Introduction to VB.net

Lecture 3

Nasrin Khodapanah

VB.NET

Agenda

- Visual Studio nice to know
- Operators
- Strings Manipulation
- Math Manipulation
- Date and Time
- Arrays
- Enum

Visual Studio Nice to know

Visual Studio Nice to Know

- Line Numbers
  - Tools/Options/Editor/Visual Basic/Line numbers
- Debugging
  - Break Point – Run – (F5)
    - F11 – Step Into (next)
    - F10 – Step Over
    - SHIFT + F11 – Step Out

Visual Studio Nice to Know

- Solution Explorer
  - Renaming a Module
  - Renaming Project
  - Project Location
  - Adding more Module
  - Save Project
  - Explore Saved Folder
  - ETC…

Operators

Assignment operators

= Assignment

^= Exponentiation followed by assignment

*= Multiplication followed by assignment

/= Division followed by assignment

\= Integer division followed by assignment

+= Addition followed by assignment

-= Subtraction followed by assignment

&= Concatenation followed by assignment

Comparison operators

< (Less than)—True if operand1 is less than operand2

<= (Less than or equal to)—True if operand1 is less than or equal to operand2

> (Greater than)—True if operand1 is greater than operand2

>= (Greater than or equal to) – True if operand1 is greater than or equal to operand2

Comparison operators

= (Equal to)—True if operand1 equals operand2

<> (Not equal to)—True if operand1 is not equal to operand2

IS—True if two objects' references refer to the same object

LIKE—Performs string pattern matching

Logical/Bitwise operators

- **AND**— Performs an **AND** operation (for logical operations: **True** if both operands are **True**, **False** otherwise; the same for bit-by-bit operations where you treat **0** as **False** and **1** as **True**).
- **Not**— Reverses the logical value of its operand, from **True** to **False** and **False** to **True**, for bitwise operations, turns **0** into **1** and **1** into **0**.

Logical/Bitwise operators

- **OR**— Operator performs an **OR** operation (for logical operations: **True** if either operand is **True**, **False** otherwise; the same for bit-by-bit operations where you treat **0** as **False** and **1** as **True**).

Logical/Bitwise operators

- **XOR**— Operator performs **an exclusive-OR** operation (for logical operations: **True** if either operand, but not both, is **True**, and **False** otherwise; the same for bit-by-bit operations where you treat **0** as **False** and **1** as **True**).

Logical/Bitwise operators

- **AndAlso**— Operator **A** "short circuited" **AND** operator; if the first operand is **False**, the second operand is not tested.
- **OrElse**— Operator **A** "short circuited" **OR** operator, if the first operand is **True**, the second is not tested.

Warning
short-circuiting

In Visual Basic .NET, even if the first operand of an AND operator evaluates to False, the remainder of the logical expression is still evaluated. Similarly, even if the first operand of an OR operator evaluates to True, the remainder of the logical expression still evaluated. This is why you have to short-circuit it with ANDALSO and/or ORELSE.

Strings manipulation

Trim

```
Dim my_string, trimmed_string As String
my_string = "#####My ### String!#######"
trimmed_string = my_string.Trim("#")
Console.WriteLine(my_string )
Console.WriteLine(trimmed_string )
```

Remove - Replace

```
Dim removed_string, replaced_string As String
removed_string = trimmed_string.Remove(3, 4)
replaced_string = removed_string.Replace("My",
"Another")
```

Upper Case

```
Dim upper_string1, upper_string2 As String
upper_string1 = replaced_string.ToUpper
upper_string2 = UCase(replaced_string)
```

Lower Case

```
Dim lower_string1, lower_string2 As String
lower_string1 = replaced_string.ToLower
lower_string2 = LCase(replaced_string)
```

Math Functions

Math Functions

Math.Abs(n)

Math.Min(n1, n2)

Math.Max(n1, n2)

Math.Sqrt(n1)

Math.Pow(n1, n2)

Date and Time

Date

- Date.Now – Give today’s date and time (sql)
- To get the date only with no name (abbreviation)
  - ToShortDateString()
- To get the date with names (long date)
  - ToLongDateString()

Time

- Date.Now – Give today’s date and time (sql)
- To get the time (HH:MM)
  - ToShortTimeString()
- To get the time (HH:MM:SS)
  - ToLongTimeString()

Time

- MMM Three-letter month.
- ddd Three-letter day of the week.
- dddd Full day name
- d Day of the month.
- HH Two-digit hours on 24-hour scale.
- mm Two-digit minutes.
- yyyy Four-digit year.

Date and Time Exercice

Clock – Exo3

1- Create a real-time Clock

- System.Threading.Thread.Sleep(1000)
    - Pause the program for 1000 ms
- Console.Clear()
    - Clear console ☺

Arrays

1D Array

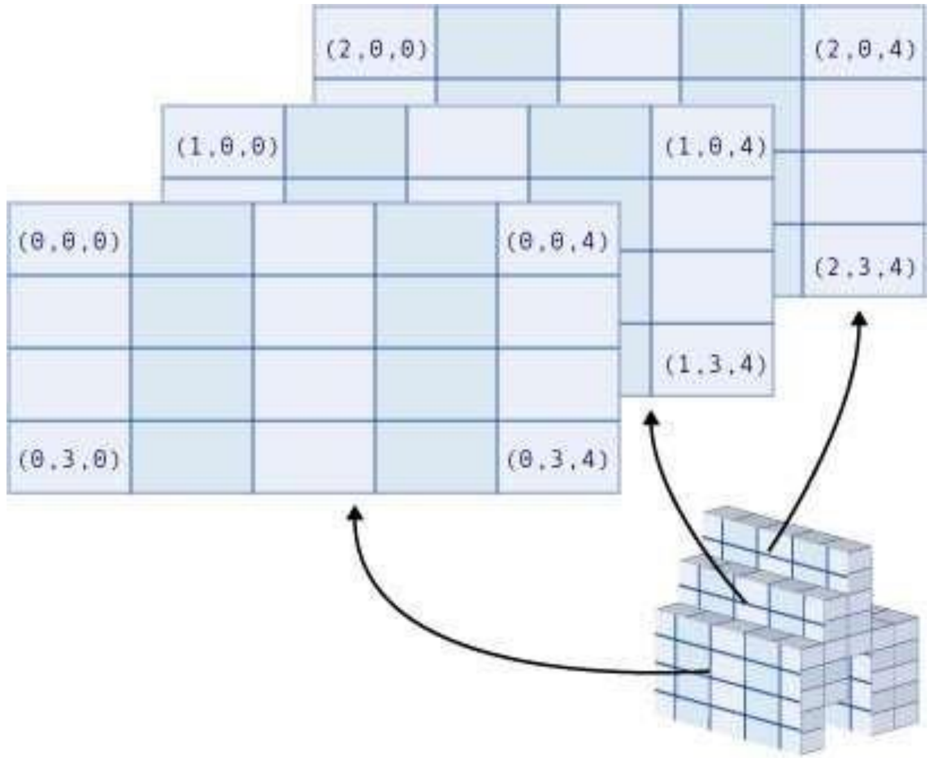
- Dim my_array (4) As Integer
  - (4) array Size = 5
  - (4) array 5 indexes (0, 1, 2, 3, 4)

2D Array

- Dim my_array (3,4) As Integer
- my_array(0,0) – First square

3D Array

- Dim my_array(2,3,4) As Integer
- My_array(0,0,0) – First square



Array Manipulation

- Redim
  - Modify array size
    - Redim my_array(20)

- Redim Preserve
  - Modify array size
    - Redim Preserve my_array(20)

Array Manipulation

```
Dim my_array(25) As Integer
For i As Integer = 0 To my_array.Length - 1
        my_array(i) = Rnd() * 10 'generates a random integer
        between 0 and 10 (inclusive)
Next

For i As Integer = 0 To my_array.Length - 1
    Console.WriteLine(my_array(i))
Next
```

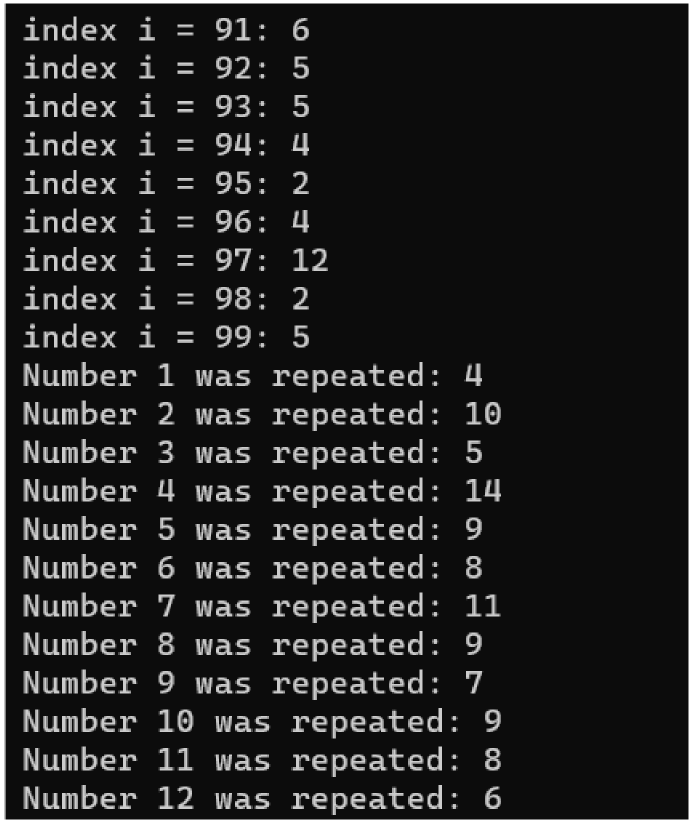
Array Exercises

EXO3 - Array

2- Create a 1D Array of size 100 and fill it with random numbers between 0 – 13 (exclusive). Use the function RND in order to get random numbers.

a. Display the array
b. Count how many of each number was found in the Array.

Array Exo



```
index i = 91: 6
index i = 92: 5
index i = 93: 5
index i = 94: 4
index i = 95: 2
index i = 96: 4
index i = 97: 12
index i = 98: 2
index i = 99: 5
Number 1 was repeated: 4
Number 2 was repeated: 10
Number 3 was repeated: 5
Number 4 was repeated: 14
Number 5 was repeated: 9
Number 6 was repeated: 8
Number 7 was repeated: 11
Number 8 was repeated: 9
Number 9 was repeated: 7
Number 10 was repeated: 9
Number 11 was repeated: 8
Number 12 was repeated: 6
```

Enum

Enum

Enum IS NOT an Array

Defines linked variables

OUTSIDE OF THE MAIN

In visual basic, Enum is a keyword and it is useful to declare an enumeration. In visual basic, the enumeration is a type that will contain a set of named constants as a list. By using enumeration, we can group constants that are logically related to each other. For example, the days of the week can be grouped by using enumeration in visual basic.

Enum

Enum days

    Sunday = 10

    Monday (by default 11 <- if not given a value; Following previous entry as Sunday = 10)

    Tuesday = 26

    Wednesday = 11

    Thursday (by default 12)

    Friday = 30

    Saturday (by default 31)

End Enum

Enum

In the Main

```
Sub Main()
    Dim day As days
    day = days.Monday
     Console.WriteLine(day)
    Console.Read()
  End Sub
```

Don`t forget to upload the Exo file