

## Unit-2

### Programming with VB.net

#### Syllabus:-

- Data types with Programmes
- Variables, Constant, Type Conversion
- Operators (all), Operator precedence Ex.
- Control Structure: Conditional statements (IF-Else, Select Case) with Programs
- Loops : Do Loops, For loops, While Loop, For each ....Next Loop
- Various Programs based on Condition and Loops
- Array: Declaring arrays and Dynamic arrays
- Types, Structure, Enumeration
- Sub Procedure
- Functions, Passing argument types

#### Question Bank

- What is VB.Net.
- How to Create Project in VB.Net? or Write down the steps to create a VB.Net solution
- What is Console application. Create a console application?
- Write control structure of VB.Net with example?
- Differentiate between function and subroutine with example?
- Explain primitive data types used in Vb.net with range and length?
- Explain various conditional statements with example?
- How to create enumeration in VB.Net? Explain with an example.
- Describe Array and dynamic array and what are benefits of dynamic array over static array?
- What is Procedure? Describe Types of procedure.
- Explain function with example.
- What is Control Loops and any 2 loop structure.
- What is control structure explain any 3 structure.
- Explain structure with example.

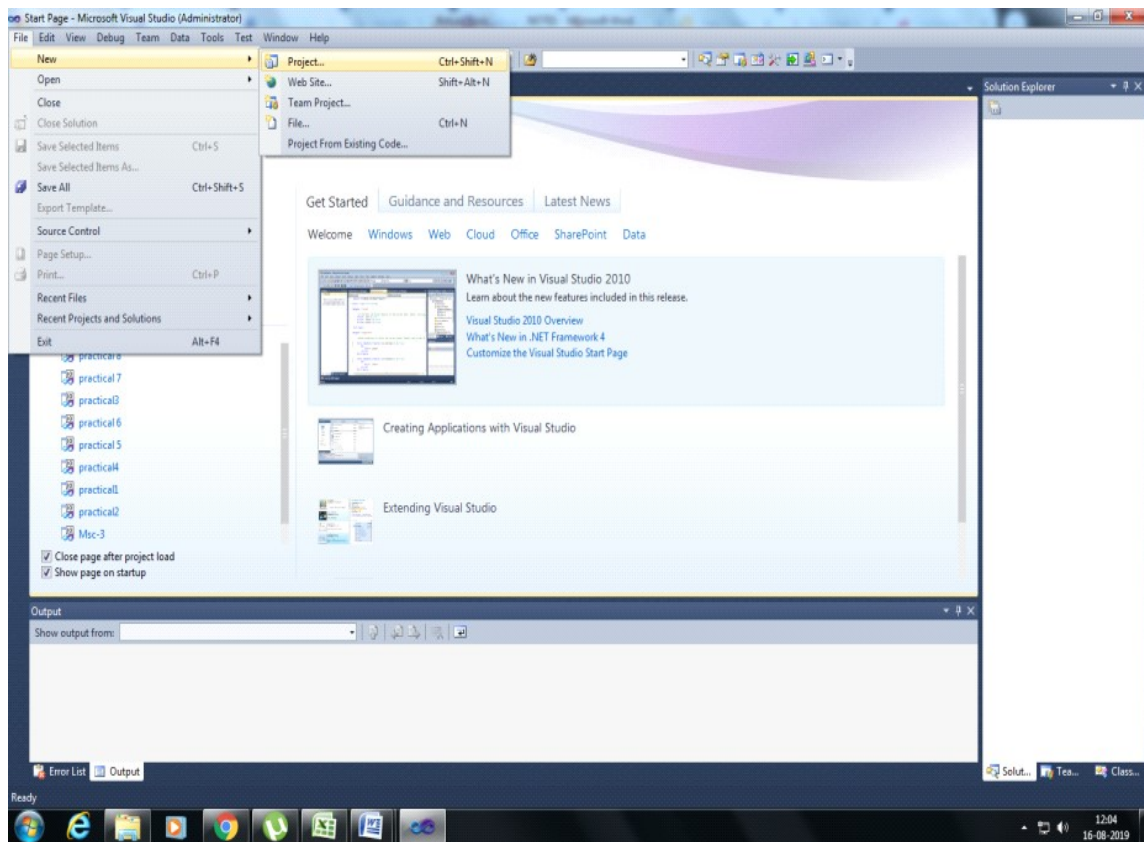
- Explain the differences between a value type and a reference type with example.
- Describe type conversion and its different types of types conversion function.
- Write different types of operators with example.

- **Q . What is VB.NET**

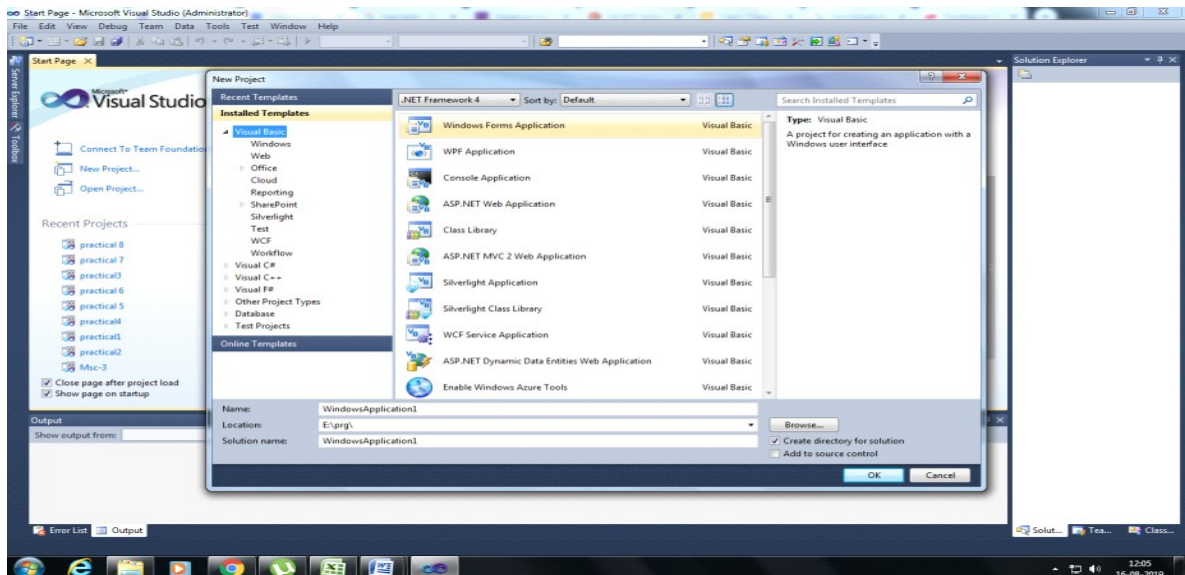
- Visual Basic .NET (VB.NET) is an object-oriented programming (OOP) language developed by Microsoft. It evolved from Visual Basic 6 (VB6) to develop web-services and web applications. It was designed to take advantage of the .NET framework-based classes and run-time environment. It is a part of .NET product group. VB.NET supports abstraction, inheritance, and polymorphism.
- . NET provides a new API, new functionality, and new tools for writing Windows and Web applications, components, and services in the Web age..NET provides a new, object-oriented API as a set of classes that will be accessible from any Programming language.
- **How to Create Project in VB.Net? or Write down the steps to create a VB.Net solution**

**Ans.** Following are the steps to create a project in VB.Net.

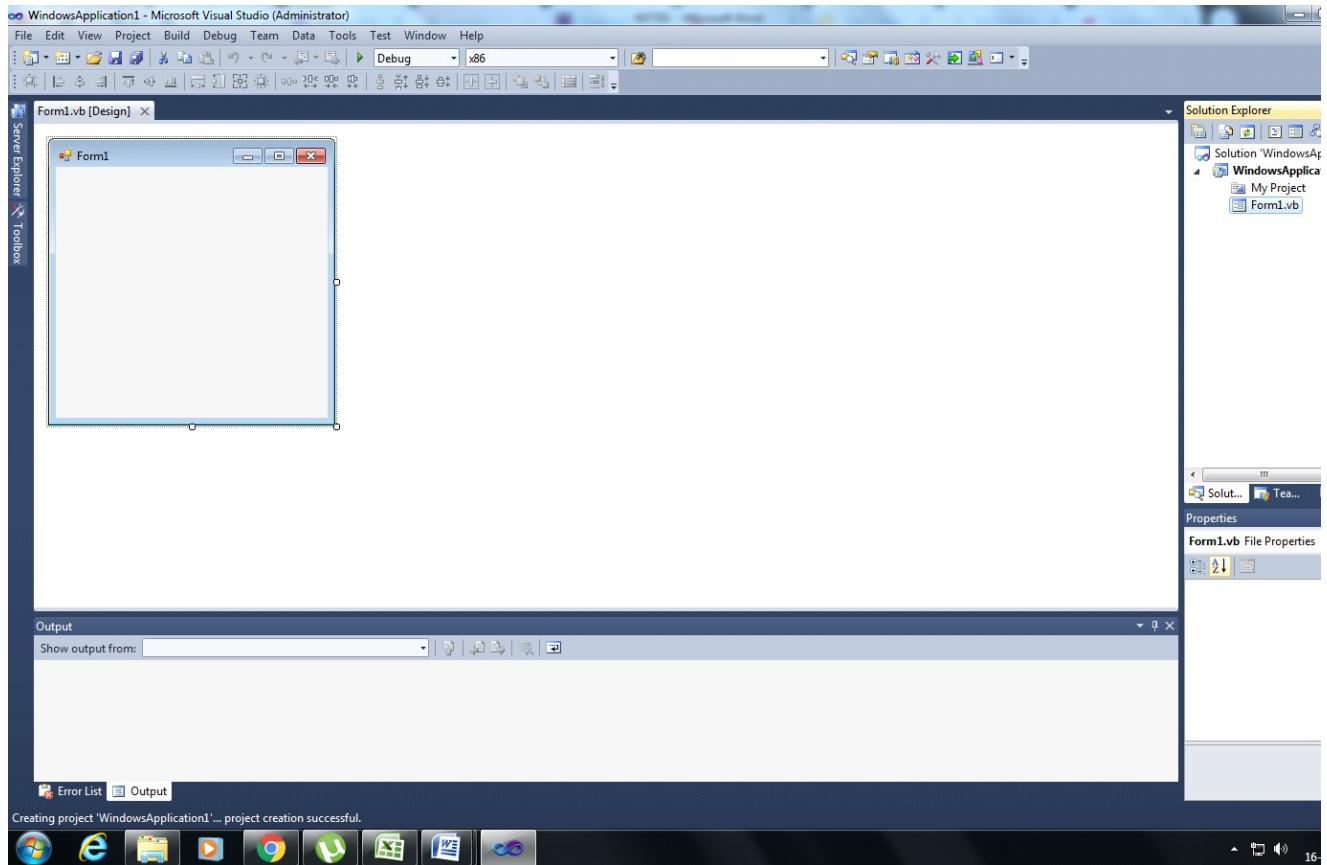
- To create a new project in VB.NET, click on the New Sub-menu under File menu.



- From which the 'New Project Dialog Box' is displayed.

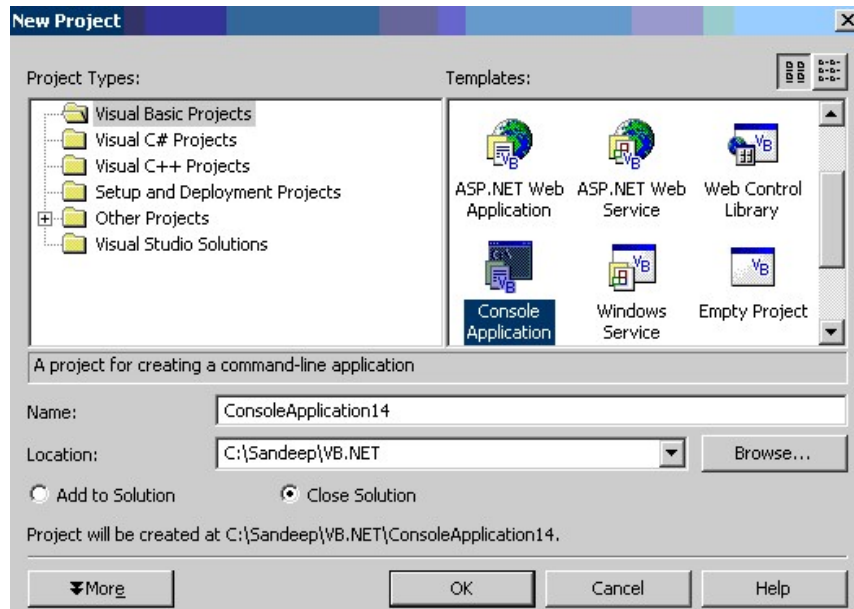


- From the Type of Project displayed in this window, we select the project that we want to create. Some of these project types are Visual Basic Projects, Visual C ++ Projects, Visual C # Projects, etc. Each of these project types has different applications; For example, from Visual Basic Project Type we can create Window Application, Console Application etc.



- In this window we will design the form according to our requirement.
- **Visual Basic Dot Net projects have different types of Templates, which are as follows –**
  - Windows application
  - Class library
  - Windows control library
  - Smart device application
  - ASP.NET Web Application
  - ASP.NET Mobile Web Application
  - ASP.NET Web Services
  - Web control library
  - Console application
  - Windows services

- Empty project
- Empty web project
- Console Applications
  - **What is Console application. Create a console application?**
- Ans. Console Applications are command-line oriented applications that allow us to read characters from the console, write characters to the console and are executed in the DOS version. Console Applications are written in code and are supported by the System.Console namespace.
- **Example on a Console Application**
- Create a folder in C: drive with any name (say, examples) and make sure the console applications which you open are saved there. That's for your convenience. The default location where all the .NET applications are saved is C:\Documents and Settings\Administrator\My Documents\Visual Studio Projects. The new project dialogue looks like the image below.



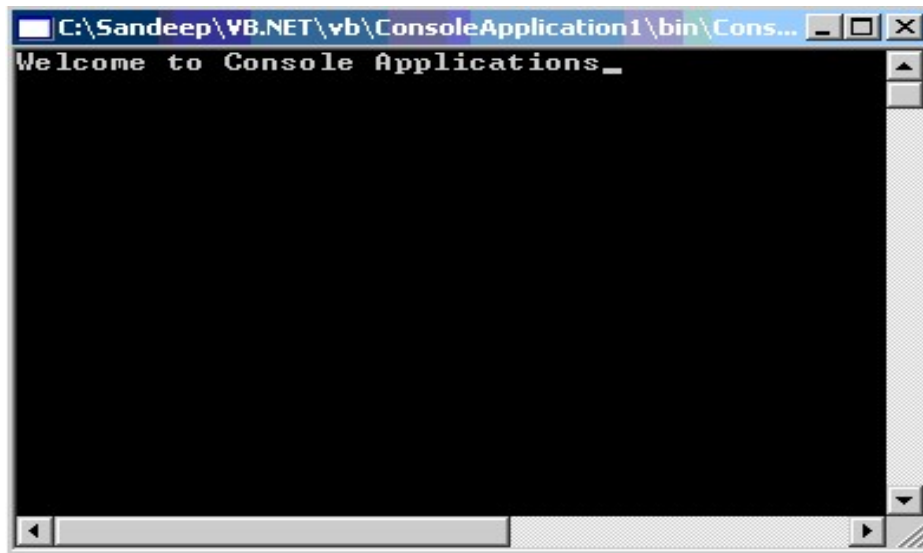
- The following code is example of a console application:

```
Module Module1

Sub Main()
System.Console.WriteLine("Welcome to Console Applications")
End Sub

End Module
```

- The result, "Welcome to Console Applications" is displayed on a DOS window as shown in the image below.



### Q. What are Data types in VB.Net?

**Ans.** When a variable is declared, then we also have to write its data type, because the value to be stored in any variable depends on its data type. The type of data type of the variable will be the same type. Value store can be done.

**DATATYPE** in a programming language describes that what type of data a variable can hold. When we declare a variable, we have to tell the compiler about what type of the data the variable can hold or which data type the variable belongs to.

**Syntax :** Dim VariableName as DataType

**VariableName :** the variable we declare for hold the values.

**DataType :** The type of data that the variable can hold

In Programming, data types define the type of data. Data types tell the type of data to be stored as well as its data storage length. The variables in Visual Basic are divided into the following five categories.

- Numeric
- String
- Boolean
- Date
- Object
- **Numeric Variables:** Numbers in which decimal points are not used are called Integer numbers. It consists of data types that store numbers. They are used to perform mathematical operations. It contains the following data type's-
  - **Byte:** This is also a numeric data type that stores values from 0 to 255. It has 1 byte in memory
  - **Short (Int16):** It is used to store integer value. In this, the value can be stored from -32768 to 32767 and it takes 2 bytes in memory.

Dim a as short

a = 10000

- **Integer(Int32):** It is used to store integer value. In this, the value can be stored from -2147483648 to 2147483647 and it takes 4 bytes in memory.

Dim a as integer

a = 1000000

- **Long (Int64):** It is used to store integer value. It can store more value than integer and it takes 8 bytes in memory.

**Integer** variables are stored signed 32 bit integer values in the range of -2,147,483,648 to +2,147,483,647

- VB.NET Runtime type : System.Int32
- VB.NET declaration : dim count as Integer
- VB.NET Initialization : count = 100
- VB.NET default initialization value : 0

- **Floating point Numbers:** Numbers in which decimal points are used, are called floating point Numbers.

- **Single:** It stores single precision floating no. It can store values from -3.402823E38 to -1.401298E-45 and 1.401298E-45 to 3.402823E38. It **takes 4 byte memory.**

- **Double:** It stores double precision floating no. It can store values from -1.797693134486232E308 to -4.94065645841247E-324 and 4.94065645841247E-324 to 1.797693134486232E308. It takes **8 byte memory.**

- **Decimal:** It can store 0 to 7.9228162514264337593543950335 positive and negative values. It takes **16 Byte in memory.**

- **String:** A group of characters is called a string. It is used to store characters. It consists of the following 2 types.

- **String:** It is used only to store text. It stores the set of characters. It can store text up to 2 GB.

- **Char:** It is used to store single character. And **takes 2 byte** space in memory.

**String** variables are stored any number of alphabetic, numerical, and special characters . Its range from 0 to approximately 2 billion Unicode characters.

- VB.NET Runtime type :System.String
- VB.NET declaration : Dim str As String
- VB.NET Initialization :str = "String Test"
- VB.NET default initialization value : Nothing

- **Boolean:** This data type is used to store true or false value. It is mostly used to check conditions.



**Boolean** variables are stored **16 bit numbers** and it can hold only True or false.

- VB.NET Runtime type :System.Boolean
- VB.NET declaration : dim check as Boolean
- VB.NET Initialization : check = false
- VB.NET default initialization value : false
- **Date:** This data type is used to store date and time. In this, use # character to store the date or time.

Dim a as date

a = # 1/4/2005 #

- **Object:** This is a common data type that is used to store all types of values with common features of all data types.
- 
- **Q . What do you understand by Variables? How to declare it?**
  - A variable is a simple name used to store the value of a specific data type in computer memory. In **VB.NET**, each variable has a particular data type that determines the size, range, and fixed space in computer memory.

The basic value types provided in VB.Net can be categorized as –

Type	Example
Integral types	SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char
Floating point types	Single and Double
Decimal types	Decimal
Boolean types	True or False values, as assigned
Date types	Date

- VB.Net also allows defining other value types of variable like **Enum** and reference types of variables like **Class**.

### Declaring Variables

In Visual Basic.Net, the dim statement is used to declare the variable. The following are Syntax.

Syntax: Dim <Variable Name> As <Data Type>

Example: Dim name As String

In this statement Dim states the dimension of the keyword variable, variable name, variable name, As keyword and data type. In Visual Basic.Net, variables are usually declared this way. Also the variable declaration statement changes according to the scope. The public keyword is used in place of Dim to declare the public variable.

Public <Variable Name> As <Data Type>

Similarly, private , Protected and friend keywords are also used to declare variables.. Multiple variables can be declared in a single statement in VB.Net. Example: Dim a, b, c as Integer



Similarly, multiple type variables can also be declared in a single dim statement.  
Example: Dim a As Integer, name As String, b As Boolean

### Variable Naming Conventions

- Variable's name always starts with a letter.
- Variable name should not always be more than 255 character.
- No other character is used except for numbers and underscore in variable name.
- Variable name should always be unique in its scope.
- Visual Basic keywords cannot be used in Variable name.
- Variable name should not contain blank space.

Module Module1

Sub Main()

Dim a As Short

Dim b As Integer

Dim c As Double

a = 10

b = 20

c = a + b

Console.WriteLine("a={0},b={1},c={2}", a, b, c)

Console.ReadKey()

End Sub

End Module

When the above code is compiled and executed, it produces the following result –

a = 10, b = 20, c = 30

### Accepting Values from User

The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

Dim message As String

message = Console.ReadLine

The following example demonstrates it –

Module Module1

Sub Main()

Dim message As String

Console.Write("Enter message: ")

message = Console.ReadLine

Console.WriteLine()

Console.WriteLine("Your Message: {0}", message)

Console.ReadLine()

EndSub

EndModule

When the above code is compiled and executed, it produces the following result (assume the user inputs Hello World) –

Enter message: Hello World

Your Message: Hello World

- **Differentiate between variable and constant?**

- **Ans.variable**, its value can be changed by the program at runtime. The accessibility or the scope of a variable refers to where the variable can be read from or written to, and its lifetime, or how long it stays in the computer memory.eg. `inti`;
- **Constant:** A Constant is something that will always remain the same throughout the entire lifetime of a program. A Constant variable cannot be modified after it is defined and it cannot be changed throughout the program. The Constant with a fixed value tells the compiler to prevent the programmer from modifying it. Whenever you try to change it, it will throw an error message. Constant variables are declared with the `const` keyword and can be used with primitive data types. Constants are set at compile time itself and assigned for value types only. e.g.
  - `public const double PI = 3.14159;`

- **Define various type conversions in Visual Basic**

The process of changing a value from one data type to another type is called **conversion**. Conversions are either **widening** or **narrowing**, depending on the data capacities of the types involved. They are also **implicit** or **explicit**, depending on the syntax in the source code.

### **Widening and Narrowing Conversions**

An important consideration with a type conversion is whether the result of the conversion is within the range of the destination data type.

A **widening conversion** changes a value to a data type that can allow for any possible value of the original data. **Widening conversions preserve the source value but can change its representation.** This occurs if you convert from an integral type to `Decimal`, or from `Char` to `String`. For example, converting a value of type `integer` to a value of type `Long` is a widening conversion because the `Long` type can accommodate every possible value of the `Integer` type. Widening conversions always succeed at run time and never incur data loss. You can always perform them implicitly, whether the [Option Strict Statement](#) sets the type checking switch to On or to Off.

### **Narrowing Conversions**

A **narrowing conversion** changes a value to a data type that might not be able to hold some of the possible values. For example, a fractional value is rounded when it is converted to an integral type, and a numeric type being converted to `Boolean` is reduced to either `True` or `False`.

The standard narrowing conversions include the following:

- The reverse directions of the widening conversions in the preceding table (except that every type widens to itself)
- Conversions in either direction between [Boolean](#) and any numeric type
- Conversions from any numeric type to any enumerated type (`Enum`)

- Conversions in either direction between [String](#) and any numeric type, Boolean, or [Date](#)
- Conversions from a data type or object type to a type derived from it

Narrowing conversions do not always succeed at run time, and can fail or incur data loss. An error occurs if the destination data type cannot receive the value being converted. For example, a numeric conversion can result in an overflow. The compiler does not allow you to perform narrowing conversions implicitly unless the [Option Strict Statement](#) sets the type checking switch to Off.

### Implicit and Explicit Conversions

An *implicit conversion* does not require any special syntax in the source code. In the following example, Visual Basic implicitly converts the value of k to a single-precision floating-point value before assigning it to q.

```
Dim k As Integer
Dim q As Double
' Integer widens to Double, so you can do this with Option Strict On.
k = 432
q = k
```

An *explicit conversion* uses a type conversion keyword. Visual Basic provides several such keywords, which coerce an expression in parentheses to the desired data type. These keywords act like functions, but the compiler generates the code inline, so execution is slightly faster than with a function call.

In the following extension of the preceding example, the CInt keyword converts the value of q back to an integer before assigning it to k.

```
' q had been assigned the value 432 from k.
q = Math.Sqrt(q)
k = CInt(q)
' k now has the value 21 (rounded square root of 432).
```

### • Explain Type Conversion Functions?

There are functions that we can use to convert from one data type to another. They include:

- **CBool**(expression): converts the expression to a Boolean data type.
- **CDate**(expression): converts the expression to a Date data type.
- **CDBl**(expression): converts the expression to a Double data type.
- **CByte**(expression): converts the expression to a byte data type.
- **CChar**(expression): converts the expression to a Char data type.
- **CLng**(expression): converts the expression to a Long data type.
- **CDec**(expression): converts the expression to a Decimal data type.
- **CInt**(expression): converts the expression to an Integer data type.
- **CObj**(expression): converts the expression to an Object data type.
- **CStr**(expression): converts the expression to a String data type.
- **CSByte**(expression): converts the expression to a Byte data type.
- **CShort**(expression): converts the expression to a Short data type.

[Module](#)[Module1](#)

[Sub](#)Main()

```

Dim dblData As Double
dblData = 5.78
Dim num As Integer

num = CInt(dblData)
Console.WriteLine(" Value of Double is: {0}", dblData)
Console.WriteLine("Double to Integer: {0}", num)
Console.ReadKey()

EndSub

EndModule

```

### Q . What is an operator?

**Ans.** An operator refers to a symbol that instructs the compiler to perform a specific logical or mathematical manipulation. The operator performs the operation on the provided operands. Microsoft VB.Net comes with various types of operators.

- Arithmetic Operators
- Comparison Operators
- Logical/Bitwise Operators
- Bit Shift Operators
- Assignment Operators
- Miscellaneous Operators

### Arithmetic Operators

You can use arithmetic operators to perform various mathematical operations in VB.NET. They include:

Symbol	Description
^	for raising an operand to the power of another operand
+	for adding two operands.
-	for subtracting the second operand from the first operand.
*	for multiplying both operands.
/	for dividing an operand against another. It returns a floating point result.
\	for dividing an operand against another. It returns an integer result.
MOD	known as the modulus operator. It returns the remainder after division.

### Comparison Operators

These operators are used for making comparisons between variables. They include the following:

Comparison Operators	Details
=	for checking whether the two operands have equal

	values or not. If yes, the condition will become true.
<>	for checking if the value of the left operand is greater than that of the right operand. then condition will become true.
>	for checking whether the value of the left operand is less than that of the right operand. If yes, the condition will become true.
<	for checking whether the value of the left operand is greater than or equal to that of the right operand. If yes, the condition will become true.
>=	for checking whether the two operands have equal values or not. If yes, the condition will become true.
<=	for checking whether the value of the left operand is less than or equal to that of the right operand. If yes, the condition will become true.

### Logical/Bitwise Operators

These operators help us in making logical decisions.

They include:

Logical/ Bite wise Operator	Descriptions
And	known as the logical/bitwise AND. Only true when both conditions are true.
Or	known as the logical/bitwise OR. True when any of the conditions is true.
Not	The logical/bitwise NOT. To reverse operand's logical state. If true, the condition becomes False and vice versa.
Xor	bitwise Logical Exclusive OR operator. Returns False if expressions are all True or False. Otherwise, it returns True.
AndAlso	It is also known as the logical AND operator. Only works with Boolean data by performing short-circuiting.
OrElse	It is also known as the logical OR operator. Only works with Boolean data by performing short-circuiting.
IsFalse	Determines whether expression evaluates to False.
IsTrue	Determines whether expression evaluates to True.

### Bit Shift Operators

Assume that the variable A holds 60 and variable B holds 13, then –

Operator	Description	Example
And	Bitwise AND Operator copies a bit to the result if it exists in both operands.	(A AND B) will give 12, which is

		0000 1100
Or	Binary OR Operator copies a bit if it exists in either operand.	(A Or B) will give 61, which is 0011 1101
Xor	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A Xor B) will give 49, which is 0011 0001
Not	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(Not A ) will give -61, which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operand's value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operand's value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

Try the following example to understand all the bitwise operators available in VB.Net  
 -

[Live Demo](#)

```

Module BitwiseOp
Sub Main()
Dim a As Integer = 60 ' 60 = 0011 1100
    Dim b As Integer = 13 ' 13 = 0000 1101
Dim c As Integer = 0
    c = a And b ' 12 = 0000 1100
    Console.WriteLine("Line 1 - Value of c is {0}", c)
    c = a Or b ' 61 = 0011 1101

    Console.WriteLine("Line 2 - Value of c is {0}", c)
    c = a Xor b ' 49 = 0011 0001

    Console.WriteLine("Line 3 - Value of c is {0}", c)
    c = Not a ' -61 = 1100 0011
  
```

```

Console.WriteLine("Line 4 - Value of c is {0}", c)
c = a << 2 ' 240 = 1111 0000

Console.WriteLine("Line 5 - Value of c is {0}", c)
c = a >> 2 '15=00001111

Console.WriteLine("Line 6 - Value of c is {0}", c)
Console.ReadLine()
EndSub
EndModule

```

When the above code is compiled and executed, it produces the following result –

```

Line 1 - Value of c is 12
Line 2 - Value of c is 61
Line 3 - Value of c is 49
Line 4 - Value of c is -61
Line 5 - Value of c is 240
Line 6 - Value of c is 15

```

### Assignment Operators

Assignment Operator	Details
=	<ul style="list-style-type: none"> <li>the simple assignment operator. It will assign values from the left side operands to the right side operands.</li> </ul>
+=	<ul style="list-style-type: none"> <li>known as the Add AND assignment operator. It will add the right operand to the left operand. Then the result will be assigned to the left operand.</li> </ul>
=	<ul style="list-style-type: none"> <li>known as the Subtract AND operator. It will subtract the right operand from left operand and assign the result to the left operand.</li> </ul>
*=	<ul style="list-style-type: none"> <li>: known as the Multiply AND operator. It will subtract the right operand from left operand and assign the result to the left operand.</li> </ul>

### Miscellaneous Operators

There are other operators supported by VB.NET. Let us discuss them:

Miscellaneous Operators	Details
GetType	This operator gives the Type of objects for a specified expression.
AddressOf	<ul style="list-style-type: none"> <li>Returns the address of a procedure</li> </ul>

### • Program on ARITHMATIC OPERATOR

Module Module1

```

Sub Main()
    Dim var_w As Integer = 11

```



```
Dim var_x As Integer = 5
Dim var_q As Integer = 2
Dim var_y As Integer
Dim var_z As Single
```

```
var_y = var_w + var_z
Console.WriteLine("Result of 11 + 5 is {0} ", var_y)
```

```
var_y = var_w - var_x
Console.WriteLine("Result of 11 - 5 is {0} ", var_y)
```

```
var_y = var_w * var_x
Console.WriteLine("Result of 11 * 5 is {0} ", var_y)
```

```
var_z = var_w / var_x
Console.WriteLine("Result of 11 / 5 is {0}", var_z)
```

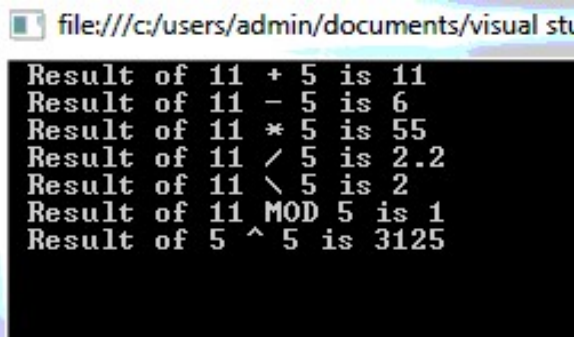
```
var_y = var_w \ var_x
Console.WriteLine("Result of 11 \ 5 is {0}", var_y)
```

```
var_y = var_w Mod var_x
Console.WriteLine("Result of 11 MOD 5 is {0}", var_y)
```

```
var_y = var_x ^ var_x
Console.WriteLine("Result of 5 ^ 5 is {0}", var_y)
Console.ReadLine()
```

End Sub

End Module



```
file:///c:/users/admin/documents/visual stu
Result of 11 + 5 is 11
Result of 11 - 5 is 6
Result of 11 * 5 is 55
Result of 11 / 5 is 2.2
Result of 11 \ 5 is 2
Result of 11 MOD 5 is 1
Result of 5 ^ 5 is 3125
```

- **PROGRAM ON COMPARATIVE OPERATORS**

Module Module1

```
Sub Main()
    Dim x As Integer = 11
    Dim y As Integer = 5
    If (x = y) Then
        Console.WriteLine("11=5 is True")
    End If
End Sub
```

```
Else
Console.WriteLine(" 11=5 is False")
End If
```


```
If (x < y) Then
Console.WriteLine(" 11<5 is True")
Else
Console.WriteLine(" 11<5 is False")
End If
```

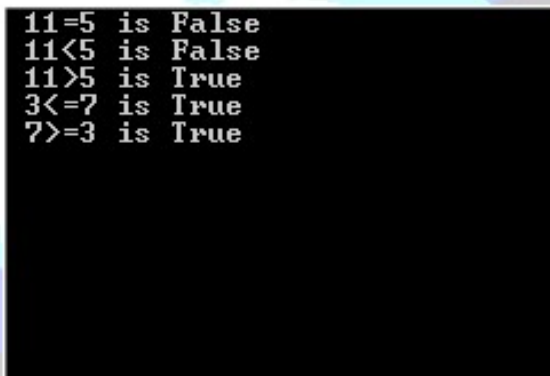
```
If (x > y) Then
Console.WriteLine(" 11>5 is True")
Else
Console.WriteLine(" 11>5 is False")
End If
```

```
x = 3
y = 7
If (x <= y) Then
Console.WriteLine(" 3<=7 is True")
End If
If (y >= x) Then
Console.WriteLine(" 7>=3 is True")
End If
Console.ReadLine()
```

```
End Sub
```

```
End Module
```

 c:\users\admin\documents\visual studio



```
11=5 is False
11<5 is False
11>5 is True
3<=7 is True
7>=3 is True
```

- **LOGICAL OPERATOR**

```
Module Module1
```

```
Sub Main()
```

```
Dim var_wAs Boolean = True
Dim var_xAs Boolean = True
```

```

    Dim var_yAs Integer = 5
    Dim var_zAs Integer = 20

    If (var_w And var_x) Then
Console.WriteLine("var_w And var_x - is true")
    End If
    If (var_wOrvar_x) Then
Console.WriteLine("var_wOrvar_x - is true")
    End If
    If (var_wXorvar_x) Then
Console.WriteLine("var_wXorvar_x - is true")
    End If
    If (var_yAndvar_z) Then
Console.WriteLine("var_yAndvar_z - is true")
    End If
    If (var_yOrvar_z) Then
Console.WriteLine("var_yOrvar_z - is true")
    End If
    'Only logical operators
    If (var_wAndAlsovar_x) Then
Console.WriteLine("var_wAndAlsovar_x - is true")
    End If
    If (var_wOrElsevar_x) Then
Console.WriteLine("var_wOrElsevar_x - is true")
    End If
var_w = False
var_x = True
    If (var_wAndvar_x) Then
Console.WriteLine("var_wAndvar_x - is true")
    Else
Console.WriteLine("var_wAndvar_x - is not true")
    End If
    If (Not (var_wAndvar_x)) Then
Console.WriteLine("var_wAndvar_x - is true")
    End If
Console.ReadLine()
End Sub
End Module

```

file:///C:/Users/admin/Documents/Visual Studio 2

```

var_w And var_x - is true
var_w Or var_x - is true
var_y And var_z - is true
var_y Or var_z - is true
var_w AndAlso var_x - is true
var_w OrElse var_x - is true
var_w And var_x - is not true
var_w And var_x - is true

```

- [Bit Shift Operators](#)

Module Module1

```
Sub Main()
```

```
    Dim w As Integer = 50
    Dim x As Integer = 11
    Dim y As Integer = 0
    y = w And x
    Console.WriteLine("y = w And x is {0}", y)
    y = w Or x
```

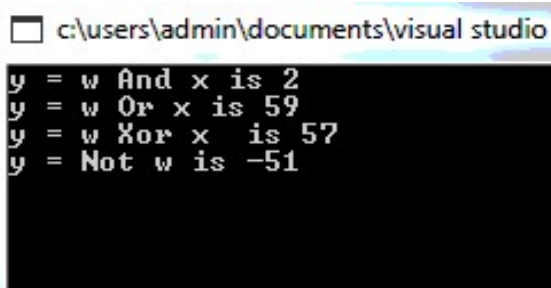
```
    Console.WriteLine("y = w Or x is {0}", y)
    y = w Xor x
```

```
    Console.WriteLine("y = w Xorx is {0}", y)
    y = Not w
```

```
    Console.WriteLine("y = Not w is {0}", y)
    Console.ReadLine()
```

```
End Sub
```

```
End Module
```



The screenshot shows a Visual Studio console window with the following output:

```
y = w And x is 2
y = w Or x is 59
y = w Xor x is 57
y = Not w is -51
```

- [Assignment Operators](#)

Module Module1

```
Sub Main()
```

```
    Dim x As Integer = 5
```

```
    Dim y As Integer
```

```
    y = x
    Console.WriteLine(" y = x gives y = {0}", y)
```

```
    y += x
    Console.WriteLine(" y += x gives y = {0}", y)
```

```
    y -= x
    Console.WriteLine(" y -= x gives y = {0}", y)
```

```
y *= x
Console.WriteLine(" y *= x gives y = {0}", y)
```

```
Console.ReadLine()
```

```
End Sub
End Module
```

file:///C:/Users/admin/Documents/Visual Studio

```
y = x gives y = 5
y += x gives y = 10
y -= x gives y = 5
y *= x gives y = 25
```

- **Explain Operator Precedence in VB.NET**

Operator precedence is used to determine the order in which different Operators in a complex expression are evaluated. There are distinct levels of precedence, and an Operator may belong to one of the levels. The Operators at a higher level of precedence are evaluated first. Operators of similar precedents are evaluated at either the left-to-right or the right-to-left level.

The Following table shows the operations, Operators and their precedence -

Operations	Operators	Precedence
Await		Highest
Exponential	^	
Unary identity and negation	+, -	
Multiplication and floating-point division	*, /	
Integer division	\	
Modulus arithmetic	Mod	
Addition and Subtraction	+, -	
Arithmetic bit shift	<<, >>	
All comparison Operators	=, <>, <, <=, >, >=, Is, IsNot, Like, TypeOf ...is	
Negation	Not	
Conjunction	And, AndAlso	
Inclusive disjunction	Or, Else	
Exclusive disjunction	Xor	Lowest

Example of **Operator Precedence in VB.NET**

```
Imports System
Module Operator_Precedence
Sub Main()
'Declare and Initialize p, q, r, s variables
Dim p As Integer = 30
Dim q As Integer = 15
```

```
Dim r As Integer = 10
Dim s As Integer = 5
Dim result As Integer
```

```
Console.WriteLine("Check Operator Precedence in VB.NET")
```

```
'Check Operator Precedence
```

```
result = (p + q) * r / s ' 45 * 10 / 5
```

```
Console.WriteLine("Output of (p + q) * r / s is : {0}", result)
```

```
result = (p + q) * (r / s) ' (45) * (10/5)
```

```
Console.WriteLine("Output of (p + q) * (r / s) is : {0}", result)
```

```
result = ((p + q) * r) / s ' (45 * 10) / 5
```

```
Console.WriteLine("Output of ((p + q) * r) / s is : {0}", result)
```

```
result = p + (q * r) / s ' 30 + (150/5)
```

```
Console.WriteLine("Output of p + (q * r) / s is : {0}", result)
```

```
result = ((p + q * r) / s) ' ((30 + 150) / 5)
```

```
Console.WriteLine("Output of ((p + q * r) / s) is : {0}", result)
```

```
Console.WriteLine(" Press any key to exit...")
```

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```

### Q. What do you understand by Control structure in VB.Net?

**Ans.** Control structures enable programmers to control the order of events in their programs.

The conditional Statements used in decision making and changing the control flow of the program execution

Visual Basic provides seven types of Looping Statements(repetition structures)  
: **While, DoWhile/Loop,**

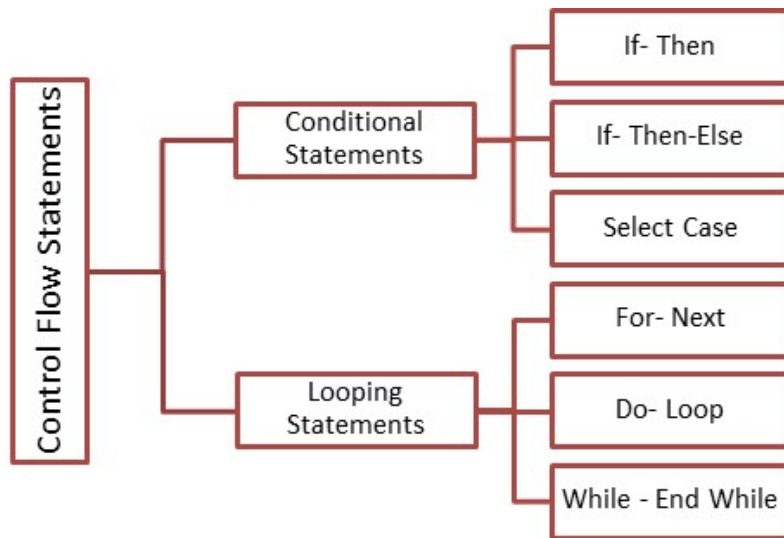
Do/LoopWhile, DoUntil/Loop, Do/LoopUntil, For/Next and ForEach/Next.

(ForEach/Next is covered in Chapter The control flow in VB.Net controls the flow of statements program. It controls the data used in the program. These statements are of two types.

The

words If, Then, Else, End, Select, Case, While, Do, Until, Loop, For, Next and Each are all Visual Basic keywords.

- Conditional Statements
- Looping Statements



**Conditional Statements:** In VB.Net conditional statements are used to check the conditions. It is mainly of three types.

- If- Then
- If - Then - Else
- Select Case

#### If- then

The if-then statement is used to check normal conditions. If the condition written in it is true, the statements of If- End If block are executed. And the control goes out when the condition is false.

#### Syntax:

If Condition Then

Statements

End If

Example:

If a>b Then

Msgbox ("A is greater")

End If

Module decisions

SubMain()

'local variable definition

Dim a As Integer = 10

' check the boolean condition using if statement

If(a <20)Then

' if condition is true then print the following

Console.WriteLine("a is less than 20")

End If

Console.WriteLine("value of a is : {0}", a)



```
Console.ReadLine()  
End Sub  
End Module
```

### If- then- else

This conditional statement consists of two blocks. An If block is executed when the condition is true and an Else block is executed when the condition is false.

#### Syntax:

```
If Condition Then  
[If block states]  
Else  
[Else Block Statements]  
End If
```

#### Example:

```
If a>b Then  
Msgbox ("A is greater")  
Else  
Msgbox ("B is greater")  
End If
```

### If...Else If...Else Statement

An If statement can be followed by an optional **Else if...Else** statement, which is very useful to test various conditions using single If...Else If statement.

```
If(boolean_expression 1)Then  
    ' Executes when the boolean expression 1 is true  
ElseIf( boolean_expression 2)Then  
    ' Executes when the boolean expression 2 is true  
ElseIf( boolean_expression 3)Then  
    ' Executes when the boolean expression 3 is true  
Else  
    ' executes when the none of the above condition is true  
End If
```

Example

```
Module decisions  
SubMain()  
    'local variable definition '  
    DimaAsInteger=100  
    ' check the boolean condition '  
    If(a =10)Then  
        ' if condition is true then print the following '  
        Console.WriteLine("Value of a is 10")  
    ElseIf (a = 20) Then  
        'ifelseif condition istrue'
```

```

Console.WriteLine("Value of a is 20") '
ElseIf(a =30)Then
'if else if condition is true
Console.WriteLine("Value of a is 30")
Else
'if none of the conditions istrue
Console.WriteLine("None of the values is matching")
EndIf
Console.WriteLine("Exact value of a is: {0}", a)
Console.ReadLine()
EndSub
EndModule

```

### Nested If Statements

It is always legal in VB.Net to nest If-Then-Else statements, which means you can use one If or ElseIf statement inside another If ElseIf statement(s).

Syntax

The syntax for a nested If statement is as follows –

```

If( boolean_expression 1)Then
'Executes when the boolean expression 1 is true
If(boolean_expression 2)Then
'Executes when the boolean expression 2 is true
End If
End If

```

You can nest ElseIf...Else in the similar way as you have nested If statement.

Example

```

Module decisions
SubMain()
'local variable definition
Dim a As Integer = 100
Dim b As Integer = 200
' check the boolean condition

If(a =100)Then
' if condition is true then check the following
If (b = 200) Then
'if condition istrue then print the following
Console.WriteLine("Value of a is 100 and b is 200")
EndIf
EndIf
Console.WriteLine("Exact value of a is : {0}", a)
Console.WriteLine("Exact value of b is : {0}", b)
Console.ReadLine()
EndSub
EndModule

```

When the above code is compiled and executed, it produces the following result –

Value of a is 100 and b is 200

Exact value of a is : 100  
Exact value of b is : 200

## Select case

The Select Case statement is used to check the value of an expression on different cases. In this, the expression written in select case is matched with value cases and if the correct match is found, the statement of that block is executed. If a case does not match, the case Else statement is executed.

Syntax:

```
Select Case textexpression
Case ExpressionList ..... N
['Statements..... N]]
[Case Else
[Statement Else]]
End select
```

### Example

```
Module decisions
SubMain()
'local variable definition
Dim grade As Char
grade = "B"
Select grade
Case "A"
Console.WriteLine("Excellent!")
Case "B", "C"
Console.WriteLine("Well done")
Case "D"
Console.WriteLine("You passed")
Case "F"
Console.WriteLine("Better try again")
Case Else
Console.WriteLine("Invalid grade")
End Select
Console.WriteLine("Your grade is {0}", grade)
Console.ReadLine()
End Sub

End Module
```

## Looping Statements in VB.Net

Looping statements are used to repeat a code or task multiple times. These statements repeatedly repeat the task according to a condition. There are mainly three types of looping statements in VB.Net.

- For-Next
- While- End While

- Do- Loop

## For-next

A **For Next loop** is used to repeatedly execute a sequence of code or a block of code until a given condition is satisfied.

Syntax

```
For variable_name As [ DataType ] = start To end [ Step step ]  
[ Statements to be executed ]  
Next
```

Example 1. Write a simple program to print the number from 1 to 10 using the For Next loop.

### **Number.vb**

```
Imports System  
Module Module1  
    Sub Main()  
        ' It is a simple print statement, and 'vbCrLf' is used to jump in the next line.  
        Console.WriteLine(" The number starts from 1 to 10 " & vbCrLf)  
        ' declare and initialize variable i  
        For i As Integer = 1 To 10 Step 1  
            ' if the condition is true, the following statement will be executed  
            Console.WriteLine(" Number is {0} ", i)  
            ' after completion of each iteration, next will update the variable counter  
        Next  
        Console.WriteLine(" Press any key to exit... ")  
        Console.ReadKey()  
    End Sub  
End Module
```

**Output:**

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
The number starts from 1 to 10
Number is 1
Number is 2
Number is 3
Number is 4
Number is 5
Number is 6
Number is 7
Number is 8
Number is 9
Number is 10
Press any key to exit...
```

Example 2. Write the following program to skip the number is 2.

#### Number.vb

```
Imports System
Module Module1
    Sub Main()
        ' declaration of variable i
        Dim i As Integer
        Console.WriteLine(" Skip one number at the completion of each iteration in between 1 to 10 "
& vbCrLf)
        ' initialize i to 1 and declare Step to 2 for skipping a number
        For i = 1 To 10 Step 2
            ' if condition is true, it skips one number
            Console.WriteLine(" Number is {0} ", i)
            ' after completion of each iteration, next will update the variable counter to step 2
        Next
        Console.WriteLine(" Press any key to exit... ")
        Console.ReadKey()
    End Sub
End Module
```

#### Output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Skip one number at the completion of each iteration in between 1 to 10
Number is 1
Number is 3
Number is 5
Number is 7
Number is 9
Press any key to exit...
_
```

- **Nested For Next Loop in VB.NET**

In VB.NET, when we write one For loop inside the body of another For Next loop, it is called **Nested For Next** loop.

Syntax:

```
For variable_name As [Data Type] = start To end [ Step step ]  
    For variable_name As [Data Type] = start To end [ Step step ]  
        [ inner loop statements ]  
    Next  
    [ Outer loop statements ]  
Next
```

Example of Nested For Next loop in VB.NET.

**Nested\_loop.vb**

```
Imports System  
Module Module1  
    Sub Main()  
        Dim i, j As Integer  
        For i = 1 To 3  
            'Outer loop  
            Console.WriteLine(" Outer loop, i = {0}", i)  
            'Console.WriteLine(vbCrLf)  
  
            'Inner loop  
            For j = 1 To 4  
                Console.WriteLine(" Inner loop, j = {0}", j)  
            Next  
            Console.WriteLine()  
        Next  
        Console.WriteLine(" Press any key to exit...")  
        Console.ReadKey()  
    End Sub  
End Module
```

**Output:**

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe

Outer loop, i = 1
Inner loop, j = 1
Inner loop, j = 2
Inner loop, j = 3
Inner loop, j = 4

Outer loop, i = 2
Inner loop, j = 1
Inner loop, j = 2
Inner loop, j = 3
Inner loop, j = 4

Outer loop, i = 3
Inner loop, j = 1
Inner loop, j = 2
Inner loop, j = 3
Inner loop, j = 4

Press any key to exit...
```

### VB.NET For Each Loop

In the VB.NET, **For Each loop** is used to iterate block of statements in an array or collection objects. when iteration through each element in the array or collection is complete, the control transferred to the next statement to end the loop.

Syntax:

```
For Each var_name As [ DataType ] In Collection_Object
[ Statements to be executed]
Next
```

**For Each loop** is used to read each element from the collection object or an array. The **Data Type** represents the type of the variable, and **var\_name** is the name of the variable to access elements from the **array** or **collection object** so that it can be used in the body of For Each loop.

Examples of For Each Loop

Write a simple program to understand the uses of For Each Next loop in VB.NET.

#### **For\_Each\_loop.vb**

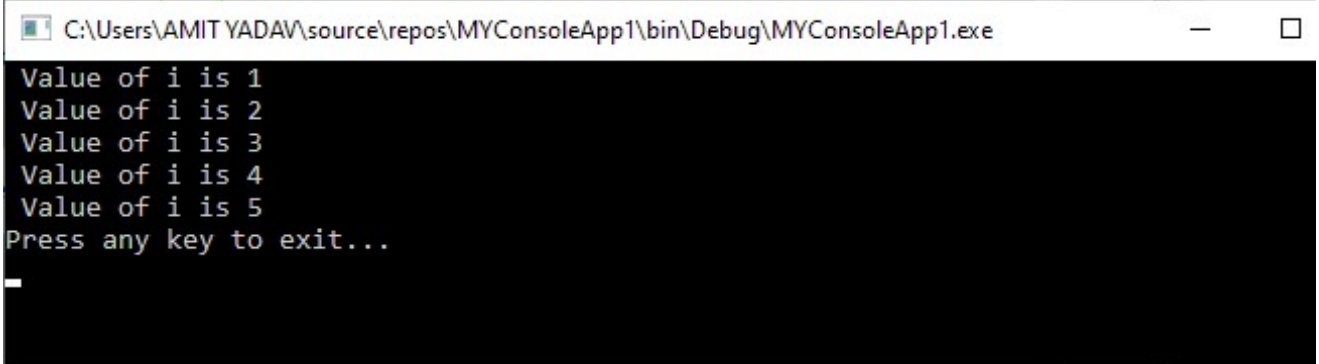
```
Imports System
Module Module1
    Sub Main()
        'declare and initialize an array as integer
        Dim An_array() As Integer = {1, 2, 3, 4, 5}
        Dim i As Integer 'Declare i as Integer

        For Each i In An_array
            Console.WriteLine(" Value of i is {0}", i)
        Next
```



```
        Console.WriteLine("Press any key to exit...")
        Console.ReadLine()
    End Sub
End Module
```

### Output:



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Value of i is 1
Value of i is 2
Value of i is 3
Value of i is 4
Value of i is 5
Press any key to exit...
_
```

Example 2: Write a simple program to print fruit names using For Each loop in VB.NET.

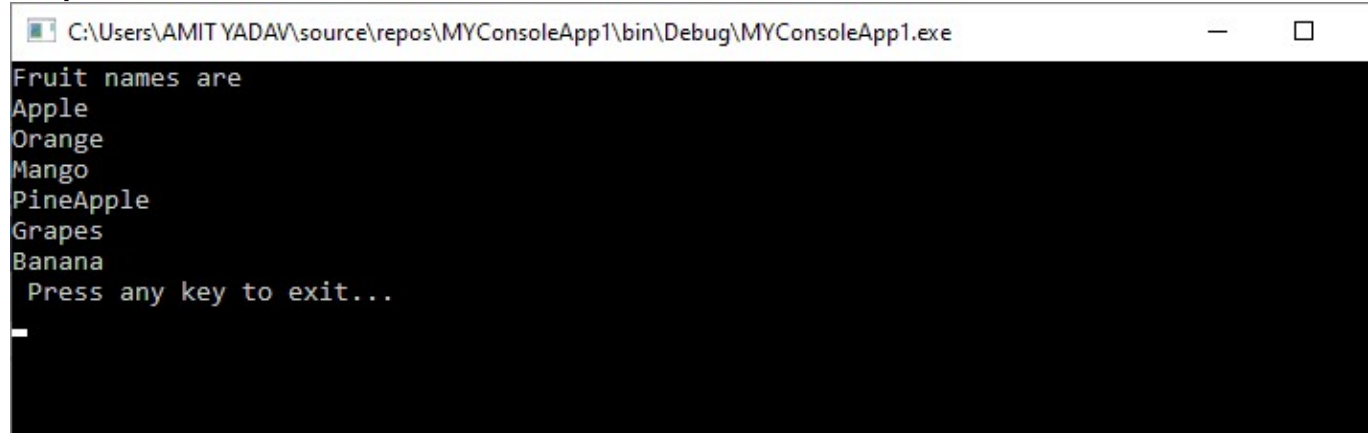
### For\_each.vb

```
Imports System
Module Module1
    Sub Main()
        'Define a String array
        Dim str() As String
        'Initialize all element of str() array
        str = {"Apple", "Orange", "Mango", "PineApple", "Grapes", "Banana"}

        Console.WriteLine("Fruit names are")

        'Declare variable name as fruit
        For Each fruit As String In str
            Console.WriteLine(fruit)
        Next
        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```

## Output:



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Fruit names are
Apple
Orange
Mango
PineApple
Grapes
Banana
Press any key to exit...
```

### VB.NET While End Loop

The **While End loop** is used to execute blocks of code or statements in a program, as long as the given **condition** is true. It is useful when the number of executions of a block is not known. It is also known as an **entry-controlled loop** statement.

#### **Syntax:**

```
While [condition]
[ Statement to be executed ]
End While
```

Here, **condition** represents any **Boolean condition**, and if the logical condition is true, the **single or block of statements** define inside the body of the while loop is executed.

Example: Write a simple program to print the number from 1 to 10 using while End loop in VB.NET.

#### **while\_number.vb**

```
Imports System
Module Module1
    Sub Main()
        'declare x as an integer variable
        Dim x As Integer
        x = 1
        ' Use While End condition
        While x <= 10
            'If the condition is true, the statement will be executed.
            Console.WriteLine(" Number {0}", x)
            x = x + 1 ' Statement that change the value of the condition
        End While
        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```

#### **Output:**

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Number 1
Number 2
Number 3
Number 4
Number 5
Number 6
Number 7
Number 8
Number 9
Number 10
Press any key to exit...
```

Example 2: Write a program to print the sum of digits of any number using while End loop in VB.NET.

#### **Total\_Sum.vb**

```
Public Class Total_Sum ' Create a Class Total_sum
    Shared Sub Main()
        'Declare an Integer variable
        Dim n, remainder, sum As Integer
        sum = 0

        Console.WriteLine(" Enter the number :")
        n = Console.ReadLine() ' Accept a number from the user

        ' Use While loop and write given below condition
        While (n > 0)
            remainder = n Mod 10
            sum += remainder
            n = n / 10
        End While
        Console.WriteLine(" Sum of digit is :{0}", sum)
        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
End Class
```

**Output:**

```
C:\Users\AMIT YADAV\source\repos\MyConsoleApp2\bin\Debug\MyConsoleApp2.exe
Enter the number :
125
Sum of digit is :8
Press any key to exit...
```

### Do While Loop

In VB.NET, Do While loop is used to execute blocks of statements in the program, as long as the condition remains true. It is similar to the [While End Loop](#), but there is slight difference between them. The **while** loop **initially checks** the defined condition, if the condition becomes true, the while loop's statement is executed. Whereas in the **Do** loop, is opposite of the while loop, it means that it executes the Do statements, and then it checks the condition.

Syntax:

```
Do
[ Statements to be executed]
Loop While Boolean_expression
// or
Do
[Statement to be executed]
Loop Until Boolean_expression
```

In the above syntax, the **Do** keyword followed a block of statements, and **While** keyword checks **Boolean\_expression** after the execution of the first Do statement.

Example 1. Write a simple program to print a number from 1 to 10 using the Do While loop in VB.NET.

#### **Do\_loop.vb**

```
Imports System
Module Module1
    Sub Main()
        ' Initializatio and Declaration of variable i
        Dim i As Integer = 1
        Do
            ' Executes the following Statement
            Console.WriteLine(" Value of variable I is : {0}", i)
            i = i + 1 'Increment the variable i by 1
            Loop While i <= 10 ' Define the While Condition

            Console.WriteLine(" Press any key to exit...")
            Console.ReadKey()
        End Sub
    End Module
```

Now compile and execute the above program by clicking on the Start button, it shows the following output:

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Value of variable i is : 1
Value of variable i is : 2
Value of variable i is : 3
Value of variable i is : 4
Value of variable i is : 5
Value of variable i is : 6
Value of variable i is : 7
Value of variable i is : 8
Value of variable i is : 9
Press any key to exit...
```

### Use of Until in Do Until Loop statement

In the VB.NET loop, there is a **Do Until loop** statement, which is similar to the **Do While loop**. The Do Statement executes as long as Until condition becomes true.

Example: Write a program to understand the uses of Do Until Loop in VB.NET.

#### **Do\_loop.vb**

```
Imports System
Module Module1
    Sub Main()
        ' Initialization and Declaration of variable i
        Dim i As Integer = 1
        Do
            ' Executes the following Statement
            Console.WriteLine(" Value of variable i is : {0}", i)
            i = i + 1 'Increment variable i by 1
        Loop Until i = 10 ' Define the Until Condition

        Console.WriteLine(" Press any key to exit...")
        Console.ReadKey()
    End Sub
End Module
```

**Output:**

```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Value of variable i is : 1
Value of variable i is : 2
Value of variable i is : 3
Value of variable i is : 4
Value of variable i is : 5
Value of variable i is : 6
Value of variable i is : 7
Value of variable i is : 8
Value of variable i is : 9
Press any key to exit...
```

## Arrays

- **What do you understand by Array?**

**Ans.** An **array** is a linear data structure that is a collection of data elements of the same type stored on a **contiguous memory** location. Each data item is called an element of the array. It is a fixed size of sequentially arranged elements in computer memory with the first element being at **index 0** and the last element at index **n - 1**, where **n** represents the total number of elements in the array.

Declaring Array: Array, like Simple Variable, is given the value of the last index of the array in parentheses (parentheses) after the name of the array

**Syntax:** Dim <array name> (subscript) As <Data Type>

**Example:** Dim employee\_name (20) As String

Dim salary (20) As Integer

Now, let us see the example to declare an array.

- 'Store only Integer values
- Dim num As Integer() or Dim num() As Integer
- 'Store only String values
- Dim name As String() or Dim name(5) As String
- 'Store only Double values
- Dim marks As Double()

### Types of Array

There are two types of Array in Visual Basic.

- One Dimensional Array
- Multi-Dimensional Array

**1.One Dimensional Array:** An array containing only one index or subscript is called a dimensional array. Multiple values can be stored in it. In this, its index is used to store or access a value in an array.

Example: Dim marks (10) As Integer

The marks array can store 0 to 9 indexes or 10 values in the marks array.

- Initializing one dimensional Array: values can be inserted in both the design and run time in a dimensional array.

Array index or Curly brackets {} are used to store the value in design time.

**Syntax:** Dim array\_name () As DataType = {element1, element2,...}

Example: Dim name () As String = {"Raj", "Shyam", "Ashu"}

or

Dim Name (2) As String

Name (0) = "Raj"

Name (1) = "Shyam"

Name (2) = "Ashu"

**2. Multi-Dimensional Array:** Arrays of many dimensions can be created in VB.Net. Arrays that contain more than one subscripts are called multi-dimensional arrays. Dim, private or public statements are also used to declare multi-Dimensional arrays. When declare it more than one subscripts are given in it. It can store values equal to multiple of subscripts such that n (2,3) array can store 6 values.

- Syntax: Dim <array name> (script1, script2...) As <Data Type>

- Example: Dim matrix (3, 3) as Integer

Dim T (2, 3, 4, 2) As Double

- Initializing Multi-Dimensional Array: Use Curly brackets to initialize Multi-Dimensional Array.

Dim n (3, 3) As Integer = {{1, 2, 3}, {4, 5, 6}, {7, 8, 9}}

- Create a program to add the elements of an array in VB.NET programming language.

**num\_Array.vb**

Imports System

Module Module1

Sub Main()

Dim i As Integer, Sum As Integer = 0

Dim marks() As Integer = {58, 68, 95, 50, 23, 89}

Console.WriteLine(" Marks in 6 Subjects")

For i = 0 To marks.Length - 1

Console.WriteLine(" Marks {0}", marks(i))

Sum = Sum + marks(i)

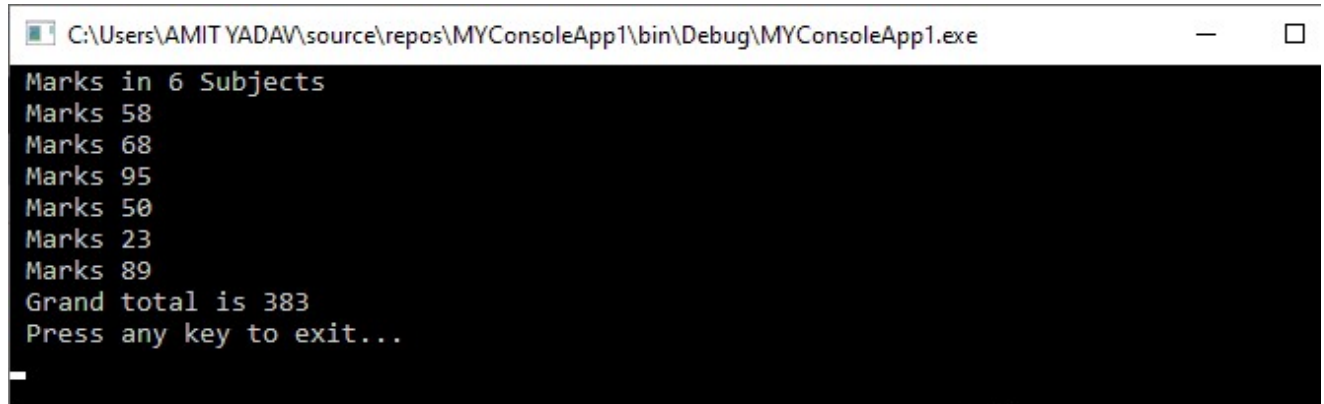
Next

Console.WriteLine(" Grand total is {0}", Sum)



```
    Console.WriteLine(" Press any key to exit...")
    Console.ReadKey()
End Sub

End Module
```



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Marks in 6 Subjects
Marks 58
Marks 68
Marks 95
Marks 50
Marks 23
Marks 89
Grand total is 383
Press any key to exit...
```

- **Dynamic Array**

A Dynamic array is used when we do not know how many items or elements to be inserted in an array. To resolve this problem, we use the dynamic array. It allows us to insert or store the number of elements at runtime in sequentially manner. A Dynamic Array can be resized according to the program's requirements at run time using the "**ReDim**" statement.

### Initial Declaration of Array

#### Syntax:

- Dim array\_name() As Integer

### Runtime Declaration of the VB.NET Dynamic array (Resizing)

#### Syntax:

- ReDim [Preserve] array\_name(subscripts)

The **ReDim** statement is used to declare a dynamic array. To resize an array, we have used a **Preserve** keyword that preserve the existing item in the array.

The **array\_name** represents the name of the array to be re-dimensioned.

A **subscript** represents the new dimension of the array.

### Initialization of Dynamic Array

- Dim myArr() As String
- ReDim myArr(3)
- myArr(0) = "One"
- myArr(1) = "Two"

- myArr(2) = "Three"
- myArr(3) = "Four"

To initialize a Dynamic Array, we have used create a string array named **myArr()** that uses the Dim statement in which we do not know the array's actual size. The **ReDim** statement is used to resize the existing array by defining the subscript (3). If we want to store one more element in index 4 while preserving three elements in an array, use the following statements.

- ReDim Preserve myArr(4)
- myArr(4) = "Five"

the above array has four elements.

Also, if we want to store multiple data types in an array, we have to use a **Variant** data type.

- Dim myArr() As Variant
- ReDim myArr(3)
- myArr(0) = 10
- myArr(0) = "String"
- myArr(0) = false
- myArr(0) = 4.6

Let's create a program to understand the dynamic array.

### **Dynamic\_Arr.vb**

Imports System

Module Module1

Sub Main()

Dim Days(20) As String

' Resize an Array using the ReDim Statement

ReDim Days(6)

Days(0) = "Sunday"

Days(1) = "Monday"

Days(2) = "Tuesday"

Days(3) = "Wednesday"

Days(4) = "Thursday"

Days(5) = "Friday"

Days(6) = "Saturday"

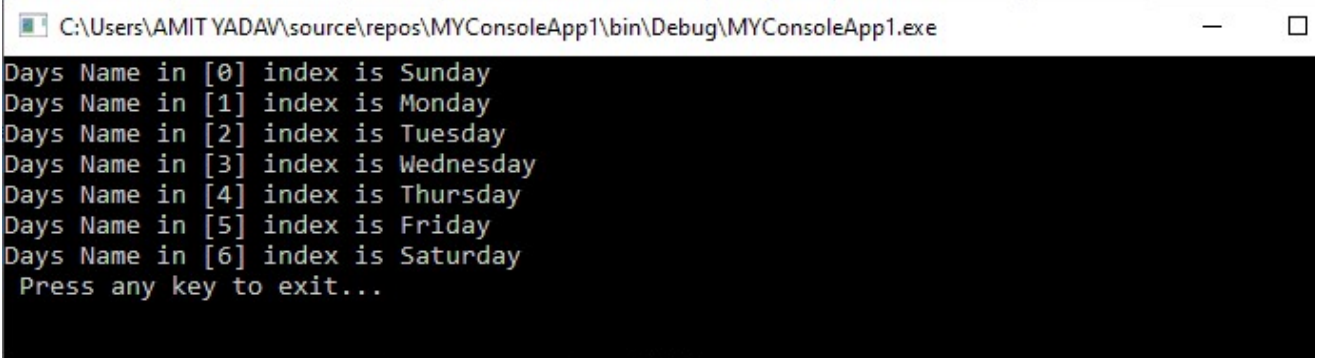
For i As Integer = 0 To Days.Length - 1

Console.WriteLine("Days Name in [{0}] index is {1}", i, Days(i))

Next

```
    Console.WriteLine(" Press any key to exit...")
    Console.ReadKey()
End Sub
End Module
```

### Output:



```
C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Days Name in [0] index is Sunday
Days Name in [1] index is Monday
Days Name in [2] index is Tuesday
Days Name in [3] index is Wednesday
Days Name in [4] index is Thursday
Days Name in [5] index is Friday
Days Name in [6] index is Saturday
Press any key to exit...
```

## Variable's Scope and Lifetime of a Variable.

The difference between lifetime and scope is quite simple. Lifetime - Refers to how long or when the variable is valid (i.e. how long will it retain its value for).

Scope - Refers to where the variable can be accessed. The difference between lifetime and scope is quite simple. **Lifetime is how long the variable retains its value for. The scope refers to where the variable can be used.**

### Variable Lifetime

The lifetime of the local variable "imylocal" extends from the moment ProcedureOne is entered to the moment ProcedureOne has finished. However, the local variable "imylocal" still exists when ProcedureTwo is called, it is just out of scope.

```
Public Sub ProcedureOne()
    Dim imylocal As Integer
    imylocal = 5
    Call ProcedureTwo()
    imylocal = 10
End Sub
```

```
Public Sub ProcedureTwo()
    'do something
End Sub
```

### Variable Scope

Every variable has a scope associated with it. This scope refers to the area of code where the variables will be recognised. Variables (and constants) have a defined scope within the program. The scope for a variable can either be: [Procedure Level](#) - Also known as a local

variable. [Module Level](#) - A private module level variable is visible to only the module it is declared in. [Global Level](#) - A public module level variable is visible to all modules in the project.

e.g.

```
Private Sub Command1_Click()
```

```
Dim i as Integer
```

```
Dim Sum as Integer
```

```
For i=0 to 100 Step 2
```

```
Sum = Sum +i
```

```
Next
```

```
MsgBox " The Sum is "& Sum
```

```
End Sub
```

A variable whose value is available to all procedures within the same Form or Module are called Form-wide or Module-wide and can be accessed from within all procedures in a component. In some situations, the entire application must access a certain variable. Such variable must be declared as Public.

**Lifetime of a Variable:** It is the period for which they retain their value. Variables declared as Public exist for the lifetime of the application. Local variables, declared within procedures with the Dim or Private statement, live as long as the procedure.

we can force a local variable to preserve its value between procedure calls with the Static keyword. The advantage of using static variables is that they help you minimize the number of total variables in the application.

Variables declared in a Form outside any procedure take effect when the Form is loaded and cease to exist when the Form is unloaded. If the Form is loaded again, its variables are initialized, as if it's being loaded for the first time.

### **Q . How to create enumeration in VB.Net?**

**Ans** In VB.NET, **Enum** is a keyword known as Enumeration.

**Enumeration** is a user-defined data type used to define a related set of constants as a list using the keyword **enum** statement. It can be used with module, structure, class, and procedure. For example, month names can be grouped using Enumeration.

#### **Syntax:**

- Enum enumeration name [ As Data type ]
- ' enumeration data\_list or Enum member list
- End Enum

In the above syntax, the **enumeration name** is a valid identifier name, and the data type can be mentioned if necessary, the enumeration **data\_list** or **Enum member list** represents the set of the related constant member of the Enumeration.

### Declaration of Enumerated Data

Following is the declaration of enumerated data using the Enum keyword in [VB.NET Programming language](#):

- Enum Fruits
- Banana
- Mango
- Orange
- Pineapple
- Apple
- End Enum

Here, **Fruits** is the name of enumerated data that contains a list of fruits called **enumeration** lists.

When we are creating an Enumerator constant data list, by default, it assigns a value 0 to the first index of the enumerator. Each successive item in the enumerator list is increased by 1. For example, in the above enumerator list, Banana value is 0, Mango value is 1, Orange value is 2, and so on.

Moreover, if we want to assign a new value to the default value of the enumerator items, set a new value to the list's first index, and then enumerator's values can automatically provide a new incremental value for successor objects in the enumerator list.

Override a default index value by setting a new value for the first element of the enumerator list.

- Enum Fruits
- Banana = 5
- Mango
- Orange
- Pineapple
- Apple
- End Enum

Here, we have assigned 5 as the new index value of the first item in an enumerator list. So, the initial value of the list's Banana=5, Mango=6, orange=7, and so on. Furthermore, we can also provide new value in the middle of the enumerator list, and then it follows the sequence set by the previous item.

**Example 1:** Write a program to understand the uses of enumeration data in [VB.NET](#).

## Enum\_Data.vb

Imports System

Module Enum\_Data

Enum Number

' List of enumerated data

One

Two

Three

Four

Five

Six

End Enum

Sub Main()

Console.WriteLine(" Without the index number")

Console.WriteLine(" Number {0}", Number.One) 'call with enumeration and items name

Console.WriteLine(" Number {0}", Number.Two)

Console.WriteLine(" Number {0}", Number.Three)

Console.WriteLine(" Number {0}", Number.Four)

Console.WriteLine(" Number {0}", Number.Five)

Console.WriteLine(" Number {0}", Number.Six)

Console.WriteLine(" Press any key to exit...")

Console.ReadKey()

End Sub

End Module

## Output:

In the above example, an enumerator **"Number"** that gets each value of the enumerated data such as **"Number.One"**, etc. to print the list.

Imports System

Module module1

Enum Number

' List of enumerated data

One

Two

Three = 5

Four

Five

Six

End Enum

Sub Main()

Console.WriteLine(" Without the index number")

Console.WriteLine(" Number {0}", Number.One) 'call with enumeration and items name

Console.WriteLine(" Number {0}", Number.Two)

Console.WriteLine(" Number {0}", Number.Three)

Console.WriteLine(" Number {0}", Number.Four)

Console.WriteLine(" Number {0}", Number.Five)

Console.WriteLine(" Number {0}", Number.Six)

```

    Dim a, b, c As Integer
    a = CInt(Number.One)
    b = CInt(Number.Three)
    c = CInt(Number.Six)
    Console.WriteLine("Number one is at position {0}", a)
    Console.WriteLine("Number Three is at position {0}", b)
    Console.WriteLine("Number Six is at position {0}", c)
    Console.WriteLine(" Press any key to exit...")
    Console.ReadKey()
End Sub
End Module

```

## Enum\_month.vb

Imports System

Module Enum\_month

Enum Monthname 'Enumeration name

January = 1

February

march

April

May

June

July

August

September = 10 'Set value to 10

October

November

December

End Enum

Sub Main()

Dim x As Integer = CInt(Monthname.June)

Dim y As Integer = CInt(Monthname.August)

Dim p As Integer = CInt(Monthname.September)

Dim q As Integer = CInt(Monthname.October)

Dim r As Integer = CInt(Monthname.December)

Console.WriteLine(" Month name is {0}", Monthname.January)

Console.WriteLine(" June Month name is in {0}", x & " Position")

Console.WriteLine(" August Month name is in {0}", y & vbCrLf & "Position")

Console.WriteLine(" After Setting a new value at the middle")

Console.WriteLine(" September Month name is {0}", p & " Position")

Console.WriteLine(" October Month name is {0}", q & " Position")

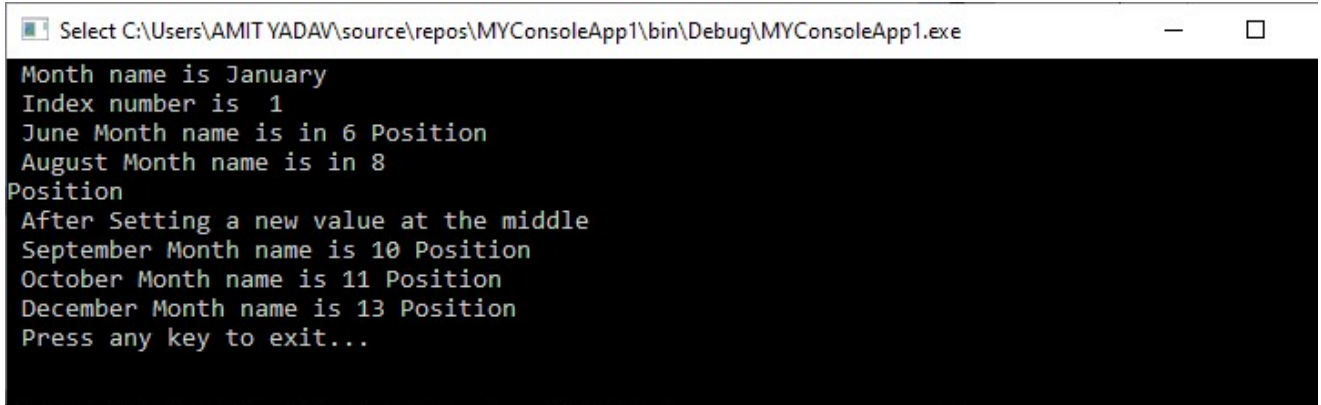
Console.WriteLine(" December Month name is {0}", r & " Position")

Console.WriteLine(" Press any key to exit...")

```
Console.ReadKey()
```

```
End Sub
```

```
End Module
```



```
Select C:\Users\AMIT YADAV\source\repos\MYConsoleApp1\bin\Debug\MYConsoleApp1.exe
Month name is January
Index number is 1
June Month name is in 6 Position
August Month name is in 8
Position
After Setting a new value at the middle
September Month name is 10 Position
October Month name is 11 Position
December Month name is 13 Position
Press any key to exit...
```

### Program -

```
Module Module1
```

```
Enum Position
```

```
    Above
```

```
    Below
```

```
End Enum
```

```
sub main()
```

```
dim value = Position.Above
```

```
    If value = Position.Below Then
```

```
        Console.WriteLine("POSITION IS BELOW")
```

```
    else
```

```
        Console.WriteLine("POSITION IS Above")
```

```
    End If
```

```
End Sub
```

```
End Module
```

### Program -

```
Module Module1
```

```
Enum Code As Byte
```



```

    Small = 0
    Medium = 1
    Large = 2
End Enum

Sub Main()
    ' The value is represented in a byte.
    Dim value As Code = Code.Medium
    Console.WriteLine(value)
End Sub

End Module

```

## Structure

Structure is a user-defined datatype in computer language which allows us to combine data of different types together. Structure helps to construct a complex data type which is more meaningful. It is somewhat similar to an Array, but an array holds data of similar type only. But structure on the other hand, can store data of any type, which is practical more useful.

**For example:** If I have to write a program to store Student information, which will have Student's name, age, branch, permanent address, father's name etc, which included string values, integer values etc. It will require something which can hold data of different types together.

In structure, data is stored in form of **records**.

```

Public Structure User
    Public name As String
    Public location As String
    Public age As Integer
End Structure

```

If you observe the above example, we defined a structure called “**User**” with the required fields and we can add required methods and properties based on our requirements.

### Visual Basic Structure Initialization

In visual basic, structures can be instantiated with or without **New** keyword. Following is the example of assigning a values to the variables of structure.

```
Dim u As User = New User()  
u.name = "Suresh Dasari"  
u.location = "Hyderabad"  
u.age = 32
```

To make use of [fields](#), [methods](#) and events of structure, it's mandatory to instantiate the structure with **New** keyword in a visual basic programming language.

## Procedure

A procedure is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has two types of procedures –

- Functions
- Sub procedures or Subs

Functions return a value, whereas Subs do not return a value.

## Functions

- In VB.NET, the function is a separate group of codes that are used to perform a specific task when the defined function is called in a program. After the execution of a function, control transfer to the **main()** method for further execution.
- It returns a value. In [vb.net](#), we can create more than one function in a program to perform various functionalities. The function is also useful to code reusability by reducing the duplicity of the code. For example, if we need to use the same functionality at multiple places in a program, we can simply create a function and call it whenever required.

## Defining a Function

The syntax to define a function is:

- [Access\_specifier ] Function Function\_Name [ (ParameterList) ] As Return\_Type
- [ Block of Statement ]
- Return return\_val
- End Function

Where,

- **Access\_Specifier:** It defines the access level of the function such as public, private, or friend, Protected function to access the method.
- **Function\_Name:** The function\_name indicate the name of the function that should be unique.
- **ParameterList:** It defines the list of the parameters to send or retrieve data from a method.

- **Return\_Type:** It defines the data type of the variable that returns by the function.

```
Public Function GetData( ByVal username As String, ByVal userId As Integer) As String
    ' Statement to be executed
End Function
```

**Example: Write a program to reverse a number and check whether the given number is palindrome or not.**

**Palindrome.vb**

Imports System

Module Palindrome

' Define a reverse() function

Function reverse(ByVal num As Integer) As Integer

' Define the local variable

Dim remain As Integer

Dim rev As Integer = 0

While (num > 0)

remain = num Mod 10

rev = rev \* 10 + remain

num = num / 10

End While

Return rev

End Function

Sub Main()

' Define the local variable as integer

Dim n, num2 As Integer

Console.WriteLine(" Enter a number")

n = Console.ReadLine() 'Accept the number

num2 = reverse(n) ' call a function

Console.WriteLine(" Reverse of {0} is {1}", n, num2)

If (n = reverse(n)) Then

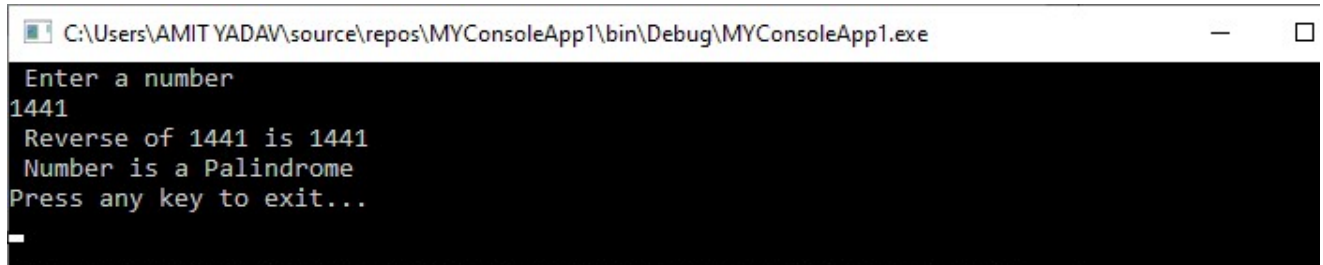
Console.WriteLine(" Number is a Palindrome")

Else

Console.WriteLine(" Number is not a Palindrome")

End If

```
    Console.WriteLine("Press any key to exit...")
    Console.ReadKey()
End Sub
End Module
```



## Sub

A Sub procedure is a separate set of codes that are used in VB.NET programming to execute a specific task, and it does not return any values. The Sub procedure is enclosed by the Sub and End Sub statement. The Sub procedure is similar to the function procedure for executing a specific task except that it does not return any value, while the function procedure returns a value.

### Defining the Sub procedure

Following is the syntax of the Sub procedure:

- [Access\_Specifier] Sub Sub\_name [ (parameterList) ]
- [ Block of Statement to be executed ]
- End Sub

Where,

- **Access\_Specifier:** It defines the access level of the procedure such as public, private or friend, Protected, etc. and information about the overloading, overriding, shadowing to access the method.
- **Sub\_name:** The Sub\_name indicates the name of the Sub that should be unique.
- **ParameterList:** It defines the list of the parameters to send or retrieve data from a method.
- Public Function GetData1( ByRef username As String, ByRef userId As Integer)
- ' Statement to be executed
- End Sub

In the VB.NET programming language, we can pass parameters in two different ways:

- Passing parameter by Value
- Passing parameter by Reference

## Passing parameter by Value

In the VB.NET, passing parameter by value is the default mechanism to pass a value in the Sub method. When the method is called, it simply copies the actual value of an argument into the formal method of Sub procedure for creating a new storage location for each parameter. Therefore, the changes made to the main function's actual parameter that do not affect the Sub procedure's formal argument.

### Syntax:

- Sub Sub\_method( ByVal parameter\_name As datatype )
- [ Statement to be executed]
- End Sub

In the above syntax, the **ByVal** is used to declare parameters in a Sub procedure. Let's create a program to understand the concept of passing parameter by value.

### Passing\_value.vb

Imports System

Module Module1

Sub Main()

' declaration of local variable

Dim num1, num2 As Integer

Console.WriteLine(" Enter the First number")

num1 = Console.ReadLine()

Console.WriteLine(" Enter the Second number")

num2 = Console.ReadLine()

Console.WriteLine(" Before swapping the value of 'num1' is {0}", num1)

Console.WriteLine(" Before swapping the value of 'num2' is {0}", num2)

Console.WriteLine()

'Call a function to pass the parameter **for** swapping the numbers.

swap\_value(num1, num2)

Console.WriteLine(" After swapping the value of 'num1' is {0}", num1)

Console.WriteLine(" After swapping the value of 'num2' is {0}", num2)

Console.WriteLine(" Press any key to exit...")

Console.ReadKey()

End Sub

' Create a swap\_value() method

Sub swap\_value(ByVal a As Integer, ByVal b As Integer)

```

' Declare a temp variable
Dim temp As Integer
temp = a ' save the value of a to temp
b = a ' put the value of b into a
b = temp ' put the value of temp into b
End Sub
End Module

```

## Passing parameter by Reference

A Reference parameter is a reference of a variable in the memory location. The reference parameter is used to pass a reference of a variable with **ByRef** in the Sub procedure. When we pass a reference parameter, **it does not create a new storage location for the sub method's formal parameter. Furthermore, the reference parameters represent the same memory location as the actual parameters supplied to the method.** So, when **we changed the value of the formal parameter, the actual parameter value is automatically changed in the memory.** The syntax for the passing parameter by Reference:

- Sub Sub\_method( ByRef parameter\_name, ByRef Parameter\_name2 )
- [ Statement to be executed]
- End Sub

In the above syntax, the **ByRef** keyword is used to pass the Sub procedure's reference parameters.

**Q. Create a program to swap the values of two variables using the ByRef keyword. *Passing\_ByRef.vb***

```

Imports System
Module Module1
Sub Main()
' declaration of local variable
Dim num1, num2 As Integer
Console.WriteLine(" Enter the First number")
num1 = Console.ReadLine()
Console.WriteLine(" Enter the Second number")
num2 = Console.ReadLine()
Console.WriteLine(" Before swapping the value of 'num1' is {0}", num1)
Console.WriteLine(" Before swapping the value of 'num2' is {0}", num2)
Console.WriteLine()

'Call a sub to pass the parameter for swapping the numbers.
swap_Ref(num1, num2)
Console.WriteLine(" After swapping the value of 'num1' is {0}", num1)

```

```
Console.WriteLine(" After swapping the value of 'num2' is {0}", num2)
Console.WriteLine(" Press any key to exit...")
Console.ReadKey()
End Sub
```

```
' Create a swap_Ref() method
Sub swap_Ref(ByRef a As Integer, ByRef b As Integer)
' Declare a temp variable
Dim temp As Integer
temp = a ' save the value of a to temp
a = b ' put the value of b into a
b = temp ' put the value of temp into b
End Sub
End Module
```

