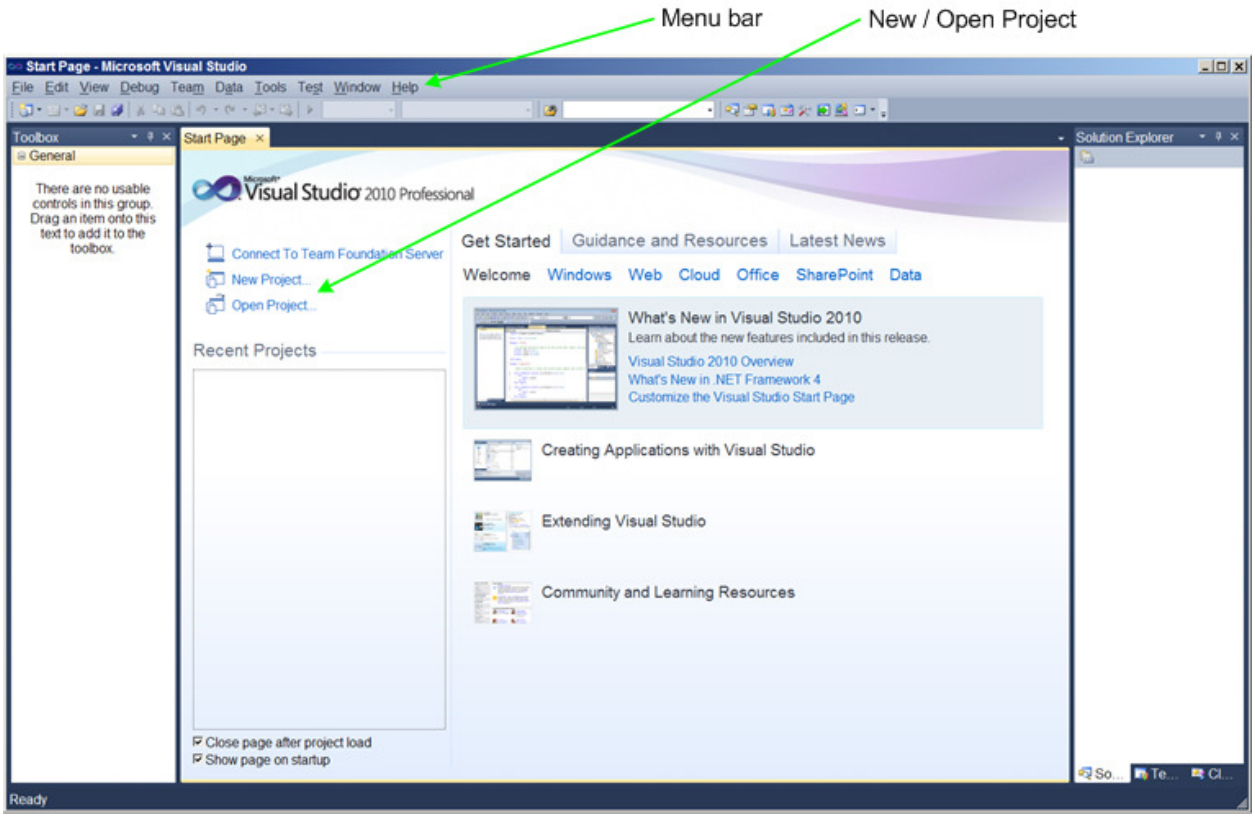
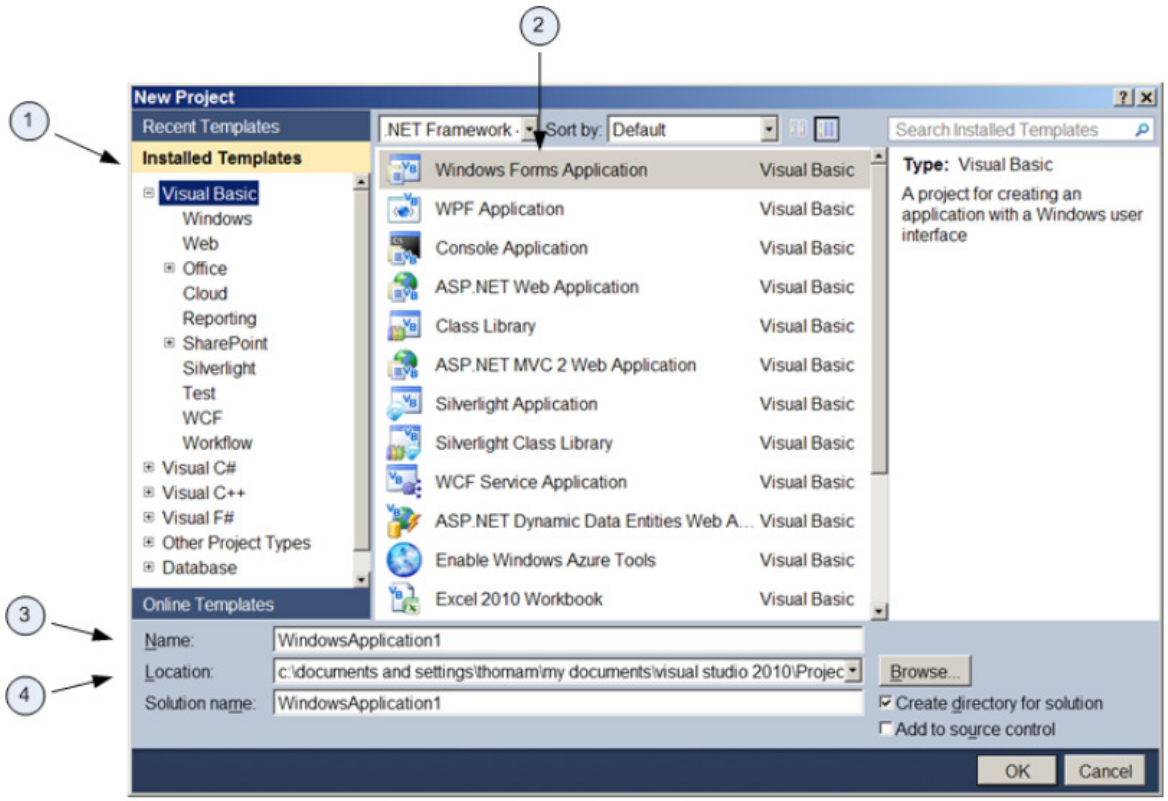
2.1 Integrated Development Environment (IDE)

- **Visual Studio** is an **Integrated Development Environment (IDE)** developed by Microsoft to develop GUI (Graphical User Interface), console, Web applications, web apps, mobile apps, cloud, and web services, etc. With the help of this IDE, you can create managed code as well as native code.
- It is not a language-specific IDE as you can use this to write code in C#, C++, VB (Visual Basic), Python, JavaScript, and many more languages. It provides support for **36** different programming languages. It is available for Windows as well as for macOS.
- **Evolution of Visual Studio:** The **first version** of VS (Visual Studio) was released in **1997**, named as **Visual Studio 97** having version number **5.0**. The latest version of Visual Studio is **15.0** which were released on **March 7, 2017**. It is also termed as **Visual Studio 2017**. The supported *.Net Framework Versions* in latest Visual Studio is **3.5 to 4.7**. Java was supported in old versions of Visual Studio but in the latest version doesn't provide any support for Java language.
- When we start VS 2010, we get startup screen as shown in fig.



Examine this screen capture for a few moments. We are not yet in the Visual Basic environment; therefore this is the **Visual Studio Start Page** (note the Start Page tab). The Menu Bar, as well as the green arrows pointing to the **New Project and Open Project** choices, is visible. You can use one of these to figure out which type of.Net project to work on.

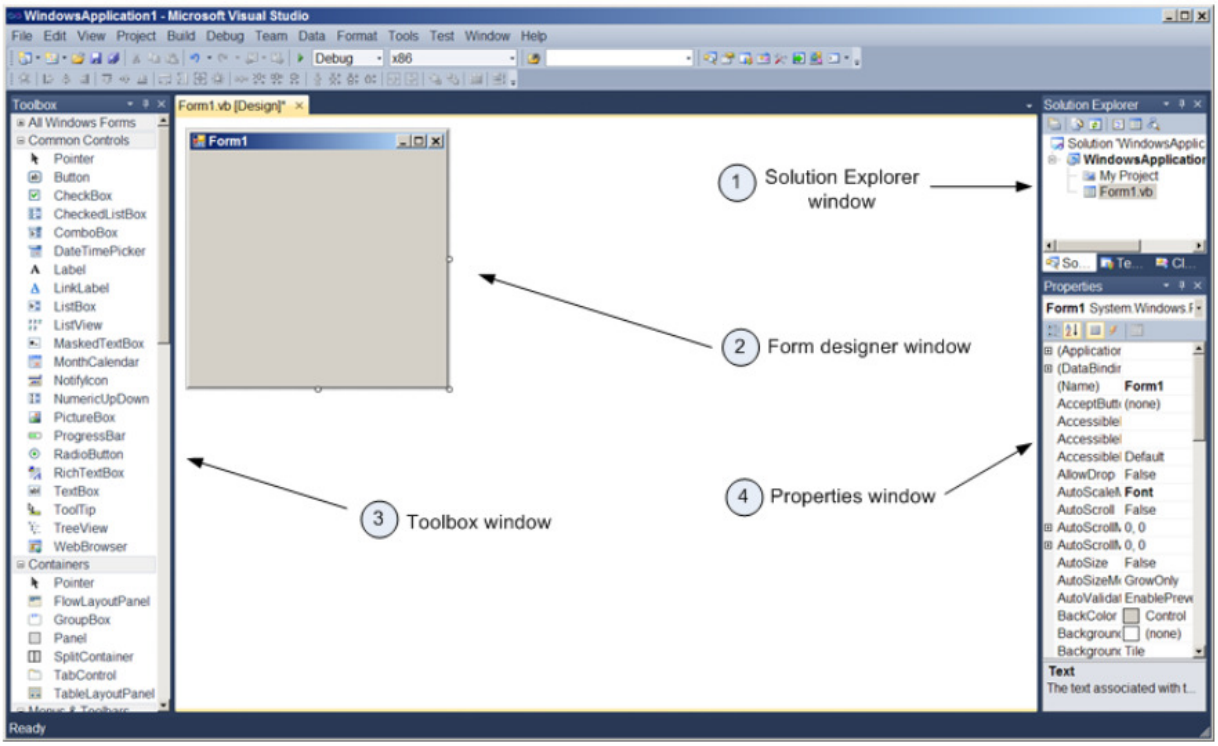
Next, on the Menu Bar, select File → New → Project menu or select New Project from the Start page and you should see a window something like:



This is the New Project form (see the title bar). This is where selections for the type of New Project will be made. There are several important selections to be made on this form:

1. **Project Templates** - this is where the **.Net language** of choice is selected. For us, this should always be **Visual Basic and Windows**.
2. **Project Applications** - this is where we can select from the standard set of predefined applications. For us, this should always be a **Console Application or a Windows Forms Application.**
3. **Name** - the name to be given to the VB.Net application. The name should always be indicative of what the application does, e.g. TaxCalculator (not WindowsApplication1).
4. **Location** - the file system location where the application folder and files will reside.

Once you click the OK button in the New Project form, you will launch **the Visual Basic.Net Integrated Development Environment** (henceforth referred to as the IDE) window. It is in this window that most, if not all development will be coordinated and performed. The startup IDE will look something like the following:



There are four windows on which to focus in this diagram:

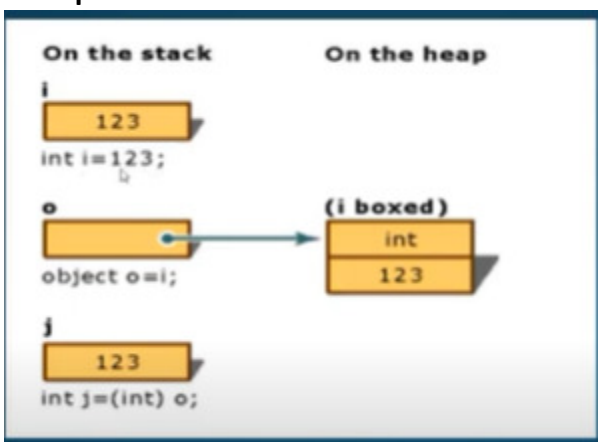
1. **Solution Explorer** - this window contains a **list of all open projects and files/items** associated with the current solution.
2. **Form designer** - this window is where **all controls** for a given solution will be placed and manipulated. A **windows application** may have one, two or many **windows forms** associated with it. Note a **console application** will have **no form designer window**, nor toolbox, since a console application contains no forms.
3. **Toolbox** - this window is where **all VB controls** will be retrieved from. In actuality, you can consider the items in the toolbox as class containers. Retrieving a control from the toolbox is analogous to instantiating an object from that class. Thus clicking on the button item (class) in the toolbox will give you a button object. You can either double click on the control you wish to add to the form, or you can drag and drop your control.
4. **Properties** - this window is where property values are set for a given control.

2.2 Variables and Data Types

2.2.1. Boxing and Unboxing

Boxing: Converting a **value type (store values in stack memory)** to a **reference (Object) type (store Values in Heap memory) is called boxing.** In boxing **implicit conversion (automatic) is take place.**

Unboxing: Converting a **reference type** to **value type** is called unboxing. In unboxing **explicit conversion is take place.**



Example:

```
Private Sub btnconversion_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnconversion.Click
Dim i As Integer = 123
        Dim obj As Object
        obj = i ' boxing ,implicit conversion
        Dim J As Integer
        J = CInt(obj) 'unboxing ,explicit conversion
        MsgBox("Value of i : " & i)
        MsgBox(obj)
        MsgBox("Value of j :" & J)
    End Sub
```

2.2.2. Enumerations

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 enumerationname [ As datatype ]
  memberlist
End Enum
```

Example:

```
Public Class Frmenum
       Enum Temperature
         Low
         Medium
         High
    End Enum
      Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
         Dim value As Temperature = Temperature.Medium
         If value = Temperature.Medium Then
             MsgBox("Temperature is Mediuam..")
         End If
    End Sub
End Class
```

2.2.3. Data Type Conversion Functions

Type conversion is used for **convert one data type to another**. There are types of conversion

(1) **Implicit conversion:** conversion is done **automatically** by the **compiler** is called implicit conversion. It will automatically convert **smaller data types to large data** types. It is also called **narrowing to widening (small to large)** conversion.

(2) **Explicit conversion**: The compiler does not convert a type to another type **automatically**. This type of conversion is called **explicit conversion**. A type conversion keyword is used when performing an explicit conversion. It is also called **widening to narrowing conversion (larger to small).**

Example:

```
Private Sub btnimex_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnimex.Click
        Dim num1 As Integer = 100
        Dim num2 As Integer = 75
        Dim total As Long
        'implicit conversion
        total = num1 + num2
        MsgBox("Total is : " & total)

        An explicite conversion requires function
        Dim num As Integer
        Dim marks As Decimal = 34.75
        'In this the Decimal values are explicitly converted to Integer data type
with rounding the marks 35
        'you have to tell compiler to do the conversion, it uses casting
        num = CInt(marks)
        MsgBox("Converted value is: " & num)
```

CType Function: it uses to convert one type to another type. Instead of remember all conversion functions, remember **only CType function**.

Syntax:

```
CType(expression, typename)
```

Expression: any valid expression

Type name: the name of any **data type, objects, structure, class, or interface**.

```
        Dim text As String = "25.56"
        Dim perd As Double
        Dim perI As Integer
        perd = CType(text, Double) + 1.14
        perI = CType(text, Integer)
        MsgBox("The value of percentage is: " & perd)
        MsgBox("The value of percentage is: " & perI)
```

Type Checking Function

Vb.net provides number of data verification or data type checking function as below. It returns Boolean value (true/false).

(1) IsDate
(2) IsNothing
(3) IsNumeric
(4) IsDBNull
(5) IsArray

(1) IsDate: it returns true or false

Example:

```
Private Sub btnisdate_Click(ByVal sender AsSystem.Object, ByVal e AsSystem.EventArgs)
Handlesbtnisdate.Click
'IsDate returns true or false
Dim dt1 As Date
Dim strtmep As String
Dim bolans As Boolean

dt1 = Now.ToShortDateString
bolans = IsDate(dt1)
MsgBox(bolans) 'True

strtmep = "testing"
bolans = IsDate(strtmep)
MsgBox(bolans)  'false

EndSub
```

(2) IsNothing: Returns true if the object variable that currently **has no assigned** value otherwise, it returns **false**.

Example:

```
Private Sub btnisnothing_Click(ByVal sender AsSystem.Object, ByVal e
AsSystem.EventArgs) Handlesbtnisnothing.Click
'Returns True if the object variable that currently has no assigned vale otherwise
,itreturs false
Dim objtemp As Object
Dim objans As Boolean
objans = IsNothing(objtemp)
MsgBox(objans) 'True
```

```
objtemp = "testing"
objans = IsNothing(objtemp)
MsgBox(objans) 'False

EndSub
```

(3) IsNumeric: Returns True of the value is numeric, otherwise it returns False

```
Private Sub btnnumeric_Click(ByVal sender AsSystem.Object, ByVal e
AsSystem.EventArgs) Handlesbtnnumeric.Click
'Returns True of the value is numeric ,otherwise it returns False
Dim objtemp As Object
Dim objans As Boolean


objtemp = 53
objans = IsNumeric(objtemp)
MsgBox(objans)

objtemp = "Rs.53"
objans = IsNumeric(objtemp)
MsgBox(objans)
EndSub
```

(4)IsDBNull: it returns true if the value evaluates to the DBnulltype, otherwise it returns False

```
Private Sub btnDBnull_Click(ByVal sender AsSystem.Object, ByVal e AsSystem.EventArgs)
Handlesbtn DBnull.Click
'IsDBNull  returns True if the value evalutes to the DBnulltype,otherwise retuns
False

Dim objtemp As Object
Dim objans As Boolean

objans = IsDBNull(objtemp)
MsgBox(objans)

objtemp = System.DBNull.Value
objans = IsDBNull(objtemp)
MsgBox(objans)
EndSub
```

(5) IsArray: Returns True of the value is array, otherwise it returns False.

Example:

```
Private Sub btnisarray_Click(ByVal sender AsSystem.Object, ByVal e
AsSystem.EventArgs) Handlesbtnisarray.Click
'Returns True of the value is array ,otherwise it returs False
Dim ary() As Integer = {1, 2}
Dim objtemp As Object
objtemp = ary
MsgBox(IsArray(objtemp)) ‘True
```

```
EndSub
```

2.2.4. Statements/ option statements /Compile option

There are **four** option statements available in Vb.Net

(1) Option Explicit: It has two modes: **1. on (by default) 2. Off**

If program has **'Option explicit on'** statement than it requires all variables have **proper declaration** otherwise it gives compile time error. If we use **'Option explicit off'** statement than vb.net create **variable declaration automatically** and program does not give an error.

Example:

```
Option Explicit On
Private Sub btnExplicit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExplicit.Click
        Dim ans As Integer
        ans = 5
    End Sub
```

```
Option Explicit Off
Private Sub btnExplicit_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExplicit.Click
        'Dim ans As Integer
        ans = 5
    End Sub
```

Note: Above program run continue without error

Note: For better coding it is recommended to declare variables with Dim keyword and data type.

(2) Option Compare:

It has two modes

1. Binary (by Default)
2. Text

we can change string comparison method by set the text or Binary

Example:

```
Private Sub btncompare_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btncompare.Click
        If "Hello" = "HELLO" Then
            MessageBox.Show("True")
        Else
            MessageBox.Show("False")
        End If
    End Sub
```

In above program if we use Option Compare Binary, messagebox show 'false' and if we use 'Text' mode than messagebox show 'True. That means when we set Option Compare to Text we can able compare string with Case insensitive comparision.

(3)Option Strict

Option Strict is prevents program from automatic variable conversions that is **implicit** data type conversions. It will check strictly **Type Checking**. While converting one data type to another if there is **data loss** then it will show a compile time error (Narrowing conversion).

It has also two modes:

1. On
2. Off (by default)

Example:

```
Private Sub btnstrict_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnstrict.Click
        Dim no1 As Double
        Dim no2 As Integer
        no1 = 9.123
        MsgBox(no1)
        no2 = no1
        MsgBox(no2)
    End Sub
```

The above program is a normal vb.net program and is in default Option Strict Off mode so we can convert the value of Double to an Integer.

(4)Option Infer

The Infer option indicates whether the compiler **should determine the type of a variable from its value.**

It has also two modes:

1. On (by default)
2. Off

Example:

```
Private Sub btninfer_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btninfer.Click
        Dim a = 25 'it is considered as integer
        Dim b = "Hello" ' it is considered as string
    End Sub
```

Example:

Option infer on.
Dim a=25
Take the mouse over variable a.
a's data type is integer(see in tooltip).

Option infer off.
Dim a=25
Take the mouse over variable a.
a's data type is object (see in tooltip).

2.3. String & Date Functions and Methods

2.3.1 String Function:

1. **Len:** Returns an integer that contains the number of characters in a string.
   Syntax: Len (string)
2. **Mid:** Returns a string containing a specified number of characters from a string.
   Syntax: Mid (string, start [, length])
   String - String expression from which characters are returned.
   Start - Long. Character position in string at which the part to be taken begins.
   Length - Length is Optional. Number of characters to return.
3. **Left:** Returns a string containing a specified number of characters from the left side of a String.
   Syntax: Left ("string", n)
4. **Right:** Returns a string containing a specified number of characters from the right side of string.
   Syntax: Right ("string", n)
5. **Space:** Returns a string consisting of the specified number of spaces.
   Syntax: Space (number)
6. **Replace: Returns** a string in which a specified substring has been replaced with another substring.
   Syntax: Replace (string, searchtext, replacetext)
7. **Trim:** Returns a string containing a copy of a specified string with no leading or trailing spaces.
   Syntax: Trim ("String")
8. **Ltrim: Returns** a string containing a copy of a specified string with no leading spaces.
   Syntax: Ltrim("string")
9. **Rtrim**: Returns a string containing a copy of a specified string with no trailing spaces.
   Syntax: Rtrim ("string")
10. **Ucase:** Returns a string or character containing the specified string converted to uppercase.
    Syntax: Ucase ("string")
11. **Lcase: Returns** a string or character converted to lowercase.
    Syntax: LCase("string")

11 InStr: Returns an **integer** specifying the start position of the **first occurrence** of one string within another.

Syntax: Instr (n, original Text, embedded Text)

Example - This example defines all the string function.

```
Private Sub btnstringfun_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnstringfun.Click
        'Len Function
        Dim leng As Integer= Len("Visual Basic")
        MessageBox.Show("length is :" & leng)
        'Mid Function
        Dim middle As String = Mid("Visual Basic", 3, 4)
        MessageBox.Show("Mid is :" & middle)
        'Replace Function
        Dim replaces As String = Replace("Visual Basic", "Basic", "Advance")
        MessageBox.Show("Replace is :" & replaces)
        'Trim Function
        Dim trimt As String = Trim("   Visual Basic  ")
        MessageBox.Show("Trim is :" & trimt)
        'Ltrim Function
        Dim ltriml As String = LTrim("    Visual Basic     ")
        MessageBox.Show("Ltrim is :" & ltriml)
        'Rtrim Function
        Dim rtrimr As String = RTrim("    Visual Basic     ")
        MessageBox.Show("Rtrim is :" & rtrimr)
        'Ucase Function
        Dim ucaseu As String = UCase("Visual Basic")
        MessageBox.Show("Ucase is :" & ucaseu)
        'Lcase Function
        Dim lcasel As String = LCase("VISUAL CASE")
        MessageBox.Show("Lcase is :" & lcasel)
        'Instr Function
        MsgBox(InStr(1, "Visual Basic", "Basic"))
    End Sub
```

2.3.2 Math Function:

1) **Abs**

Returns the absolute value of a number.

Syntax:

Abs(n)

2) **Max**

Returns the larger of two numbers.

Syntax:

Max(n1,n2)

3) **Min**

Returns the smaller of two numbers.

Syntax:

Min(n1,n2)

4) **Pow**

Returns a specified number raised to the specified power.

Syntax:

Pow(n1,n2)

Here, n1 is the number and n2 is the power of the number.

5) **Sqrt**

Returns the square root of a specified number.

Syntax:

Sqrt(n1)

6) **Ceiling**

Returns the smallest integral value that's greater than or equal to the specified Decimal or Double.

Syntax:

Ceiling(n)

7) **Floor**

Returns the largest integer that's less than or equal to the specified Decimal or Double number.

Syntax:

Floor(n)

Example:

```
Private Sub btnmath_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnmath.Click

    Dim absolute As Integer
    absolute = Abs(-16)
MessageBox.Show("The absolute value is: " & absolute)

    Dim minimum As Integer
    minimum = Min(18, 12)
MessageBox.Show("The minimum value is: " & minimum)

    Dim maximum As Integer
    maximum = Max(18, 12)
MessageBox.Show("The maximum value is: " & maximum)

    Dim square As Integer
    square = Sqrt(9)
MessageBox.Show("The square root is: " & square)

    Dim power As Integer
    power = Pow(2, 3)
MessageBox.Show("The power is: " & power)

---Remaining
  End Sub
```

2.3.3 Date Function:

1). DateSerial:-
It returns a **Date value** representing a specified Year,Month,and Day.
Syntax: -

DateSerial(Year,Month,Day)

2). Year: -
It will extract Year part from any Date.It returns only **integer value.**
Syntax: -

Year(Date)

3). Month:-
It will extract Month part such as 1,2,3,4 and so on from any Date.It returns only **integer value**.
Syntax: -

Month(Date)

4). MonthName:-

It will show Month Name as January,February and so on from any Date.It returns only **string value**.
Syntax: -

MonthName(Month)

5). Day:-
It will display Day in number.It returns only **integer** value.Actually it's an **Enum** which as Sunday,Monay and so on. It specifies the day of the week.
Syntax: -

Day(Date)

6) DateDiff
The DateDiff function returns the number of intervals between two dates.
Syntax

DateDiff(interval,date1,date2)

The **interval** you want to use to calculate the differences between date1 and date2
Can take the following interval values:

yyyy	Year
q	Quarter
m	Month
y	Day of year
d	Day
w	Weekday
ww	Week of year
h	Hour
n	Minute
s	Second

7) FormatDateTime

Display a date in different formats:

Syntax:

FormatDateTime(date,format)

A value that specifies the date/time format to use. YouCan take the following values:

0 = vbGeneralDate– Default.	Returns date: mm/dd/yyyy and time if specified: hh:mm:ss PM/AM.
1 = vbLongDate	Returns date: weekday, monthname, year
2 = vbShortDate	Returns date: mm/dd/yyyy
3 = vbLongTime	Returns time: hh:mm:ss PM/AM
4 = vbShortTime	Return time: hh:mm

2.3.4 Methods:

In visual basic, Method is **a separate code block** and that will contain **a series of statements** to **perform particular operations**. Generally In visual basic **Methods** are useful to improve the code **reusability** by reducing the **code duplication**. Suppose. If we have the **same functionally** to perform in **multiple places**, then we can create **one method** with the required functionality and use it wherever it is required in the application.

In visual basic, we can create the **Methods** either by using **Sub** or **Function** keywords like as shown below. **If we create a method with Sub keyword that will not allow us to return any value. In case, if you want to return any value, then you need to use Function keyword to create the method.**

Syntax of Visual Basic Methods

```
<Access Specifier> Sub Method_Name([<Parameters>])
        Statements to Execute
End Sub
```

OR

```
<Access_Specifier> Function Method_Name(<Parameters>) As <Return_Type>
        Statements to Execute
        Return return val
End Function
```

Parameter	Description
Access_Specifier	It is useful to define an access level either **public or private** etc. to allow other classes to access the method. If we didn't mention any access modifier. Then by **default** it is **private**.
Method_Name	It must be a **unique name** to identify the method.
Parameters	The method parameters are useful **to send or receive data from a method** and these are enclosed within parentheses and are **Separated by commas**. In case. If no parameters are required for a method then. We need to define a method with **empty Parentheses**.
Return_Type	It is useful to specify the type of value the method can return

2.4. Modules, Procedures and Functions

Modules

- It is a container for a group of related **functions, subroutines, properties, and variables** that can be accessed from any part of the program without **having to create an instance** of the module.

Example:

Right click on your project and add module file then write down the following code in module file.

Module Code:

```
Module modulemathop
    Public Function Add(ByVal num1 As Integer, ByVal num2 As Integer) As
Integer
        Return num1 + num2
    End Function

    Public Function Subtract(ByVal num1 As Integer, ByVal num2 As Integer) As
Integer
        Return num1 - num2
    End Function
End Module
```

Form Code:

```
Private Sub btnadd_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnadd.Click
        Try
            MsgBox(Add(txtno1.Text, txtno2.Text))
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub

    Private Sub btnsub_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnsub.Click
        Try
            MsgBox(Subtract(txtno1.Text, txtno2.Text))
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try
    End Sub
```

Procedures:

Procedures are made up of series of Visual Basic statements that, when called, are executed. After the call is finished, control returns to the statement that called the procedure.

Syntax:

```
<Access Specifier> Sub Method_Name(<Parameters>)
        Statements to Execute
End Sub
```

Example:

```
'procedue without argumet
    Public Sub clearcontrol()
        txtname.Text = ""
        txtadd.Text = ""
        txtage.Text = ""
    End Sub
    'procedue with argumet
    Public Sub hi(ByVal str As String)
        MessageBox.Show("hello how r u" & str)
    End Sub

Private Sub btnprocedure_Click(…) Handles btnprocedure.Click
      ‘call procedure
        hi(txtname.Text) 'with argument
        clearcontrol() 'without argument
End Sub
```

Passing Parameters by Value

This is the default mechanism for passing parameters to a method. In this mechanism, when a method is called, a **new storage location** is created for each value parameter. The values of the actual parameters are copied into them. So, the changes made to the parameter inside the method have **no effect on the argument**.

In VB.Net, you declare the reference parameters using the **ByVal** keyword.

Example:

```
Sub swapbyval(ByVal x As Integer, ByVal y As Integer)
        Dim temp As Integer
        temp = x ' save the value of x
        x = y    ' put y into x
        y = temp 'put temp into y
End Sub

Private Sub btnbyval_Click(…) Handles btnbyval.Click
        MsgBox("before swap")
        MessageBox.Show(txtno2.Text)
        MessageBox.Show(txtno3.Text)

        swapbyval(txtno2.Text, txtno3.Text)
        MsgBox("after swap")
        MessageBox.Show(txtno2.Text)
        MessageBox.Show(txtno3.Text)
End Sub
```

Passing Parameters by Reference

A reference parameter is a reference to a memory location of a variable. When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters. The reference parameters represent the same memory location as the actual parameters that are supplied to the method.

In VB.Net, you declare the reference parameters using the **ByRef** keyword.

Example:

```
Sub swapbyref(ByRef x As Integer, ByRef y As Integer)
        Dim temp As Integer
        temp = x ' save the value of x
        x = y    ' put y into x
        y = temp 'put temp into y
End Sub

 Private Sub btnref_Click(…) Handles btnref.Click
        MsgBox("before swap")
        MessageBox.Show(txtno2.Text)
        MessageBox.Show(txtno3.Text)

        swapbyref(txtno2.Text, txtno3.Text)
        MsgBox("after swap")
        MessageBox.Show(txtno2.Text)
        MessageBox.Show(txtno3.Text)
End Sub
```

Functions:

The Function statement is used to declare the name, parameter and the body of a function. The syntax for the Function statement is

Syntax:

```
<Access_Specifier> Function Method_Name(<Parameters>) As <Return_Type>
        Statements to Execute
        Return return val
End Function
```

Example:

```
Public Function add(ByVal a As Integer, ByVal b As Integer) As Integer
        Return a + b
    End Function

Private Sub btnfun_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnfun.Click
        Dim ans As Integer = add(txtno1.Text, txtno2.Text)
        MsgBox(ans)
    End Sub
```

2.4.1. Passing variable number of arguments

- You can pass a variable number of arguments to a function using the "**ParamArray**" keyword. This allows you to pass **any number of arguments** to the function, and they will be treated as an **array**.
- **Example:**
  Function that accepts a variable number of arguments

```
Public Function ConcatStrings(ByVal separator As String, ByVal ParamArray
strings() As String) As String
        Dim result As String = String.Join(separator, strings)
        Return result
End Function
```

Call this function and write down following code

```
Dim result1 As String = ConcatStrings(",", "apple", "orange", "banana")
MsgBox(result1)
Dim result2 As String = ConcatStrings("-", "one", "two", "three", "four")
MsgBox(result2)
```

- In the first call, the separator is a comma, and three string arguments are passed. In the second call, the separator is a dash, and four string arguments are passed. Both calls will return a concatenated string that combines all of the arguments with the specified separator.

2.4.2. Optional arguments

The optional parameter contains a default value with **Optional keyword**. If we will not pass the value for optional parameters then it will use the default value. If we pass the values then it will override value.

```
Function AddNum(ByVal num1 As Integer, Optional ByVal num2 As Integer = 20, Optional
ByVal num3 As Integer = 30) As Integer
        Return num1 + num2 + num3
    End Function

Private Sub btnoptionalarg_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnoptionalarg.Click
        Dim result As Integer = 0
        result = AddNum(10)
        MsgBox("Addition is: {0}" & result)

        result = AddNum(10, 40)
        MsgBox("Addition is: {0}" & result)

        result = AddNum(10, 40, 60)
        MsgBox("Addition is: {0}" & result)
    End Sub
```

2.5. Arrays and Collections

Arrays:

- It stores a fixed size sequential collection of elements of the same type.
- It is used to store a collection of data.
- It consists of contiguous memory locations.
- The lowest element corresponds to the first and the highest element to the last.
- It provides best performance for certain requirements.
- The elements in an array can be stored and accessed by using the index of the array.

To declare an array in VB.Net, you use the **Dim** statement

Example:

(1) Dim intData(10) As Integer ' an array of 11 elements
(2) Dim strData(20) As String ' an array of 21 strings
(3) Dim twoDarray(10, 20) As Integer'a two dimensional array of integers
(4) Dim ranges(10, 100) 'a two dimensional array.

You can also **initialize** the array elements while declaring the array.

Example:

(1) Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
(2) Dim names() As String = {"Karthik", "Sandhya", "Shivangi", "Ashwitha", "Somnath"}
(3) Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}

Example

```
Private Sub btnarry_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnarry.Click
        lstarr.Items.Clear()
        Dim num(3) As Integer
        num(0) = 10
        num(1) = 20
        num(2) = 30
        For i = 0 To 2
            lstarr.Items.Add(num(i))
        Next
    End Sub
```

Dynamic Arrays

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the **ReDim** statement.

Syntax:

```
ReDim [Preserve] arrayname(subscripts)
```

Where,

- The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
- Arrayname is the name of the array to re-dimension. Subscript specifies the new dimension.

Example

```
Private Sub btndyarray_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btndyarray.Click
        lstarr.Items.Clear()
        Dim marks() As Integer
        ReDim marks(2)
        marks(0) = 85
        marks(1) = 75
        marks(2) = 90
        MsgBox(marks.Length)
        'ReDim marks(10)
        ReDim Preserve marks(10)
        marks(3) = 80
        marks(4) = 76
        marks(5) = 92
        marks(6) = 99
        marks(7) = 79
        marks(8) = 75
        MsgBox(marks.Length)
        For i = 0 To 10
            lstarr.Items.Add(i & vbTab & marks(i))
        Next i
    End Sub
```

Multi-Dimensional Arrays

VB.Net allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays.

You can declare a 2-dimensional array of strings as –

Dim twoDStringArray(10, 20) As String

3-dimensional array of Integer variables –

Dim threeDIntArray(10, 10, 10) As Integer.

Example of Array properties

```
Private Sub btnproperty_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnproperty.Click
        Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
        Dim arrymethod As Integer
        'Property of Array
        arrymethod = list.Length
        MessageBox.Show("Lenght of the Array." & arrymethod)
        arrymethod = list.GetLength(0)
        MessageBox.Show("Lenght of the specified dimension of Array." & arrymethod)
        arrymethod = list.GetUpperBound(0)
        MessageBox.Show("Returns the highest subscript(index)value." & arrymethod)
        arrymethod = list.GetLowerBound(0)
        MessageBox.Show("Returns the lowest subscript(index)value." & arrymethod)
        arrymethod = list.Rank
        MessageBox.Show("gets the rank (number of dimensions) of the Array." &
arrymethod) '1

        Dim arry2(5, 4, 3) As Integer
        arrymethod = arry2.GetLength(0)
        MessageBox.Show("Lenght of the specified dimension of Array." & arrymethod)
        arrymethod = arry2.GetLength(1)
        MessageBox.Show("Lenght of the specified dimension of Array." & arrymethod)
        arrymethod = arry2.GetLength(2)
        MessageBox.Show("Lenght of the specified dimension of Array." & arrymethod)
        arrymethod = arry2.Rank
        MessageBox.Show("gets the rank (number of dimensions) of the Array." &
arrymethod)
        Dim arry3(5, 4) As Integer
        arrymethod = arry3.GetLength(0)
        MessageBox.Show("Lenght of the specified dimension of Array." & arrymethod)
        arrymethod = arry3.GetLength(1)
        MessageBox.Show("Lenght of the specified dimension of Array." & arrymethod)
        arrymethod = arry3.Rank
        MessageBox.Show("gets the rank (number of dimensions) of the Array." &
arrymethod)
    End Sub
```

Collections:

- A collection can also store group of objects. But unlike an array which is of fixed length, the collection can grow or shrink dynamically.
- Items can be added or removed at run time.
- These are the specialized classes for data storage and retrieval.
- It supports stack, queues, lists and hash tables.

Collection includes various classes are as follows:

Class	Description
ArrayList	It represents ordered collection of an object that can be indexed individually.
Hashtable	It uses a key to access the elements in the collection.
SortedList	It uses a key as well as an index to access the items in a list.
Stack	It represents LIFO(Last-In-First-Out) collection of object.
Queue	It represents FIFO(First-In-First-Out) collection of object.

Array List

It represents ordered collection of an object that can be indexed individually.

Example:

```
Private Sub btnarraylist_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btncollection.Click
        Dim i As Integer
        Dim ItemList As New ArrayList()
        ItemList.Add(1)
        ItemList.Add("Dhyan")
        ItemList.Add("7 years")
        ItemList.Add("Boy")
        ItemList.Add("Surat")
        For i = 0 To ItemList.Count - 1
            ListBox1.Items.Add(ItemList.Item(i))
        Next
    End Sub
```

Methods of Array List

```
Private Sub btnarrylistmethod_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btncollection.Click
        Dim i As Integer
        Dim ItemList As New ArrayList()
        ItemList.Add(1)
        ItemList.Add("Dhyan")
        ItemList.Add("7 years")
        ItemList.Add("Boy")
        ItemList.Add("Surat")
        For i = 0 To ItemList.Count - 1
            ListBox4.Items.Add(ItemList.Item(i))
```

```
        Next

        ' ''Method Add,insert,remove,RemoveAt,sort
        'ItemList.Remove(1)
        ItemList.Remove("7 years")
        For i = 0 To ItemList.Count - 1
            lstremove.Items.Add(ItemList.Item(i))
            'MsgBox(ItemList.Item(i))
        Next

        ''insert an item index start with 0
        ItemList.Insert(0, "Patel")
        ' ItemList.Add(3)

        For i = 0 To ItemList.Count - 1
            lstinsert.Items.Add(ItemList.Item(i))
            'MsgBox(ItemList.Item(i))
        Next
        ''sort itemms in an arraylist
        'ItemList.Sort()

        'remove item from a specified index
        ItemList.RemoveAt(3)


        For i = 0 To ItemList.Count - 1
            lstremoveat.Items.Add(ItemList.Item(i))
            'MsgBox(ItemList.Item(i))
        Next
    End Sub
```

Hashtable: It uses a key to access the elements in the collection.

```
Private Sub btnhasttable_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnhasttable.Click
        ListBox4.Items.Clear()
        Dim h1 As New Hashtable
        h1.Add(14, "India")
        h1.Add(15, "Dhyan")
        h1.Add(16, "boy")
        h1.Add(17, "Surat")
        For Each key In h1.Keys
            ListBox4.Items.Add(h1(key))
        Next
    End Sub
```

SortedList: It uses a key as well as an index to access the items in a list.

```
Private Sub btnsortlist_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnsortlist.Click
        Dim s1 As New SortedList 'combination of Array and Hast Table
        s1.Add(1, "Dhyan")
```

```
        s1.Add(2, "Surat")
        For i = 0 To s1.Count - 1 'index is used
            ListBox4.Items.Add(s1.GetByIndex(i))
        Next
        For i = 0 To s1.Count - 1 'key is used
            ListBox4.Items.Add(s1.GetKey(i))
        Next
    End Sub
```

2.6. Control Flow Statements

2.6.1. Conditional Statements

It is used to decide the flow of program. It enables us to execute a certain set of statements based on condition. It is also known as Branching statements or test structure or decision structure.

Conditional Statements includes

(1) If … Then … End If
(2) If … Then … Else … End If
(3) If … Then … ElseIf … End If
(4) Nested If … Then … End If
(5) Select Case…. End Select

(1) If …Then … End If Statement

Syntax:

```
If condition Then
      Statement Block
End if
```

It works as follow:

(1) First a condition is checked.
(2) If the condition is TRUE then the statement block will execute.
(3) If the condition is FALSE then the statement block will not execute and the control is transferred to the statement after the End If statement.

Example:

```
Private Sub btnendif_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnendif.Click
        If txtname.Text = "" Then
            MsgBox("Enter your Name")
            txtname.Focus()
        End If
End Sub
```

(2) If … Then … Else … End If

Syntax:

```
If condition then
    Statement block 1
Else
   Statement block 2
End if
```

The above structure works as follow :(1) First a condition is checked. (2) If the condition is TRUE then it will execute the statement block 1.(3) If the condition is FALSE then it will execute the statement block 2.

Example:

```
If txtname.Text = "" Then
    MsgBox("Enter your name")
    txtname.Focus()
Else
    MsgBox("your name is" & txtname.Text)
End If
```

(3) If … Then … ElseIf … End If

Syntax:

```
If Condition1 Then
    Statement Block 1
ElseIf Condition2 Then
    Statement Block 2
ElseIf Condition3 Then
   Statement Block 3
……..
ElseIfConditionN Then
    Statement Block N
Else
   Default Statement Block
End if
```

The above structure works as follow:

(1) First Condition1 is checked.

(2) If the Condition1 is TRUE then it will execute statement block 1.

(3) If the Condition1 is FALSE then Condition2 is checked and the same process is repeated until any of the condition specified becomes TRUE.

(4) If all the condition evaluates to false then it will execute the default statement block followed by the else statement.

Example:

```
Private Sub ifthenelseif_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ifthenelseif.Click
```

```
        If txtname.Text = "" Then
            MsgBox("Enter your Name")
            txtname.Focus()
        ElseIf txtadd.Text = "" Then
            MsgBox("Enter your Address")
            txtadd.Focus()
        ElseIf txtage.Text = "" Then
            MsgBox("Enter your Age")
            txtage.Focus()
        Else
            MsgBox(txtname.Text & txtadd.Text & txtage.Text)
        End If
    End Sub
```

(4) Nested If … Then … End If

When one If … Then … End If statement is contained within another If … Then … End If statement then it is known as Nested If … Then … End If statement.

Syntax:

```
If condition-1 then
   If condition-2 then
      Statement block 1
  Else
     Statement block 2
   End if
Else
    Statement block 3
End if
```

The above structure works as follow:

(1) First Condition1 is checked.

(2) If condition1 is TRUE then it tests for condition2.

(3) If condition2 is TRUE then it executes statement block 1.

(4) If condition2 is FALSE then it executes statement block 2.

(5) If condition1 is FALSE then it executes statement block 3.

Example:

```
Private Sub btnnestedif_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnnestedif.Click
        Dim age As Integer
        If txtage.Text <> "" Then
            If IsNumeric(txtage.Text) Then
                age = txtage.Text
                MsgBox("your age is" & age)
            Else
                MsgBox("Please Enter Numeric Value")
                txtage.Focus()
                txtage.Text = ""
            End If
        Else
```

```
            MsgBox("Please Enter age")
            txtage.Focus()
        End If
End Sub
```

(5) Select Case … End Select

It is also known as multiple choice decision statement. It allows you to select one option from the list of available options. It is the alternative of If …Then ... Else If structure.

Syntax

```
Select Case expression
Case Value1
    Statement Block 1
Case Value 2
    Statement Block 2
……….
Case Value N
    Statement Block N
Case Else
   Default statement Block
End Select
```

The above structure works as follow:

(1) It compares the value of expression against the list of values specified in the different
case values. When the match is found it executes the statement block associated with that case value.

(2) If no match is found then the default statement associated with Case Else will executes.

Example:

```
Private Sub btnselectcase_Click_1(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnselectcase.Click
        Dim A As Integer, B As Integer, C As Integer
        Dim op As String
        Select Case op
            Case "+"
                C = A + B
            Case "-"
                C = A - B
            Case "*"
                C = A * B
            Case "/"
                C = A / B
            Case Else
                MsgBox("Wrong Option")
        End Select
    End Sub
```

Some Features of Select Case … End Select structure:

(1) If you want to specify the action to be taken on the values between specified ranges then you can also specify range in the case value as shown below:

```
Dim age As Integer
        age = txtage.Text
        Select Case age
            Case 1 To 6
                MsgBox("Kid")
            Case 7 To 18
                MsgBox("Tin Age")
            Case 19 To 100
                MsgBox("Adult")
        End Select
```

(2) You can also specify multiple values with the Case as shown below:

```
Select Case Grade
            Case 10, 9
                MsgBox("Excellent")
            Case 8, 7
                MsgBox("Very Good")
            Case 6, 5
                MsgBox("Good")
            Case Else
                MsgBox("Poor")
        End Select
```

2.6.2. Loop Statements

Looping Statements are used when a group of statement is to be executed repeatedly until a condition is true or false.

Vb.net supports the following loop Statements

(1) For …Next
(2) While …end while
(3) Do loop
(4) For each next

(1) For… Next

It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.

Syntax:

```
For counter = start To end [Step increment/decrement]
  Logic
Next
```

Example

```
Private Sub btnfornext_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnfornext.Click
        Dim i As Integer
        For i = 0 To 10
            MessageBox.Show("The value of i is:" & i)
        Next
End Sub
```

(2) While loop

While loop keeps executing until the condition against which it tests remain true.

Syntax:

```
While Condition
        Logic
End while
```

Example:

```
Private Sub btnwhileloop_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnwhileloop.Click
        Dim num1 As Integer = 1
        While num1 < 10
            MessageBox.Show("The value of num1 is:" & num1)
            num1 = num1 + 1
        End While
    End Sub
```

(3) Do loop

Repeats a block of statements while a Boolean condition is true or false.

Syntax:

```
Pretest
Do {While/until} condition
        Logic
Loop
```

OR

```
PostTest
Do
        Logic
Loop {While/until} condition
```

We can use either while or until

While: Repeat the loop until condition is true.

Until: Repeat the loop until condition is false.

Example:

Do until

```
Private Sub btndoloop_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btndoloop.Click

        Dim num1 As Integer = 1
            Do Until num1 = 5
            MessageBox.Show("The value of num1 is:" & num1)
            num1 = num1 + 1
        Loop

    End Sub
```

Do while

```
Private Sub btndoloop_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btndoloop.Click
        Dim num1 As Integer = 1
        Do While num1 < 5 '1 to 4
            MessageBox.Show("The value of num1 is:" & num1)
            num1 = num1 + 1
        Loop

    End Sub
```

For each…Next

The for each … next loop is similar to the for…next loop but it executes the statements block for each element in a collection or array Repeats a group of statements for each element in collection

Syntax

```
For each element in group
        Logic
Next
```

Example

Print array data for each loop

```
Private Sub btnforeachnext_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnforeachnext.Click
        Dim arr(2), temp As Integer
        arr(0) = 10
        arr(1) = 20
        arr(2) = 30
        For Each temp In arr
            MsgBox(temp)
        Next
    End Sub
```

With…End with

Executes a series of statements making repeated reference to a single object. To make this type of code more efficient and easier to read, we use this block. The uses of it do not require calling again and again the name of the object Set multiple properties and methods quickly.

Syntax:

```
With object Name
        Logic
End with
```

Example:

```
Private Sub FrmControlStru_Load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load
        With btnsave
            .Text = "update"
            .ForeColor = Color.Red
            .BackColor = Color.Black
        End With
    End Sub
```

2.6.3. MsgBox and InputBox

Modal or Modeless Dialog boxes.

Forms and dialog boxes are either **modal or modeless**.

Modal dialog box

- A Modal dialog box is one that the user must first close in order to have access to any other window or dialog box of the same application.
- When it is displaying, the user cannot use any other part of WordPad unless he or she closes this object first. Dialog boxes that display important message must be modal.

Example

- Date and Time dialog box, Save dialog box, save as dialog box, Messagebox and Inputbox

Modeless dialog box

- Modeless dialog box is if the user does not have to close it in order to continue using the application that owns the dialog box.
- We can jump or shift the focus between the other object.
- Parallel we can work with multiple objects.
- Modeless forms or dialog box are difficult to manage, because users can access them in unpredictable order.

Example:

The **Find and the Replace dialog boxes** of most applications is an example of a **modeless dialog box.** If it is opened, the user does not have to close it in order to use the application or the document in the background.

MsgBox

Displays a message in a dialog box and wait for the user to click a button, and returns an integer indicating which button the user clicked.**MsgBox is the model dialog box.**

This format is as follows:

yourMsg=MsgBox(Prompt, button+icon style, Title)

- Prompt (Compulsory), will display the message in the message box.
- The Style Value (Optional) will determine what type of command buttons appear on the message box.
- The Title (Optional) argument will display the title of the message box.

Table: Return Values and Command Buttons

Table: Return Values and Command Buttons		
Value	**Named Constant**	**Button Clicked**
1	vbOk	Ok button
2	vbCancel	Cancel button
3	vbAbort	Abort button
4	vbRetry	Retry button
5	vbIgnore	Ignore button
6	vbYes	Yes button
7	vbNo	No button

Example

```
Msgbox (“Hi”)
Msgbox (“Welcome”, “Message to User”)

Private Sub btnmsgbox(…) Handles btnmsgbox.Click
    Dim testMsg As Integer
```

```
testMsg = MsgBox("Click to Test", vbYesNoCancel + vbExclamation, "Test Message")
      If testMsg = 6 Then
MessageBox.Show("You have clicked the yes button")
ElseIftestMsg = 7 Then
MessageBox.Show("You have clicked the NO button")
      Else
MessageBox.Show("You have clicked the Cancel button")
      End If
   End Sub
```

Messagebox

- Displays a dialog box that can contain **text, buttons, and symbols to inform the user**.
- It is an **advance** version of Msgbox function.Messagebox is the class and show is the method of it.

Syntax

MessageBox.Show (Text, caption, button, icon, defaultbutton, option, helpbutton)

Text: Display a message box with specified text.

Caption:Display a message box with specified text and Caption.

Button:Display a message box with specified text and Caption and button.

Icon:Display a message box with specified text,caption, button and icon.

Default button:Display a message box with specified text,caption,button, icon and default button.

Option:Display a message box with specified text,caption,button, icon and default button. option

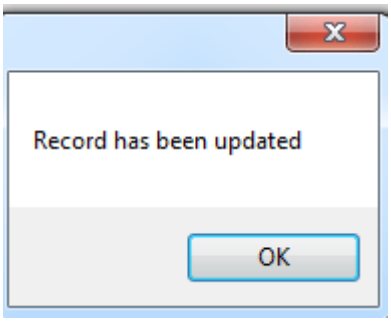
Help button:Display a message box with specified text,caption,button, icon and default button.

1. General message

The simplest form of a Messagebox is a dialog with a text and OK button. The following code creates a simple Messagebox.

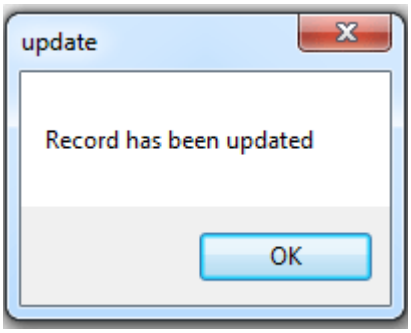
Example1-To display a simple message.

MessageBox.Show ("Record has been updated")



Example2-To display a message with title.

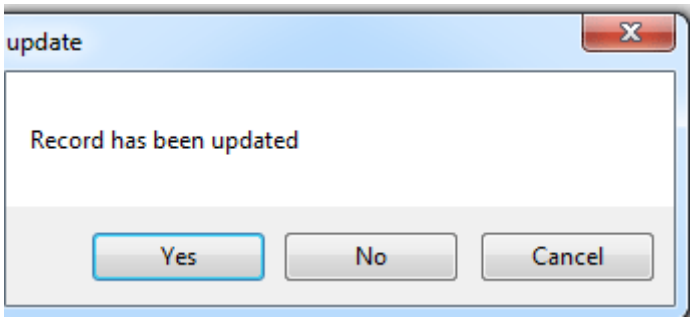
MessageBox.Show ("Record has been updated", "update")



Example3-Message box with caption and buttons.

MessageBoxButtons-specifies which buttons to display on a message box.

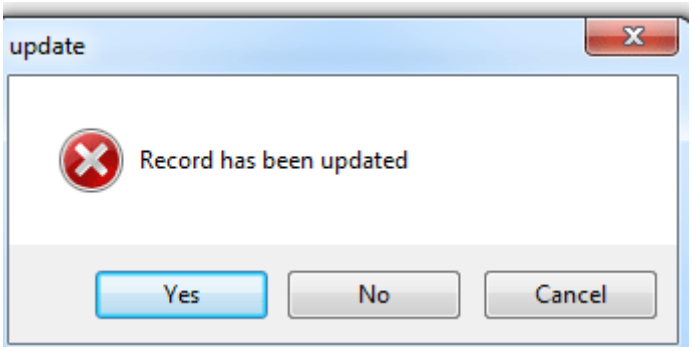
MessageBox.Show ("Record has been updated", "update", MessageBoxButtons.YesNoCancel)



Example4-Message box with caption, buttons and icon.

MessageBox.Show ("Record has been updated", "update", MessageBoxButtons.YesNoCancel,

MessageBoxIcon.Error)



The InputBox ()

An InputBox () function will display a message box where the user can enter a value or a message in the form of text. You can use the following format:

myMessage=InputBox(Prompt, Title, default_text, x-position, y-position)

- myMessage is a variant data type but typically it is declared as string, which accept the message input by the users. The arguments are explained as follows:
- Prompt - the message displayed normally as a question asked.
- Title - The title of the Input Box.
- default-text - The default text that appears in the input field where users can use it as his intended input or he may change to the message he wishes to enter.
- X-position and y-position are the position or the coordinates of the input box.

Example:

```
Private Sub btninputbox_Click(…) Handles btninputbox.Click
    Dim ans As String
ans = InputBox("enter Name", "Information", "enter your name here")
MsgBox(ans)
End Sub
```