VB.net

Datos por valor vs por referencia, arrays, matrices, colecciones y procedimientos

Tipos de datos por valor vs por referencia I

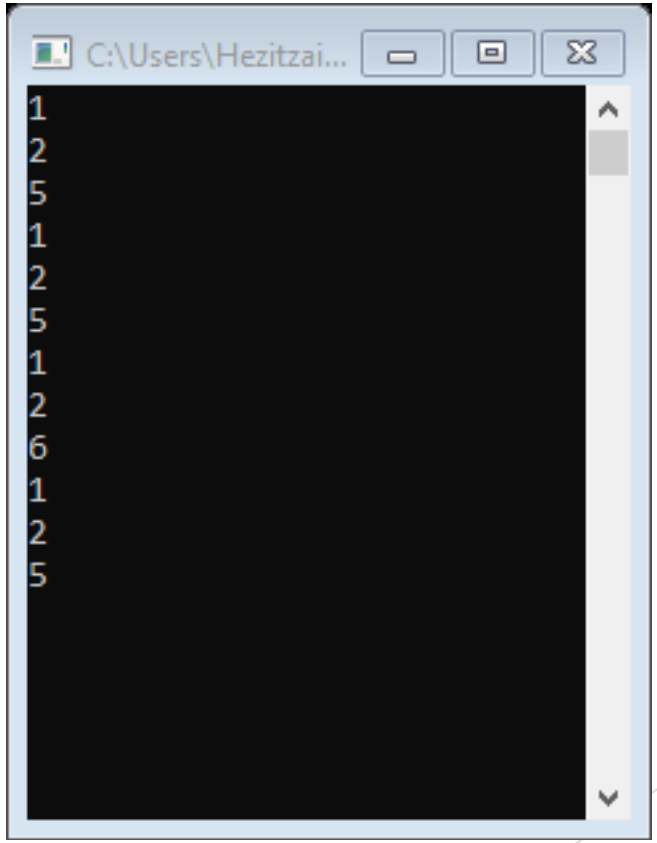
- Hasta ahora todos los tipos de datos eran por valor, es decir, el CLR reservaba un espacio de memoria para ellos y se guardaban ahí los datos.
- Los objetos que se crean a partir de una clase, como los arrays, son un tipo de datos por referencia, es decir, no reservan espacio de memoria hasta que usamos la instrucción New.
- Cuando tenemos un dato por referencia hasta que no reservemos espacio en memoria para el, este dato será un puntero que apunta a un objeto, por lo que si el objeto cambia el dato también cambiara.

Tipos de datos por valor vs por referencia II

```
Sub main()
    Dim miArray As Integer() = New Integer() {1, 2, 3}
    Dim miArray2 As Integer()
    miArray2 = miArray
    miArray(2) = 5
    For Each num In miArray
        Console.WriteLine(num)
    Next
    For Each num In miArray2
        Console.WriteLine(num)
    Next

    ReDim Preserve miArray2(2)
    miArray(2) = 6
    For Each num In miArray
        Console.WriteLine(num)
    Next
    For Each num In miArray2
        Console.WriteLine(num)
    Next

    Console.ReadLine()
End Sub
```



Arrays estáticos unidimensionales I

- Para declarar un array usamos esta instruccion:

  Dim nombre() As tipo

- Estas instrucciones no reservan espacio en memoria. Para hacerlo tenemos que darle una dimensión al array. Podemos hacerlo de tres maneras.

  Declararlo con tamaño: Dim nombre(tamaño*) As tipo

  Dim nombre = New tipo(tamaño*) {}

  Declararlo con valores: Dim nombre() As tipo ={valor1, valor2, …}

  Dim nombre =New tipo() {valor1, valor2,…}

  Redimensionándolo : Redim nombre(tamaño*)

  Redim Preserve nombre(tamaño*)

Preserve: Mantiene los datos del array al dimensionar la última dimensión de un array. No puede alterar el rango, ni tipo de datos.

Arrays estáticos unidimensionales II

- `Dim arr1(5) As Integer`
- `Dim arr2 = New String(5) {}`
- `Dim arr3() As Integer = {1, 2, 3, 4}`
- `Dim arr4 = New Integer() {1, 2, 3, 4, 5}`

Arrays estáticos unidimensionales III

- La clase array tiene diferentes propiedades y métodos:
  - Conocer el tamaño: array.Lenght
  - Ordenar el array: Array.Sort(arrayOrigen)
  - Copiar el array: arrayOrigen.CopyTo(arrayDestino, posición)

.Sort: No se puede usar en arrays de más de una dimensión

- Podéis encontrar más en
  - https://docs.microsoft.com/es-es/dotnet/api/system.array?view=netframework-4.8

Matrices I

- Básicamente funcionan como los arrays unidimensionales, pero se pone una coma por cada dimensión del array. Para declararlo:

  Dim nombre(,) As tipo

- Para reservar espacio en memoria:

  Dim nombre(tamaño1*, tamaño2*) As tipo

  Dim nombre = New tipo (tamaño1*, tamaño2*) {}

  Dim nombre(,) As tipo = {{valor1, valor2}, {valor3,valor4},…}

  Dim nombre = New tipo(,) {{valor1, valor2}, {valor3,valor4},…}

- Para saber el tamaño de cada nivel de la matriz podemos usar su método array.GetLenght(nivel)

Matrices II

- `Dim matriz1(1, 2) As Integer`
- `Dim matriz2 = New Integer(1, 2) {}`
- `Dim matriz3(,) As Integer = {{1, 2}, {3, 4}}`
- `Dim matriz4 = New Integer(,) {{1, 2}, {3, 4}}`

Matrices escalonadas I

- Cuando la estructura de datos es bidimensional, pero no rectangular se usan matrices escalonadas. Por ejemplo, para recoger la temperatura más alta en cada día del mes en un año.
- Para crear una matriz escalonada usamos para declararlo:

  Dim nombre()() As tipo

- Y para reservar espacio:

  Dim nombre(tamaño1)() As tipo

  Dim nombre()() As tipo = New tipo(tamaño)() {}

  Dim nombre()() As tipo = {({valor1, valor2,…}), ({valor3,valor4,…}),…}

- En las matrices escalonadas al usar el método array.GetLenght(1) nos dará una excepción porque el tamaño de la segunda dimensión no es constante.

Matrices escalonadas II

- `Dim matriz1()() As Integer`
- `Dim matriz2(3)() As Integer`
- `Dim matriz3()() As Integer = New Integer(3)() {}`
- `Dim matriz4()() As Integer = {({1, 2}), ({3, 4, 5}), ({6, 7})}`

Arrays dinámicos o colecciones

- Pueden aumentar y reducirse de manera dinámica, no tienen un tamaño fijo.
- Para crearlos podemos hacerlo de las siguientes maneras:

  Dim nombre As New List(Of tipo)

  Dim nombre As New List(Of tipo) From {valor1, valor2, valor3…}

- .Count:
  - Devuelve el número de elementos reales que contiene: lista.Count

Arrays dinámicos o colecciones II

- Algunos métodos de las listas
  - Añade un elemento a la lista: lista.Add(valor)
  - Añade un elemento a la lista en un índice concreto: lista.insert(indice, valor)
  - Elimina de la lista el primer elemento que coincida con el valor: lista.Remove(valor)
  - Elimina de la lista el elemento que esta en el índice: lista.RemoveAt(indice)
  - Elimina de la lista tantos elementos como la cantidad empezando por el indice: lista.RemoveRange(indice, cantidad)
  - Devuelve true si el valor esta en la lista y false si no esta: lista.Contains(valor)
  - Ordena la lista: lista.Sort()
  - Busca el índice del primer elemento que coinciden con el valor: lista.IndexOf(valor)
  - Elimina los elementos de la lista. lista.clear()

Arrays dinámicos o colecciones III

- Existen otro tipo de colecciones diferentes a las listas.
  - Dictionary: Representa una colección de pares clave y valor que se organizan según la clave.
  - Queue: Representa una colección de objetos FIFO.
  - Stack: Representa una colección de objetos LIFO.

- Podéis echar un ojo a estas listas en la siguiente dirección
  - https://docs.microsoft.com/es-es/dotnet/visual-basic/programming-guide/concepts/collections

For Each … [As tipo] In …

- Para recorrer arrays, matrices o colecciones podemos usar el bucle For Each … [As tipo] In …

```
For Each esteAnimal As String In animales
    console.writeline(esteAnimal)
Next
```

Procedimientos I

- Existen dos tipos de procedimientos:
  - Sub -> No devuelven ningún valor

    Llamada-> NombreMetodo(variable1, variable2…)

    Sub NombreMetodo (variable1 As tipo, variable2 As tipo…)

    instrucciones

    End Sub

  - Function -> Devuelven un valor a la instrucción que los invoca

    Llamada-> variablePrincipal=NombreFunción(variable1, variable2…)

    Function NombreFunción (variable1 As tipo, variable2 As tipo…) As tipo

    instrucciones

    Return variableSecundaria o NombreFunción= variableSecundaria

    End Function

*Es recomendable escribir el nombre de los procedimientos con la primera letra en mayúsculas.

Procedimientos II: Argumentos

- Al pasar argumentos a un procedimiento, podemos pasarlos por valor(ByVal) o por referencia(ByRef).
- Por defecto se mandan por referencia.
- También podemos usar argumentos opcionales. Para ello tendremos que poner Optional delante de la declaración del argumento y darle un valor por defecto.

```
Private Function CalcularArea(lado1 As Double, Optional lado2 As Double = 0) As
Double

    If lado2 = 0 Then Return lado1 ^ 2 Else Return lado1 * lado2

End Function
```

Procedimientos III: Accesibilidad

Visual Basic .NET	C#	Descripción del ámbito
Private	private	Accesible dentro del mismo módulo, clase o estructura.
Friend	internal	Accesible desde dentro del mismo proyecto, pero no desde fuera de él.
Protected	protected	Accesible desde dentro de la misma clase o desde una clase derivada de ella.
Protected Friend	protected internal	Accesible desde clases derivadas o desde dentro del mismo proyecto, o ambos.
Public	public	Accesible desde cualquier parte del mismo proyecto, desde otros proyectos que hagan referencia al proyecto, y desde un ensamblado generado a partir del proyecto.

*De forma predeterminada todos los procedimientos son Public