Web Programming with VB.NET

1. Introduction to VB.NET

VB.NET (Visual Basic .NET) is an object-oriented programming language developed by Microsoft. It is part of the **.NET platform** and can be used to create Windows applications, web applications, services, and other software.

VB.NET is commonly used with **ASP.NET** for web programming.

Basic VB.NET Program

```
Module Program
    Sub Main()
        Console.WriteLine("Hello, World!")
    End Sub
End Module
```

Explanation

- Module Program — defines a module.
- Sub Main() — starting point of a console application.
- Console.WriteLine() — displays output.
- End Sub — ends the procedure.
- End Module — ends the module.

---

2. Data Types

A **data type** specifies the kind of data that a variable can store. VB.NET provides several built-in data types.

Common VB.NET Data Types

Data Type	Description	Example
Byte	0 to 255	Dim age As Byte = 20
Short	Small integer	Dim x As Short = 1000
Integer	32-bit integer	Dim marks As Integer = 85
Long	64-bit integer	Dim population As Long = 1000000

Data Type	Description	Example
Single	Single-precision decimal	Dim price As Single = 25.5
Double	Double-precision decimal	Dim pi As Double = 3.14159
Decimal	High-precision decimal	Dim salary As Decimal = 50000.5D
Char	Single character	Dim grade As Char = "A"c
String	Collection of characters	Dim name As String = "Rahul"
Boolean	True or False	Dim result As Boolean = True
Date	Date and time	Dim d As Date = Today
Object	Can hold any type	Dim value As Object = 10

Syntax

Dim variableName As DataType

or

Dim variableName As DataType = value

Example

Dim studentName As String = "Amit"

Dim age As Integer = 21

Dim percentage As Double = 87.5

Dim passed As Boolean = True

Dim birthDate As Date = #08/11/2005#

---

3. Variables

A **variable** is a named memory location used to store data. The value of a variable can change during program execution.

Syntax

Dim variableName As DataType

Example

Dim x As Integer

x = 10

```
x = 20
Console.WriteLine(x)
```

Output:

```
20
```

Variable Declaration and Initialization

```
Dim name As String = "John"
Dim age As Integer = 25
Dim salary As Decimal = 45000D
```

Multiple Variables

```
Dim x As Integer = 10, y As Integer = 20
```

Constants

A constant is a value that cannot be changed during program execution.

Syntax

```
Const constantName As DataType = value
```

Example

```
Const PI As Double = 3.14159
Console.WriteLine(PI)
```

---

4. Expressions

An **expression** is a combination of variables, constants, operators, and values that produces a result.

Syntax

```
operand operator operand
```

Examples

```
Dim a As Integer = 10
Dim b As Integer = 20
Dim sum As Integer = a + b
Dim difference As Integer = b - a
Dim product As Integer = a * b
```

Dim division As Double = b / a

Here:

a + b is an expression.

Types of Expressions

Arithmetic Expression

Dim result As Integer = 10 + 20 * 2

Relational Expression

Dim result As Boolean = (10 > 5)

Logical Expression

Dim result As Boolean = (10 > 5) And (20 > 10)

String Expression

Dim fullName As String = "John" & " Smith"

---

5. Operators

Operators are symbols or keywords used to perform operations on values and variables.

5.1 Arithmetic Operators

Operator	Meaning	Example
+	Addition	a + b
-	Subtraction	a - b
*	Multiplication	a * b
/	Division	a / b
\	Integer division	a \ b
Mod	Remainder	a Mod b
^	Power	a ^ b

Example

Dim a As Integer = 10

Dim b As Integer = 3

```
Console.WriteLine(a + b)     '13
Console.WriteLine(a - b)     '7
Console.WriteLine(a * b)     '30
Console.WriteLine(a / b)     '3.333...
Console.WriteLine(a \ b)     '3
Console.WriteLine(a Mod b)   '1
Console.WriteLine(a ^ b)     '1000
```

5.2 Relational/Comparison Operators

These operators compare two values and return True or False.

Operator	Meaning
=	Equal
<>	Not equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal

Example

```
Dim a As Integer = 10
Dim b As Integer = 20


Console.WriteLine(a < b)
Console.WriteLine(a = b)
Console.WriteLine(a <> b)
```

Output:

```
True
False
```

True

5.3 Logical Operators

Operator	Description
And	True if both conditions are true
Or	True if at least one condition is true
Not	Reverses the Boolean result
AndAlso	Short-circuit AND
OrElse	Short-circuit OR
Xor	True when exactly one condition is true

Example

```
Dim age As Integer = 25
Dim hasID As Boolean = True
If age >= 18 AndAlso hasID Then
    Console.WriteLine("Allowed")
End If
```

5.4 Assignment Operators

VB.NET supports assignment and compound assignment operators.

```
Dim x As Integer = 10

x += 5
x -= 2
x *= 3
x /= 2
```

After these operations, x is modified successively.

5.5 String Concatenation Operator

The & operator is used to join strings.

Dim firstName As String = "John"

Dim lastName As String = "Smith"

Dim fullName As String = firstName & " " & lastName

Console.WriteLine(fullName)

Output:

John Smith

---

6. Flow Control

Flow control determines the order in which program statements are executed.

The major flow-control structures are:

1. Conditional execution
2. Selection
3. Looping
4. Jump statements

1. Conditional Statements

Conditional statements execute different blocks of code depending on whether a condition is true or false.

VB.NET provides:

- If...Then
- If...Then...Else
- If...Then...ElseIf...Else
- Nested If
- Select Case

1.1 If...Then

Syntax

If condition Then

```
    statements
End If
```

Example

```
Dim marks As Integer = 75
If marks >= 40 Then
    Console.WriteLine("Pass")
End If
```

Output:

```
Pass
```

1.2 If...Then...Else

Syntax

```
If condition Then
    statements
Else
    statements
End If
```

Example

```
Dim marks As Integer = 35

If marks >= 40 Then
    Console.WriteLine("Pass")
Else
    Console.WriteLine("Fail")
End If
```

Output:

```
Fail
```

1.3 If...ElseIf...Else

Used when multiple conditions need to be checked.

Syntax

```
If condition1 Then
    statements
ElseIf condition2 Then
    statements
ElseIf condition3 Then
    statements
Else
    statements
End If
```

Example

```
Dim marks As Integer = 82
If marks >= 90 Then
    Console.WriteLine("Grade A+")
ElseIf marks >= 80 Then
    Console.WriteLine("Grade A")
ElseIf marks >= 70 Then
    Console.WriteLine("Grade B")
ElseIf marks >= 60 Then
    Console.WriteLine("Grade C")
Else
    Console.WriteLine("Fail")
End If
```

Output:

```
Grade A
```

1.4 Nested If

An If statement inside another If statement is called a nested If.

Example

```
Dim age As Integer = 20
Dim hasID As Boolean = True

If age >= 18 Then
    If hasID Then
        Console.WriteLine("Entry allowed")
    Else
        Console.WriteLine("ID required")
    End If
Else
    Console.WriteLine("Entry not allowed")
End If
```

1.5 Select Case

Select Case is useful when one expression must be compared with multiple possible values.

Syntax

```
Select Case expression
    Case value1
        statements
    Case value2
        statements
    Case Else
        statements
End Select
```

Example

```
Dim day As Integer = 3

Select Case day
    Case 1
        Console.WriteLine("Monday")
    Case 2
        Console.WriteLine("Tuesday")
    Case 3
        Console.WriteLine("Wednesday")
    Case 4
        Console.WriteLine("Thursday")
    Case 5
        Console.WriteLine("Friday")
    Case Else
        Console.WriteLine("Invalid day")
End Select
```

Output:

Wednesday

Range in Select Case

```
Dim marks As Integer = 75

Select Case marks
    Case 90 To 100
        Console.WriteLine("A+")
    Case 80 To 89
        Console.WriteLine("A")
    Case 70 To 79
        Console.WriteLine("B")
```

```
    Case 40 To 69
        Console.WriteLine("C")
    Case Else
        Console.WriteLine("Fail")
End Select
```

---

2. Looping Structures

A **loop** repeatedly executes a block of statements while a condition is satisfied or for a specified number of times.

Main loops in VB.NET:

1. For...Next
2. For Each...Next
3. While...End While
4. Do While...Loop
5. Do Until...Loop

2.1 For...Next Loop

Used when the number of iterations is known.

Syntax

```
For counter = start To end [Step increment]
    statements
Next
```

Example

```
For i As Integer = 1 To 5
    Console.WriteLine(i)
Next
```

Output:

```
1
2
```

3

4

5

Step

```
For i As Integer = 2 To 10 Step 2
    Console.WriteLine(i)
Next
```

Output:

2

4

6

8

10

Reverse Loop

```
For i As Integer = 10 To 1 Step -1
    Console.WriteLine(i)
Next
```

2.2. For Each Loop

For Each is used to iterate through elements of a collection or array.

Syntax

```
For Each variable In collection
    statements
Next
```

Example

```
Dim numbers() As Integer = {10, 20, 30, 40, 50}

For Each number As Integer In numbers
```

```
    Console.WriteLine(number)
Next
```

Output:

10

20

30

40

50

2..3. While Loop

The While loop executes as long as its condition is True.

Syntax

```
While condition
    statements
End While
```

Example

```
Dim i As Integer = 1

While i <= 5
    Console.WriteLine(i)
    i += 1
End While
```

Output:

1

2

3

4

5

Important: The condition is checked before each iteration.

2.4. Do While Loop

A Do While loop repeats while the condition is true.

Syntax

```
Do While condition
    statements
Loop
```

Example

```
Dim i As Integer = 1

Do While i <= 5
    Console.WriteLine(i)
    i += 1
Loop
```

2.5. Do Until Loop

A Do Until loop continues until its condition becomes true.

Syntax

```
Do Until condition
    statements
Loop
```

Example

```
Dim i As Integer = 1

Do Until i > 5
    Console.WriteLine(i)
    i += 1
```

```
Loop
```

Output:

```
1
2
3
4
5
```

---

3. Exit and Continue Statements

These statements control loop execution.

Exit For

```
For i As Integer = 1 To 10
    If i = 5 Then
        Exit For
    End If

    Console.WriteLine(i)
Next
```

Output:

```
1
2
3
4
```

Continue For

```
For i As Integer = 1 To 5
    If i = 3 Then
        Continue For
    End If
```

```
    Console.WriteLine(i)
Next
```

Output:

1

2

4

5

Arrays

An **array** is a collection of elements of the same data type stored under one variable name.

Arrays use **zero-based indexing**, meaning the first element has index 0.

One-Dimensional Array

Syntax

```
Dim arrayName(size) As DataType
```

Example

```
Dim marks(4) As Integer


marks(0) = 80
marks(1) = 75
marks(2) = 90
marks(3) = 65
marks(4) = 85
```

This array contains five elements, indexed from 0 through 4.

Array Initialization

```
Dim marks() As Integer = {80, 75, 90, 65, 85}
```

Accessing Array Elements

```
Console.WriteLine(marks(0))
```

```
Console.WriteLine(marks(2))
```

Output:

80

90

Using For Loop

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

Multidimensional Arrays

A multidimensional array contains data in multiple dimensions.

Syntax

```
Dim arrayName(rows, columns) As DataType
```

Example

```
Dim matrix(1, 2) As Integer

matrix(0, 0) = 10
matrix(0, 1) = 20
matrix(0, 2) = 30

matrix(1, 0) = 40
matrix(1, 1) = 50
matrix(1, 2) = 60
```

It represents:

10 20 30

40 50 60

Nested Loop

```
For i As Integer = 0 To 1
```

```
    For j As Integer = 0 To 2
        Console.Write(matrix(i, j) & " ")
    Next
    Console.WriteLine()
Next
```

Object-Oriented Programming (OOP)

Object-Oriented Programming is a programming approach based on objects and classes.

The major OOP concepts are:

1. Class
2. Object
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism

VB.NET strongly supports object-oriented programming.

Class

A **class** is a blueprint or template used to create objects.

A class can contain:

- Variables/fields
- Properties
- Methods
- Constructors
- Events

Syntax

```
Class ClassName
```

```
    ' Members
End Class
```

Example

```
Class Student
    Public Name As String
    Public Age As Integer

    Public Sub Display()
        Console.WriteLine("Name: " & Name)
        Console.WriteLine("Age: " & Age)
    End Sub
End Class
```

Here, Student is a class.

Object

An **object** is an instance of a class.

If Student is the class, an individual student created from it is an object.

Syntax

```
Dim objectName As New ClassName()
```

Example

```
Dim s1 As New Student()

s1.Name = "Rahul"
s1.Age = 21

s1.Display()
```

Output:

```
Name: Rahul
```

Age: 21

Multiple objects can be created from one class:

Dim s1 As New Student()

Dim s2 As New Student()

s1.Name = "Rahul"

s2.Name = "Priya"

Properties

A **property** provides controlled access to data in a class.

Properties are commonly used to implement **encapsulation**.

Syntax

Public Property PropertyName As DataType

Example

Class Student

    Public Property Name As String

    Public Property Marks As Integer

End Class

Creating an object:

Dim s As New Student()

s.Name = "Amit"

s.Marks = 85

Console.WriteLine(s.Name)

Console.WriteLine(s.Marks)

Property with Get and Set

A property can explicitly define Get and Set.

```
Class Student
    Private _marks As Integer
    Public Property Marks As Integer
        Get
            Return _marks
        End Get
        Set(value As Integer)
            If value >= 0 AndAlso value <= 100 Then
                _marks = value
            End If
        End Set
    End Property
End Class
```

Usage:

```
Dim s As New Student()
s.Marks = 85

Console.WriteLine(s.Marks)
```

The Set block controls how the value is assigned, while Get returns the value.

Methods

A **method** is a procedure defined inside a class that performs a particular task.

There are two common types:

- Sub — performs an operation but does not return a value.
- Function — performs an operation and returns a value.

Sub Method

Syntax

```
Public Sub MethodName(parameters)
```

```
    statements
End Sub
```

Example

```
Class Calculator

    Public Sub Add(ByVal a As Integer, ByVal b As Integer)
        Console.WriteLine(a + b)
    End Sub

End Class
```

Usage:

```
Dim c As New Calculator()
c.Add(10, 20)
```

Output:

```
30
```

Function Method

Syntax

```
Public Function MethodName(parameters) As ReturnType
    statements
    Return value
End Function
```

Example

```
Class Calculator
    Public Function Add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return a + b
    End Function
End Class
```

Usage:

```
Dim c As New Calculator()


Dim result As Integer = c.Add(10, 20)
Console.WriteLine(result)
```

Output:

```
30
```

Constructors

A **constructor** is a special method that is automatically called when an object is created.

In VB.NET, the constructor is defined using Sub New().

Syntax

```
Public Sub New()
    statements
End Sub
```

Example

```
Class Student
    Public Property Name As String
    Public Sub New()
        Name = "Unknown"
    End Sub
End Class
```

Usage:

```
Dim s As New Student()
Console.WriteLine(s.Name)
```

Output:

```
Unknown
```

Parameterized Constructor

```
Class Student


    Public Property Name As String
    Public Property Age As Integer


    Public Sub New(ByVal studentName As String, ByVal studentAge As Integer)
        Name = studentName
        Age = studentAge
    End Sub


End Class
```

Usage:

```
Dim s As New Student("Rahul", 21)


Console.WriteLine(s.Name)
Console.WriteLine(s.Age)
```

Encapsulation

Encapsulation means combining data and methods into one unit, usually a class, and controlling access to the internal data.

Private fields and public properties are commonly used for encapsulation.

Example

```
Class BankAccount
    Private balance As Decimal
    Public Property AccountBalance As Decimal
        Get
            Return balance
```

```
        End Get

        Private Set(value As Decimal)
            balance = value
        End Set
    End Property

    Public Sub Deposit(amount As Decimal)
        If amount > 0 Then
            balance += amount
        End If
    End Sub

End Class
```

Usage:

```
Dim account As New BankAccount()

account.Deposit(5000)
Console.WriteLine(account.AccountBalance)
```

The internal balance field cannot be directly accessed from outside the class.

Abstraction

Abstraction means hiding unnecessary implementation details and exposing only the required functionality.

For example, a user can call:

```
car.Start()
```

without knowing the internal implementation of the engine.

Abstract Class Example

```
MustInherit Class Animal

    Public MustOverride Sub MakeSound()

End Class
```

Derived class:

```
Class Dog
    Inherits Animal
    Public Overrides Sub MakeSound()
        Console.WriteLine("Bark")
    End Sub
End Class
```

Usage:

```
Dim d As New Dog()
d.MakeSound()
```

Output:

```
Bark
```

Inheritance

Inheritance allows one class to acquire the properties and methods of another class.

- Base class — parent class
- Derived class — child class

Syntax

```
Class Child
    Inherits Parent
End Class
```

Example

```
Class Person
    Public Property Name As String
    Public Sub DisplayName()
        Console.WriteLine(Name)
    End Sub
End Class
```

Derived class:

```
Class Student
    Inherits Person
    Public Property Marks As Integer
End Class
```

Usage:

```
Dim s As New Student()
s.Name = "Rahul"
s.Marks = 85
s.DisplayName()
Console.WriteLine(s.Marks)
```

The Student class inherits Name and DisplayName() from Person.

Polymorphism

Polymorphism means "many forms." It allows the same method or interface to behave differently depending on the object.

Two important forms are:

1. Method overriding
2. Method overloading

Method Overriding

```
Class Animal
    Public Overridable Sub Sound()
```

```
        Console.WriteLine("Animal sound")
    End Sub
End Class
```

Derived class:

```
Class Dog
    Inherits Animal
    Public Overrides Sub Sound()
        Console.WriteLine("Bark")
    End Sub
End Class
```

Usage:

```
Dim a As Animal = New Dog()
a.Sound()
```

Output:

Bark

Method Overloading

Multiple methods can have the same name but different parameters.

```
Class Calculator


    Public Function Add(a As Integer, b As Integer) As Integer
        Return a + b
    End Function
    Public Function Add(a As Integer, b As Integer, c As Integer) As Integer
        Return a + b + c
    End Function
End Class
```

Usage:

```
Dim c As New Calculator()
Console.WriteLine(c.Add(10, 20))
Console.WriteLine(c.Add(10, 20, 30))
```

Output:

30

60

Scope

Scope defines the part of a program in which a variable, method, or other member can be accessed.

Local Scope

A variable declared inside a procedure is available only within that procedure.

```
Sub Calculate()
    Dim x As Integer = 10
    Console.WriteLine(x)
End Sub
```

x cannot normally be accessed outside Calculate().

Class/Instance Scope

A variable declared inside a class but outside a method can be accessed by the class's methods.

```
Class Student
    Private name As String
    Public Sub SetName()
        name = "Rahul"
    End Sub
End Class
```

Module Scope

A variable declared at module level can be accessible to procedures within that module, depending on its access modifier.

```
Module Program
    Private count As Integer = 10
    Sub Display()
        Console.WriteLine(count)
    End Sub
End Module
```

Access Modifiers

Access modifiers control the visibility of classes and members.

Modifier	**Meaning**
Public	Accessible from anywhere
Private	Accessible only within the declaring type
Protected	Accessible within the class and derived classes
Friend	Accessible within the same assembly
Protected Friend	Protected OR Friend access
Private Protected	Protected within the same assembly

Example

```
Class Student
    Public Name As String
    Private marks As Integer
    Protected age As Integer
End Class
```

Events

An **event** is a notification that something has happened.

Events are especially important in **event-driven programming**, such as ASP.NET and Windows applications.

Examples include:

- Button click
- Page load
- Text changed
- Mouse movement
- Key press

A method that responds to an event is called an **event handler**.

Event Handler Example

In an ASP.NET Web Forms application:

```
Protected Sub Button1_Click(
    sender As Object,
    e As EventArgs
) Handles Button1.Click
    Label1.Text = "Button clicked!"
End Sub
```

When the user clicks Button1, the Button1_Click event handler executes.

Explanation

- Button1_Click — event handler method.
- sender — object that generated the event.
- e — event data.
- Handles Button1.Click — associates the method with the button's Click event.

Declaring Custom Events

VB.NET allows programmers to create their own events.

Syntax

Public Event EventName()

The event can be raised using:

RaiseEvent EventName()

Example

```
Class Student
    Public Event StudentPassed()
    Public Sub CheckResult(marks As Integer)
        If marks >= 40 Then
            RaiseEvent StudentPassed()
        End If
    End Sub
End Class
```

Handling the event:

```
Dim s As New Student()


AddHandler s.StudentPassed,
    Sub()
        Console.WriteLine("Student passed!")
    End Sub
s.CheckResult(75)
```

Output:

Student passed!

Web Programming with VB.NET

VB.NET can be used for web programming through **ASP.NET**, particularly ASP.NET Web Forms in traditional .NET Framework applications.

A web application generally consists of:

- Web pages
- Server-side code
- Controls
- Events
- Data access
- Business logic
- HTML/CSS/JavaScript

Simple ASP.NET Web Forms Example

ASPX page:

```
<asp:TextBox ID="txtName" runat="server"></asp:TextBox>

<asp:Button ID="btnHello"    runat="server"    Text="Say Hello" />


<asp:Label ID="lblMessage"    runat="server">

</asp:Label>
```

VB.NET code-behind:

```
Protected Sub btnHello_Click(    sender As Object,    e As EventArgs) Handles
btnHello.Click

   lblMessage.Text = "Hello, " & txtName.Text

End Sub
```

If the user enters:

Rahul

and clicks the button, the page displays:

Hello, Rahul

Complete Example Combining Concepts

The following example combines **class, properties, method, object, conditional statement, and event handling**.

Student Class

```
Public Class Student
```

```
    Public Property Name As String
    Public Property Marks As Integer
    Public Function GetResult() As String
        If Marks >= 40 Then
            Return "Pass"
        Else
            Return "Fail"
        End If
    End Function
End Class
```

Using the Class

```
Dim student1 As New Student()
student1.Name = "Rahul"
student1.Marks = 85
Console.WriteLine("Name: " & student1.Name)
Console.WriteLine("Marks: " & student1.Marks)
Console.WriteLine("Result: " & student1.GetResult())
```

Output:

```
Name: Rahul
Marks: 85
Result: Pass
```

ASP.NET Event Example

```
Protected Sub btnResult_Click( sender As Object,   e As EventArgs) Handles
btnResult.Click
    Dim student1 As New Student()
    student1.Name = txtName.Text
    student1.Marks = Convert.ToInt32(txtMarks.Text)
    lblResult.Text =    student1.Name & " - " & student1.GetResult()
End Sub
```