datos
Meses.CopyTo(Otro, 0)
` Contenido vector destino
Console.WriteLine("Visualizar el vector de destino")
For Each Dato In Otro
    Console.WriteLine("Mes de {0} ", Dato)
Next
Console.ReadLine()
End Sub
End Module

```

1.6.7 GetLength, GetLowerBound, GetUpperBound

Estos métodos devuelven el número de elementos del vector, el primer índice y el último.

```

Module Ejemplo
Sub Main()
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}
    Console.WriteLine("Visualizamos el contenido ")
    Console.WriteLine("Elementos del vector {0} ", Meses.GetLength(0))

    ` Visualizamos el vector
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next
    Console.WriteLine("Elementos del vector {0} ", Meses.GetLength(0)) ` 12
    Console.WriteLine("Primer índice {0} ", Meses.GetLowerBound(0)) ` 0
    Console.WriteLine("Ultimo índice {0} ", Meses.GetUpperBound(0)) ` 11
    Console.ReadLine()
End Sub
End Module

```

1.6.8 GetType

Devuelve el tipo que tiene el array dentro de la clase System

```

Module Ejemplo
Sub Main()
    Dim X As Int16

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                            "Abril", "Mayo", "Junio", _
                            "Julio", "Agosto", "Septiembre", _
                            "Octubre", "Noviembre", "Diciembre"}

    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido ")
    Console.WriteLine(Meses.GetType)
    For Each Valor In Meses
        Console.WriteLine(Valor)
    Next
    Console.ReadLine()
End Sub
End Module

```

1.6.9 GetValue

Devuelve el contenido de la posición que indicamos en el argumento.
En el ejemplo el valor devuelto es Abril.
Pero también es válido y da también Abril.

```
Console.WriteLine(Meses(3))
Module Ejemplo
  Sub Main()
    Dim X As Int16

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                             "Abril", "Mayo", "Junio", _
                             "Julio", "Agosto", "Septiembre", _
                             "Octubre", "Noviembre", "Diciembre"}

    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido ")
    Console.WriteLine(Meses.GetValue(3))

    For Each Valor In Meses
      Console.WriteLine(Valor)
    Next
    Console.ReadLine()
  End Sub
End Module
```

1.6.10 Búsqueda, IndexOf, LastIndexOf

Devuelve la posición de la primera, IndexOf, o de la última, LastIndexOf, vez que aparece el elemento indicado en el argumento.

En el ejemplo el resultado es 1 y 7.

```
Module Ejemplo
  Sub Main()
    Dim X As Int16

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                             "Abril", "Mayo", "Junio", _
                             "Julio", "Febrero", "Septiembre", _
                             "Octubre", "Noviembre", "Diciembre"}

    Dim Valor As String

    Console.WriteLine(Meses.IndexOf(Meses, "Febrero"))      ` 1
    Console.WriteLine(Meses.LastIndexOf(Meses, "Febrero")) ` 7

    Console.WriteLine("Visualizamos el contenido ")
    For Each Valor In Meses
      Console.WriteLine(Valor)
    Next
    Console.ReadLine()
  End Sub
End Module
```

1.6.11 Rank

Devuelve el número de dimensiones de un array.

```
Console.WriteLine("Meses tiene {0} dimensiones ", Meses.Rank)
```

1.6.12 Reverse

Invierte el contenido actual del vector.

```
Module Ejemplo
  Sub Main()
    Dim X As Int16

    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                             "Abril", "Mayo", "Junio", _
                             "Julio", "Agosto", "Septiembre", _
                             "Octubre", "Noviembre", "Diciembre"}

    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido ")

    Meses.Reverse(Meses)
    [Array].Reverse(Meses) ` se queda como estaba

    For Each Valor In Meses
      Console.WriteLine(Valor)
    Next
    Console.ReadLine()
  End Sub
End Module
```

1.6.13 SetValue

Establece un valor nuevo para el elemento que se indica, en el ejemplo sustituimos Noviembre por Juan.

```
Module Ejemplo
  Sub Main()
    Dim X As Int16
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                             "Abril", "Mayo", "Junio", _
                             "Julio", "Agosto", "Septiembre", _
                             "Octubre", "Noviembre", "Diciembre"}

    Dim Valor As String

    Console.WriteLine("Visualizamos el contenido ")
    Meses.SetValue("Juan", 10)
    For Each Valor In Meses
      Console.WriteLine(Valor)
    Next
    Console.ReadLine()
  End Sub
End Module
```

1.6.14 Sort

Este método lo que hace es clasificar el vector, que evidentemente siempre se agradece. Hay dos sistemas en el ejemplo,

```
[Array].Sort(Meses)
```

En este se usa la clase genérica Array para usar sus métodos.

```
Meses.Sort(Meses)
```

Aquí se usa el vector, con el método sort.

Como se puede comprobar cualquiera de los dos es válido.

```
Module Ejemplo
Sub Main()
  Dim X As Int16

  Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                           "Abril", "Mayo", "Junio", _
                           "Julio", "Agosto", "Septiembre", _
                           "Octubre", "Noviembre", "Diciembre"}

  Dim Valor As String

  Console.WriteLine("Visualizamos el contenido ")

  Meses.Sort(Meses)
  [Array].Sort(Meses)

  For Each Valor In Meses
    Console.WriteLine(Valor)
  Next
  Console.ReadLine()
End Sub
End Module
```

2. Funciones del lenguaje.

2.1 Introducción.

Bueno, ya sabemos lo que son las funciones del lenguaje, pero dado que hay funciones nuevas y curiosas, vamos a ver al menos esas.

Entrar en cuales son las más importantes es del genero inútil, pues lo de importante es muy personal.

Después de cotillear por la red, hemos observado que se sigue en varios entornos el criterio de tener activado el Option Strict en On.

Eso provoca un régimen muy exhaustivo de control del intérprete, lo cual ahorra futuros problemas, pero te obliga a la conversión continua de los tipos de datos a los correctos, y acabas aprendiendo de memoria los nombres de las funciones de conversión, sobre todo las numéricas, CInt, CSng, CDbI, CLng ...

Por otro lado la mayoría de funciones necesarias, aparecen en sus temas correspondientes, cadenas, archivos. ...

2.2 CType.

La misión de ésta función es el de la conversión del tipo de objeto recibido al tipo de objeto indicado, evidentemente siempre que eso sea posible.

Es muy útil, por lo menos de momento la hemos utilizado mucho, uno de sus usos está en el uso de la clase ElementoLista, para convertir el contenido de un ListBox, al tipo de la mencionada clase.

```
Private Sub Lista01_Click(ByVal sender As Object, _
                        ByVal e As System.EventArgs) _
    Handles Lista01.Click
    Dim Grupo As ElementoLista = CType(Lista01.SelectedItem, ElementoLista)
    CargaListaAlumnos(Lista02, Conexion, Grupo.Codigo)
    CargaAsignaturas(Lista03, Conexion, Grupo.Codig2, Grupo.Codig3)
End Sub
```

El ejemplo es, otro de los mejores destinos de ésta función, o al menos de momento.

Le damos el objeto que recibimos en el evento Click, u otro claro está, y el nos devuelve el nombre del objeto, u otra propiedad, con lo que podemos direccionar, o saber cual es el control que ha recibido el Click del ratón.

No es el Index, pero sirve..

```
Private Sub Label1_Click(ByVal sender As Object, _
                        ByVal e As System.EventArgs) _
    Handles Label1.Click, _
    Label2.Click
    MessageBox.Show(CType(sender, Label).Text)
    Select Case CType(sender, Label).Text
        Case "Label1"
            MessageBox.Show("Se ha pulsado el label uno", "Prueba del nueve",
                MessageBoxButtons.OK, MessageBoxIcon.Information)
        Case "Label2"
            MessageBox.Show("Se ha pulsado el label dos", "Prueba del nueve",
                MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Select
End Sub
```

O podemos hacer

```
Private Sub Label1_Click(ByVal sender As Object, _
                        ByVal e As System.EventArgs) _
    Handles Label1.Click, _
           Label2.Click

    Dim Indice As Integer
    Indice = CInt(Strings.Right(CType(sender, Label).Name, 1))

    Select Case Indice
        Case 1
            MessageBox.Show("Se ha pulsado el label uno", "Prueba del nueve",
                MessageBoxButtons.OK, MessageBoxIcon.Information)
        Case 2
            MessageBox.Show("Se ha pulsado el label dos", "Prueba del nueve",
                MessageBoxButtons.OK, MessageBoxIcon.Information)
    End Select
End Sub
```

Que aún creemos que es mucho más interesante, ya que índice se convierte en el Index de la versión 6, y podemos usarlo en cualquier tipo de llamada en procedimientos.

2.3 Iif.

Esta función toma tres argumentos, el primero es una expresión cuyo resultado será True o False, y los dos siguientes el objeto a devolver, según el resultado de la expresión.

```
Module Ejemplo
    Dim Nota As Integer = 1

    Sub Main()
        Console.WriteLine("La nota {0} es: {1} ", Nota, CStr(Iif((Nota > 10
Or Nota < 0), "Incorrecta", "Correcta")))
        Console.ReadLine()
    End Sub
End Module
```

En el ejemplo se toma como primer parámetro un valor numérico al cual se le comprueba su rango. Si el rango no es correcto se devuelve el texto "Incorrecto" y si lo es se devuelve el texto "Correcto".

No hay porque devolver un String, se puede devolver cualquier tipo de objeto.

2.4 DateAdd, DateDiff

Estas funciones, más que importantes, igual que Format, son cómodas, por eso las hemos puesto, un ejemplo es mejor que mil palabras.

```
Module Ejemplo
  Sub Main()
    Dim Fecha As Date
    Dim Nueva As Date
    Dim Cuantos As Long
    Dim Diferencia As Long
    Dim Intervalo As DateInterval

    Fecha = Now
    Cuantos = 12
    Intervalo = DateInterval.Month
    Nueva = DateAdd(Intervalo, Cuantos, Fecha)
    Diferencia = DateDiff(Intervalo, Fecha, Nueva)
    Console.WriteLine("Fecha actual {0} nueva fecha {1} ", Fecha, Nueva)
    Console.WriteLine("Diferencia {0} {1} ", Diferencia, Intervalo)

    Intervalo = DateInterval.Weekday
    Nueva = DateAdd(Intervalo, Cuantos, Fecha)
    Diferencia = DateDiff(Intervalo, Fecha, Nueva)
    Console.WriteLine("Fecha actual {0} nueva fecha {1} ", Fecha, Nueva)
    Console.WriteLine("Diferencia {0} {1} ", Diferencia, Intervalo)

    Intervalo = DateInterval.Year
    Nueva = DateAdd(Intervalo, Cuantos, Fecha)
    Diferencia = DateDiff(Intervalo, Fecha, Nueva)
    Console.WriteLine("Fecha actual {0} nueva fecha {1} ", Fecha, Nueva)
    Console.WriteLine("Diferencia {0} {1} ", Diferencia, Intervalo)

    Intervalo = DateInterval.Hour
    Nueva = DateAdd(Intervalo, Cuantos, Fecha)
    Diferencia = DateDiff(Intervalo, Fecha, Nueva)
    Console.WriteLine("Fecha actual {0} nueva fecha {1} ", Fecha, Nueva)
    Console.WriteLine("Diferencia {0} {1} ", Diferencia, Intervalo)
    Console.ReadLine()
  End Sub
```


3. Estructuras de Control.

3.1 Introducción.

Las instrucciones de control son las mismas, pero se ven reforzadas por algunas mejoras.
Las instrucciones básicas son:

```
If Then
  Else
End if

Select Case
  Case
  Case else
End Select
```

Donde hay cambios es en las expresiones, que han aparecido AndAlso y OrElse, lo que agiliza la evaluación de las mismas, ya que no siguen evaluando cuando ya de entrada se cumple una parte de la expresión OrElse, o no se cumple en el caso del AndAlso,

3.2 Instrucción If

El If puede estar anidado y acabar con un End If, con lo que queda muy claro cual es su zona de influencia.

También se dispone del Elseif, pero nada de esto es nuevo pues ya lo conocíamos.

```
If Condición Then ....

If Condición Then ..... Else .....

If Condición Then
  ....
else
  ....
End If
```

```
If Condición Then
  If Condición Then
    ....
  else
    ....
  End If
Else
  ....
End If
```

En fin nada que no sepamos.

3.3 Select Case

A la hora de crear estructuras, la instrucción Select Case es mucho más versátil.

```
Select Case Condición
  Case true
  Case else
End Select

Select Case Condición
  Case 1
  Case 2,3,5, 10 To 20
  Case else
End Select
```

Por otro lado el uso de enumeraciones facilita o ayuda mucho en la creación del código. El ejemplo se corresponde con el evento Keypress de un TextBox.

```
Select Case Char.IsDigit(e.KeyChar) Or _
    Char.IsNumber(e.KeyChar) Or _
    Char.IsPunctuation(e.KeyChar)
Case True
    ' válido
Case Else
    Select Case e.KeyChar
    Case ControlChars.Back
    Case Else
        ' no vale
        e.Handled = True
    End Select
End Select
```

4. Bucles

4.1 Introducción.

Los que ya conocíamos,

```
Do
Loop

While
End while

For
Next
```

El Bucle Do Loop admite las cuatro posibilidades de Bucle, Repeat y While, While es una herencia quizás de las versiones anteriores de Basic, pues es redundante con Do While.

4.2 Do Loop

El bucle Do Loop tiene las siguientes posibilidades

```
Do
...
Loop Until Condicion

Do Until Condicion
...
Loop

Do While Condicion
...
Loop

Do
...
Loop While Condicion

While Condicion
...
End while
```

4.3 For Next

El bucle For es como el de toda la vida, pero incorpora la posibilidad de recorrer colecciones, que ya había en Vb6.

```
For Each Valor In Meses
    Console.WriteLine(Valor)
Next

For X=0 To 20
    Console.WriteLine(X)
Next
```

Nada nuevo.

5. Procedimientos.

5.1 Introducción.

Pueden ser Públicos o Privados, Protegido, Protected, o Amigo, Friend.

Pueden aparecer en una estructura, en una clase, o en un formulario.

Las variables declaradas en su interior son propias al procedimiento, locales.

La recepción de las variables exteriores se hace con las reglas ya conocidas, por defecto es por valor, no por referencia como antes, y delante de la variable aparecerá siempre, ByVal por valor, o ByRef por referencia.

Igual que en Vb6, podemos declarar variables opcionales, Optional, pero han de estar inicializadas a un valor por defecto.

La salida será en la última línea, End Sub, o cuando encuentra la instrucción Exit Sub, cuando hay tratamiento de errores.

También pueden usarse de forma recursiva.

5.2 Ejecución normal.

```
Module Ejemplo
  Private Sub ProcedureSinDatos()
    Console.WriteLine("Sin datos")
  End Sub

  Private Sub ProcedureConDatos(ByVal Texto As String)
    Console.WriteLine(Texto)
    ' como es ByVal no cambia
    Texto = Texto & "Cambiado"
  End Sub

  Private Sub ProcedureConDatosPorRef(ByRef Texto As String)
    Console.WriteLine(Texto)
    ' como es ByRef cambia
    Texto = Texto & " cambiados"
  End Sub

  Private Sub ProcedureConDatosOpcional(ByVal Texto As String, _
    Optional ByVal TextoDos As String = "")
    Console.WriteLine(Texto)
    ' como es ByVal no cambia
    Texto = Texto & " cambiados"
    Console.WriteLine(TextoDos)
  End Sub

  Sub Main()
    Dim Cadena As String = "Con datos"

    ProcedureSinDatos()

    ProcedureConDatos(Cadena)
    Console.WriteLine(Cadena)

    ProcedureConDatosPorRef(Cadena)
    Console.WriteLine(Cadena)

    ProcedureConDatosOpcional(Cadena)
    ProcedureConDatosOpcional(Cadena, Cadena)

  End Sub
End Module
```

5.3 Tratamiento de errores.

En la versión actual de VB, hay dos sistemas de captura de errores, uno lo llaman estructurado y el otro no..

El no estructurado es el clásico, debe ser por lo de las etiquetas y el Goto,

```
On Error GoTo Errores
```

```
Private Sub Procedure()  
On Error Goto Errores  
  
` ejecución normal  
  
Salida:  
  
` ejecución para la salida, con o sin errores  
  
Exit Sub  
  
Errores:  
` Tratamiento de los errores  
Select case Err  
Case 3021  
Resume Salida  
Case else  
Resume salida  
End select  
End Sub
```

El estructurado se basa en el uso de una instrucción denominada Try con los siguientes elementos.

```
... / ... Viene el procedimiento.  
  
` Código sin control de errores.  
  
Try  
` Ejecución normal  
Catch Ex as Exception  
` Ejecución para la captura de los errores  
Finally  
` Siempre se ejecuta, haya o no errores, como final de la instrucción  
End Try  
  
` Código sin control de errores.  
  
... / ... Sigue el procedimiento
```

El ejemplo que hay a continuación es un ejemplo sencillo sin captura del código que genera el error, solamente captura la situación de anormalidad, pero sin poder especificar cual.

Podemos observar que la parte inicial del procedimiento no esta dentro de la captura de errores.

Y que la parte del Finally se ha preparado para que al ejecutarse, el texto sea siempre congruente.

Por otro lado a la hora de declarar el tipo de la excepción en la parte del Catch, hay que declarar la excepción del tipo adecuado a la excepción que se va a producir, sino el tratamiento se queda cojo.

La lista desplegable que se activa a la hora de seleccionar el tipo nos facilitará recordar o elegir cual hemos de usar.

```

Private Sub Lee()
    Dim Registro As RegRandom
    Dim Canal As Int16 = FreeFile()
    Dim Nombre As String = "No existe.Txt"
    Dim Posicion As Int16
    Dim Stream As System.IO.StreamReader

    Console.WriteLine()
    Console.WriteLine("Probamos a leer el archivo ")
    Try
        FileOpen(Canal, Nombre, OpenMode.Input) ` Secuencial
        For Posicion = 1 To 12
            FileGet(Canal, Registro, Posicion)
            Console.WriteLine("Leído {0} ", Registro.NombreMes)
        Next

        ' Cierre del Archivo
        FileClose(Canal)
        Console.WriteLine("El archivo ha sido leído correctamente.")
    Catch Ex as Io.Exception
        Console.WriteLine("No se ha podido abrir el archivo")
    Finally
        Console.WriteLine("Final del proceso de lectura")
    End Try
End Sub

```

En el siguiente ejemplo se ha tratado de capturar un error como el de archivo inexistente, y después la situación de error imprevisto, es decir se capturan dos tipos de error en una misma sentencia Try. No se puede utilizar otro objeto distinto a Exc en la línea donde se escribe el Catch.

```

Module Ejemplo
    Structure RegRandom
        <VBFixedString(12)> Public NombreMes As String
    End Structure

    Private Sub Lee()
        Dim Registro As RegRandom
        Dim Canal As Int16 = FreeFile()
        Dim Nombre As String = "No existe.Txt"
        ' Dim Nombre As String = "C:\No hay\No existe.Txt"
        Dim Posicion As Int16

        Console.WriteLine()
        Console.WriteLine("Probamos a leer el archivo ")
        Try
            ' probar a cambiar el nombre para el file not found,

            System.IO.File.OpenText(Nombre) ` Archivo de texto.
            ' Cierre del Archivo
            FileClose(Canal)
        Catch exc As System.IO.FileNotFoundException
            Console.WriteLine("No se ha podido encontrar el archivo")
        Catch exc As System.IO.DirectoryNotFoundException
            Console.WriteLine("No se ha podido encontrar el directorio")
        Catch exc As System.IO.IOException
            Console.WriteLine("Otros errores.")
        Finally
            Console.WriteLine("Final del proceso de lectura")
        End Try
    End Sub
End Module

```

```

Sub main()
    Lee()
End Sub
End Module

```

El objeto Exc utilizado en el Catch debe ser un objeto predefinido en la clase System

El ejemplo siguiente es equivalente al anterior pero usando On Error Goto Errores

```

Module Ejemplo
    Structure RegRandom
        <VBFixedString(12)> Public NombreMes As String
    End Structure

    Private Sub Lee()
        On Error GoTo Errores
        Dim Registro As RegRandom
        Dim Canal As Int16 = FreeFile()
        Dim Nombre As String = "C:\No hay\No existe.Txt"
        Dim Nombre As String = "No existe.Txt"
        Dim Posicion As Int16

        Console.WriteLine()
        Console.WriteLine("Probamos a leer el archivo")
        System.IO.File.OpenText(Nombre) ' Archivo de texto

        ' Cierre del Archivo
        FileClose(Canal)

Salida:

        Console.WriteLine("Final del proceso de lectura")

        Exit Sub

Errores:

        Select Case Err.Number
            Case 53
                Console.WriteLine("No se ha podido encontrar el archivo")
                Resume Salida
            Case 75
                Console.WriteLine("No se ha podido encontrar el directorio")
                Resume Salida
            Case Else
                Console.WriteLine("Error imprevisto.")
                Resume Salidaa
        End Select

    End Sub

    Sub main()
        Lee()
    End Sub
End Module

```

El uso del On Error sigue siendo más practico y comodo que el del Try.

5.4 Envío y recepción de Arrays.

El envío de un array a un procedimiento se hace colocando el nombre del array en la línea de llamada.

La recepción se hace indicando el nombre dentro del Procedimiento seguido de paréntesis vacíos y declarando el tipo del Array.

No tiene porque coincidir el nombre del array en el envío y en la recepción.

El número de elementos es definido en el programa principal.

```
Module Ejemplo
Structure RegRandom
    <VBFixedString(12)> Public NombreMes As String
End Structure

Private Sub GrabaArchivo(ByVal NomM() As String)
    Dim Registro As RegRandom
    Dim Canal As Int16 = FreeFile()
    Dim Nombre As String = "PruebaRandom.Txt"
    Dim Posicion As Int16
    Dim Valor As String

    Console.WriteLine("Probamos a grabar registros al archivo ")
    FileOpen(Canal, Nombre, OpenMode.Random, , , Len(Registro))
    For Each Valor In NomM
        Posicion += 1
        Registro.NombreMes = [Strings].LSet(Valor, 12)
        FilePut(Canal, Registro, Posicion)
        Console.WriteLine("Grabado {0} ", Valor)
    Next
    FileClose(Canal)      ' Cierre del Archivo
End Sub

Private Sub Lee()
    Dim Registro As RegRandom
    Dim Canal As Int16 = FreeFile()
    Dim Nombre As String = "PruebaRandom.Txt"
    Dim Posicion As Int16

    Console.WriteLine()
    Console.WriteLine("Probamos a leer el archivo ")
    FileOpen(Canal, Nombre, OpenMode.Random, , , Len(Registro))

    For Posicion = 1 To 12
        FileGet(Canal, Registro, Posicion)
        Console.WriteLine("Leído {0} ", Registro.NombreMes)
    Next

    ' Cierre del Archivo
    FileClose(Canal)

End Sub

Sub main()
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                             "Abril", "Mayo", "Junio", _
                             "Julio", "Agosto", "Septiembre", _
                             "Octubre", "Noviembre", "Diciembre"}

    GrabaArchivo(Meses)
    Lee()
End Sub
End Module
```


6. Funciones de usuario.

6.1 Introducción.

Pueden ser públicas o privadas, protegida o amigo.

Pueden aparecer en una estructura, en una clase, o en un formulario.

Las variables declaradas en su interior son propias al procedimiento, locales.

La recepción de las variables exteriores se hace con las reglas ya conocidas, por defecto es por valor, no por referencia como antes, y delante de la variable aparecerá siempre, ByVal por valor, o ByRef por referencia.

Igual que en Vb6, podemos declarar variables opcionales, Optional, pero han de estar inicializadas a un valor por defecto.

La salida será en la última línea, End function, o cuando encuentra la instrucción Exit Function, cuando hay tratamiento de errores.

La novedad en esta versión es la posibilidad de que se puedan devolver valores en la función en forma de Array, cosa que VB6, no era posible.

También es posible recibir un ParamArray, que significa que podemos recibir un número variable de datos en la entrada y se reciben en una estructura en forma de Array.

6.2 Ejecución Normal.

En el ejemplo que sigue tenemos una función que es el mínimo que se puede decir de la misma.

```
Module Ejemplo
  Const Pi = 3.141516

  Private Function Sencilla() As Single
    Sencilla = Pi
  End Function

  Sub Main()
    Console.WriteLine(Sencilla)
  End Sub
End Module
```

En el siguiente ejemplo la función recibe un parámetro y realiza un cálculo, para devolver un resultado.

En el ejemplo utilizamos la instrucción de consola ReadLine, para la introducción del Radio de la circunferencia.

```
Module Ejemplo
  Const Pi = 3.141516

  Private Function LongCircunf(ByVal Radio As Single) As Single
    LongCircunf = 2 * Pi * Radio
  End Function

  Sub Main()
    Console.WriteLine(LongCircunf(Console.ReadLine()))
  End Sub
End Module
```

6.3 Tratamiento de errores.

El siguiente ejemplo de función podemos observar como insertamos la captura de errores con el uso del On Error Goto Etiqueta.

Se puede utilizar cualquiera de los dos formatos de captura de errores en las funciones.

```

Module Ejemplo
    Const Pi = 3.141516

    Private Function AreaCircunf(ByVal Radio As Single) As Single
        On Error GoTo Errores

        AreaCircunf = 0
        AreaCircunf = Pi * Radio * Radio

Salida:

        Exit Function

Errores:

        Select Case Err.Number
            Case 5
                ` El formato este no es válido, solo como ejemplo.
                Console.WriteLine("El valor es inadecuado")
                Resume Salida
            Case Else
                Console.WriteLine(Err.Number)
        End Select

    End Function

    Sub Main()
        Console.WriteLine(AreaCircunf(Console.ReadLine()))
    End Sub
End Module

```

EL uso de On Error Goto Etiqueta, aún es factible de más opciones, pero ya son las variantes de dicha opción.

En la captura de errores se tratarán aquellos códigos de error que son susceptibles de ocurrir en función del código escrito.

El valor inicial que se le de a la función es interesante para que siempre tenga un valor de retorno conocido en caso de error.

El ejemplo que sigue es con el uso de Try

```

Module Ejemplo

    Private Function Fallo() As Single
        Dim X As Int16 = 0
        Fallo = 0
        Try
            Fallo = 4 \ X
            Catch exc As System.DivideByZeroException
                Console.WriteLine("división por cero")
            Catch ex As Exception
                Console.WriteLine("El valor es inadecuado")
        End Try
    End Function

    Sub Main()
        Console.WriteLine(Fallo)
    End Sub
End Module

```

6.4 Envío y recepción arrays.

No hay diferencia con respecto a los procedimientos.

El ejemplo que hay a continuación es muy sencillo, pero ilustrativo.

El ejemplo no tiene nada más que valor de ejemplo pues es absolutamente irreal.

```
Module Ejemplo
  Private Function NombreMes (ByVal Meses() As String, _
                              ByVal Mes As Int16) _
                              As String
    NombreMes = ""
    Select Case Mes
      Case 1 To 12
        NombreMes = Meses (Mes - 1)
    End Select
  End Function

  Sub main()
    Dim Meses() As String = {"Enero", "Febrero", "Marzo", _
                              "Abril", "Mayo", "Junio", _
                              "Julio", "Agosto", "Septiembre", _
                              "Octubre", "Noviembre", "Diciembre"}

    Console.WriteLine (NombreMes (Meses, 1))
  End Sub
End Module
```

6.5 Recepción variable de argumentos, ParamArray.

El ejemplo que sigue recibe un número indeterminado de datos y los trata en un array en el interior de la función.

```
Module Ejemplo

  Private Function Suma (ByVal ParamArray V() As Integer) As Long
    Dim X As Integer = 0
    While V.Length > X
      Suma = Suma + V(X)
      X = X + 1
    End While
  End Function

  Sub Main()
    Console.WriteLine ("El resultado de la función es {0} ", Suma(1, 2, 3))
    Console.ReadLine()
  End Sub

End Module
```

6.6 Valores opcionales, Optional.

Podemos declarar valores opcionales en la línea de argumentos, y esos valores darles un valor de inicialización por defecto, que puede ser nulo o cero en función del tipo de dato..

No pueden coincidir con una entrada con ParamArray.

```

Module Ejemplo

    Private Function Funciona(ByVal Entra As Integer, _
                               Optional ByVal Otro As Integer = 2) _
                               As Integer
        Funciona = Entra + Otro
    End Function

    Sub Main()
        Console.WriteLine("El resultado es {0} ", Funciona(4))    ` Será 6
        Console.WriteLine("El resultado es {0} ", Funciona(4 + 5)) ` Será 11
        Console.ReadLine()
    End Sub
End Module

```

Si se inicializa a cero, lo normal, los valores serán ...

```

Module Ejemplo

    Private Function Funciona(ByVal Entra As Integer, _
                               Optional ByVal Otro As Integer = 0) _
                               As Integer
        Funciona = Entra + Otro
    End Function

    Sub Main()
        Console.WriteLine("El resultado es {0} ", Funciona(4))    ` Será 4
        Console.WriteLine("El resultado es {0} ", Funciona(4 + 5)) ` Será 9
        Console.ReadLine()
    End Sub
End Module

```