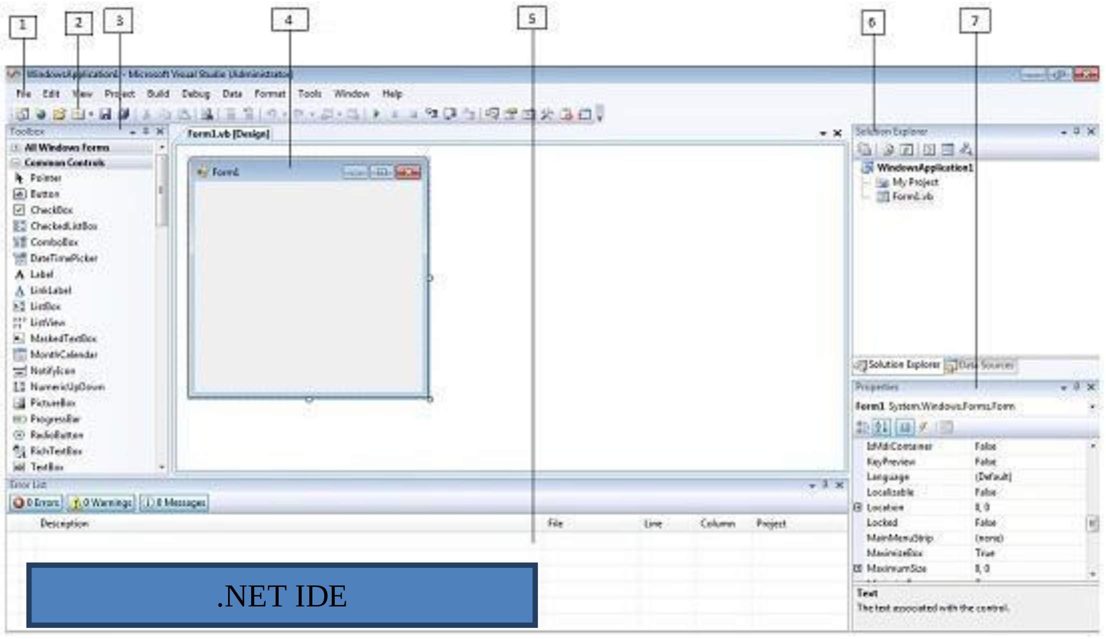
IDE

An integrated development environment (IDE), also known as integrated design environment and integrated debugging environment, is a type of computer software suite that assists computer programmers to develop software.

The integrated development environment (IDE) is important in helping you create, run and debug any .Net programs or applications. You can consider VB.Net IDE as Microsoft Word and VB.Net programs as Word documents.



1. **Menu Bar**

   It consists of menus that help you manipulate VB.Net programs in the project. The menus are listed from left to right as File, Edit, View, Project, Build, Debug, Tools, Window, and Help.

2. **Standard Toolbar**

Contains buttons that are shortcuts to some commonly used menu items.

3. Toolbox (Ctrl + Alt +X)

The window is very important in the VB.Net IDE. It contains control templates or components that are available for you to use. You can simply drag and drop any control from toolbox to your form

4. Forms Designer (Shift + F7)

We can drag and drop controls in this view of the form. We can also see some type of preview of our form

5. Output Window

The Output window is where many of the tools, including the compiler, send their output. Every time you start an application, a series of messages is displayed in the Output window. These messages are generated by the compiler, and you need not understand them at this point. If the Output window is not visible, choose View > Other Windows > Output from the menu.

6. Solution Explorer (Ctrl + Alt +L)

The window contains a Windows Explorer-like tree view of all the customizable forms and general code (modules) that make up a VB.Net application. The Solution Explorer provides you with an organized view of your project and program files associated with the project. Select the Solution Explorer on the View menu when you cannot find the Solution Explorer in your IDE.

7. Properties Window (F4)

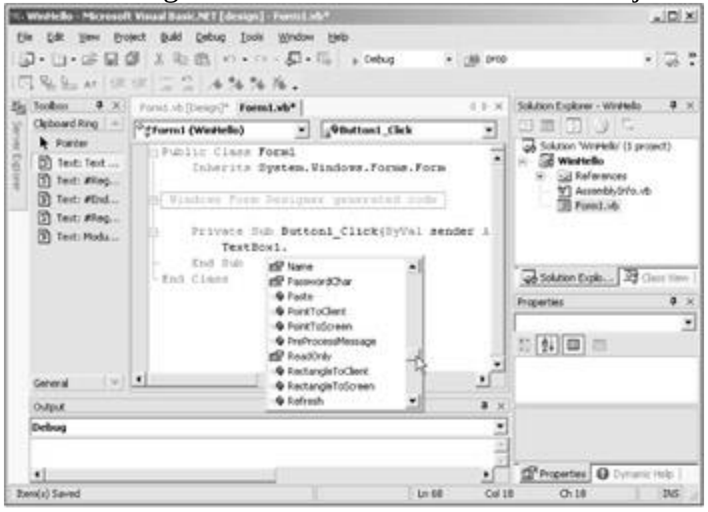
Window The properties window displays the properties for a form or a control. Properties describe attributes such as size, color, and font of a control. Each form or control has its own set of properties. When you click on a control or the form, the properties will be listed in the properties window. There are two columns in the properties window. The first column lists the property names and the second column shows the current value of the property. The value can be changed at the design phase of the form or through the program code. There are lots of properties associated with controls.

Title Bar -It shows the title of the VB.Net project you are currently working on. The default project title is the project name you have specified when you create a new project. If you would like to change the project name or title to other name, you can change it through Project -> Project Properties.

Code editor window (F7) –where we can write the coding of the form or class etc.

IntelliSense

One useful feature of VB .NET code designers is Microsoft's *IntelliSense*. IntelliSense is what's responsible for those boxes that open as you write your code, listing all the possible options and even completing your typing for you. IntelliSense is one of the first things you encounter when you use VB .NET, and you can see an example in Figure 1.25, where I'm looking at all the members of a text box object.


WinHello - Microsoft Visual Basic.NET [design] - Form1.vb*
File Edit View Project Build Debug Tools Window Help
Debug
Toolbox
Clipboard Ring
Pointer
Form1.vb [Design]* Form1.vb*
Form1 (WinHello)
Button1_Click
Solution Explorer - WinHello
Solution 'WinHello' (1 project)
WinHello
References
AssemblyInfo.vb
Form1.vb
Solution Explo... Class View
Properties
General
Output
Debug
Properties Dynamic Help
Item(s) Saved Ln 68 Col 18 Ch 18 INS

```
Public Class Form1
    Inherits System.Windows.Forms.Form

 Windows Form Designer generated code

    Private Sub Button1_Click(ByVal sender A
        TextBox1.
    End Sub
End Class
```

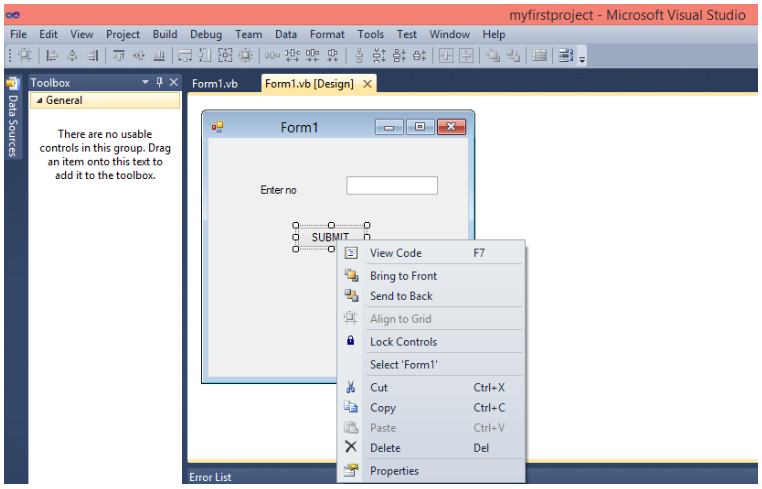
```
Name
PasswordChar
Paste
PointToClient
PointToScreen
PreProcessMessage
ReadOnly
RectangleToClient
RectangleToScreen
Refresh
```

Context Menu

It contains shortcut for frequently performed actions.

To open context menu , select any object and click the right mouse button

Context menu will open according to the selected object


myfirstproject - Microsoft Visual Studio
File Edit View Project Build Debug Team Data Format Tools Test Window Help
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Data Sources
Form1.vb
Form1.vb [Design]
Form1
Enter no
SUBMIT
View Code F7
Bring to Front
Send to Back
Align to Grid
Lock Controls
Select 'Form1'
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Delete Del
Properties
Error List

What is Auto Hide icon?
This is a new feature in Visual Studio that hides away the windows not currently in use.The reference to "not currently *using*" suggests that the windows or panels are not in focus; however, they are not closed down. Thus, as you change windows like going from Solution Explorer to Help the one you are leaving slides closed.

Variable:

A variable is something that is used in a program to store data in memory.

Variable in VB.Net has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory.

The **Dim** statement is used for variable declaration and storage allocation for one or more variables.

Example: **Dim** Counter **as** Integer

Datatype:

The data type of a programming element refers to what kind of data it can hold and how the data is stored.

```
Dim b As Byte   Dim n As IntegerDim c As Char   Dim s As String

b = 1n = 1234567

c = "U"

s = "Me"
```

Visual Basic type	**Common language runtime type structure**	**Nominal storage allocation**	**Value range**
Miscellaneous data type			
Boolean	System.Boolean	2 bytes	True or False.
Byte	System.Byte	1 byte	0 through 255 (unsigned).
Object	System.Object (class)	4 bytes	Any type can be stored in a variable of type Object.
Date	System.DateTime	8 bytes	0:00:00 on January 1, 0001 through 11:59:59 PM on December 31, 9999.
Character data type			
Char	System.Char	2 bytes	0 through 65535 (unsigned).
String (variable-length)	System.String (class)	Depends on implementing platform	0 to approximately 2 billion Unicode characters.
Numeric data type			
Decimal	System.Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal; smallest nonzero number is +/-0.0000000000000000000000000001 (+/-1E-28).
Double (double-precision floating-point)	System.Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324 for negative values; 4.94065645841246544E-324 through 1.79769313486231570E+308 for positive values.
Integer	System.Int32	4 bytes	-2,147,483,648 through 2,147,483,647.
Long (long integer)	System.Int64	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807.
Short	System.Int16	2 bytes	-32,768 through 32,767.
Single (single-precision floating-point)	System.Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values.
User define datatype (structure,enum,array)			
User-Defined Type (structure)	(inherits from System.ValueType)	Depends on implementing platform	Each member of the structure has a range determined by its data type and independent of the ranges of the other members.

Data Type Conversion Function

CBool(expression)- Converts the expression to Boolean data type.

```
Dim a, b, c As Integer

Dim check As Boolean

a = 5

b = 5

' The following line of code sets check to True.

check = CBool(a = b)

c = 0

' The following line of code sets check to False.

check = CBool(c)
```

CByte(expression)- Converts the expression to Byte data type.

```
Dim aDouble As Double

Dim aByte As Byte

aDouble = 125.5678

' The following line of code sets aByte to 126.

aByte = CByte(aDouble)
```

CChar(expression) - Converts the expression to Char data type.

```
Dim aString As String

Dim aChar As Char

' CChar converts only the first character of the string.

aString = "BCD"

' The following line of code sets aChar to "B".

aChar = CChar(aString)
```

CDate(expression) - Converts the expression to Date data type

```
Dim aDateString, aTimeString As String

Dim aDate, aTime As Date

aDateString = "February 12, 1969"

aTimeString = "4:35:47 PM"

' The following line of code sets aDate to a Date value.

aDate = CDate(aDateString)

' The following line of code sets aTime to Date value.

aTime = CDate(aTimeString)
```

CDbl(expression)- Converts the expression to Double data type.

```
Dim aDec As Decimal

Dim aDbl As Double

' The following line of code uses the literal type character D to make aDec a Decimal.

aDec = 234.456784D

' The following line of code sets aDbl to 1.9225456288E+1.

aDbl = CDbl(aDec * 8.2D * 0.01D)
```

CDec(expression)- Converts the expression to Decimal data type.

```
Dim aDouble As Double

Dim aDecimal As Decimal

aDouble = 10000000.0587

' The following line of code sets aDecimal to 10000000.0587.

aDecimal = CDec(aDouble)
```

CInt(expression)- Converts the expression to Integer data type.

```
Dim aDbl As Double

Dim anInt As Integer

aDbl = 2345.5678

' The following line of code sets anInt to 2346.

anInt = CInt(aDbl)
```

CLng(expression)- Converts the expression to Long data type.

```
Dim aDbl1, aDbl2 As Double

Dim aLng1, aLng2 As Long

aDbl1 = 25427.45

aDbl2 = 25427.55

' The following line of code sets aLng1 to 25427.

aLng1 = CLng(aDbl1)

' The following line of code sets aLng2 to 25428.

aLng2 = CLng(aDbl2)
```

CObj(expression)- Converts the expression to Object type.

```
Dim aDouble As Double

Dim anObject As Object

aDouble = 2.7182818284

' The following line of code sets anObject to a pointer to aDouble.

anObject = CObj(aDouble)
```

CShort(expression)- Converts the expression to Short data type.

```
Dim aByte As Byte

Dim aShort As Short

aByte = 100

' The following line of code sets aShort to 100.

aShort = CShort(aByte)
```

CSng(expression)- Converts the expression to Single data type.

```
Dim aDouble1, aDouble2 As Double

Dim aSingle1, aSingle2 As Single

aDouble1 = 75.3421105

aDouble2 = 75.3421567

' The following line of code sets aSingle1 to 75.34211.

aSingle1 = CSng(aDouble1)

' The following line of code sets aSingle2 to 75.34216.

aSingle2 = CSng(aDouble2)
```

CStr(expression)- Converts the expression to String data type.

```
Dim aDouble As Double
Dim aString As String
aDouble = 437.324
' The following line of code sets aString to "437.324".
aString = CStr(aDouble)
```

CTYPE Function

It used to convert one type to another type. Instead of remember all conversion function, remember only CTYPE function.

Syntax: CType(expression,typename)

Example:

```
dim a as integer
dim b as integer

  b=66.7
 a=CType(b,Integer)
```

Operators:

Operators	Description
Arithmetic Operators	+,-,*,/,\,MOD,^
Assignment Operators	=,^=,*=,/=,\=,+=,-=,<<=,>>=,&=
Comparison Operators	>,<,=,<>,>=,<=
Concatenation Operators	+,&
Logical/Bitwise Operators	And,Or,Not,Xor
Bit Shift Operators	>>,<<

Boxing and Unboxing

Boxing and unboxing is an important concept in VB.NET's type system. With Boxing and Unboxing one can link between value-types and reference-types by allowing any value of a value-type to be converted to and from type object.

Boxing

- Boxing is a mechanism in which value type is converted into reference type.
- It is implicit conversion process

Unboxing

- Unboxing is a mechanism in which reference type is converted into value.
- It is explicit conversion process.

Example:

```
DimIasInteger = 10

Dim o asObject = I‘Boxing

Dim J As integer=CInt(o) ‘Unboxing
```

Constant

The **constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals

const s1 as String = “hello”

Comments in vb.net

Rem or **‘** (single quote) can be used to comment the line.

Statements

A **statement** is a complete instruction in Visual Basic programs. It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as −

- **Declaration statements** − these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
- **Executable statements** − these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.

Option Explicit and Option Strict

Option Explicit

- **Option Explicit** statement ensures whether the compiler requires all variables to be explicitly declared or not before it use in the program.
  Option Explicit [On Off]
- The **Option Explicit** has two modes. **On** and **Off** mode.
- when ON , you have to declare all the variable before you use it in the program . If not , it will generate a compile-time error whenever a variable that has not been declared is encountered .
- when OFF , Vb.Net automatically create a variable whenever it sees a variable without proper declaration.

- By default the ***Option Explicit is On***
- With the Option Explicit On , you can reduce the possible errors that result from misspelled variable names.
- Because in Option Explicit On mode you have to declare each variable in the program for storing data.

Option Explicit On	**Option Explicit Off**

Public Class Form1
  Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
      Dim someVariable As String
someVariable = "Option Explicit ON"
MsgBox(someVariable)
  End Sub
End Class
	Option Explicit Off
	
	Public Class Form1
	    Private Sub Button1_Click(ByVal sender As System.Object, _
	ByVal e As System.EventArgs) Handles Button1.Click
	someVariable = "Option Explicit ON"
	MsgBox(someVariable)
	    End Sub
	End Class

Option Strict

- **Option Strict** is prevents program from automatic variable conversions, that is implicit data type conversions .

  Option Strict [On Off]

- By default ***Option Strict is Off***

Public Class Form1
Private Sub Button1_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles Button1.Click
  Dim longNum As Long
  Dim intNum As Integer
longNum = 12345
intNum = longNum
MsgBox(intNum)
End Sub
End Class
	Option Strict On
	
	Public Class Form1
	    Private Sub Button1_Click(ByVal sender As System.Object, _
	ByVal e As System.EventArgs) Handles Button1.Click
	 Dim longNum As Long
	        Dim intNum As Integer
	
	longNum = 12345
	intNum = CInt(longNum)
	
	MsgBox(intNum)
	
	    End Sub
	End Class

Option Strict is Off	***Option Strict is ON***

Option Compare

The **Option Compare** statement controls whether string comparisons are carried out using binary comparisons or text comparisons. If no such statement is specified in a file, the compilation environment controls which type of comparison will be used.

Syntax: option compare {binary | text }

- When Option Compare is not used in a module, the default comparison method is Binary.
- When Option Compare is used, it must appear at the start of the module's declarations section, before any procedures.
- **Binary comparison** the default text comparison method in Visual Basic—uses the internal binary code of each character to determine the sort order of the characters. For example, "A" < "a".
- ***Text comparison*** uses the locale settings of the current system to determine the sort order of the characters. Text comparison is case insensitive. For example, "A" = "a".

<u>Type Checking Function:</u>

VB.NET provides number of data verification or data type checking function as below:

IsDate()

- Returns True if the value of variable is date value; Otherwise, it returns False

```
        DimMyVar, MyCheck

        MyVar = "04/28/2014"' Assign valid date value.
        MyCheck = IsDate(MyVar)    ' Returns True.
        MsgBox(MyCheck)

        MyVar = "April 28, 2014"' Assign valid date value.
        MyCheck = IsDate(MyVar)    ' Returns True.
MsgBox(MyCheck)

        MyVar = "13/32/2014"' Assign invalid date value.
        MyCheck = IsDate(MyVar)    ' Returns False.
        MsgBox(MyCheck)

        MyVar = "04.28.14"' Assign valid time value.
        MyCheck = IsDate(MyVar)    ' Returns True.
MsgBox(MyCheck)

        MyVar = "04.28.2014"' Assign invalid time value.
        MyCheck = IsDate(MyVar)    ' Returns False.
MsgBox(MyCheck)
```

IsNothing()

- Returns True if the object variable that currently has no assigned value; Otherwise, it returns False

```
    Dim objtemp As Object
      Dim bolans As Boolean

bolans = IsNothing(objtemp)
MsgBox(bolans)  ' return true
```

```
objtemp = "dolly"
bolans = IsNothing(objtemp)
MsgBox(bolans)  ' return false
```

IsNumeric()

- Returns True if the value is numeric; Otherwise returns false

```
    DimMyVar, MyCheck
    MyVar = "53"' Assign value.
    MyCheck = IsNumeric(MyVar)    ' Returns True.
MsgBox(MyCheck)

    MyVar = "459.95"' Assign value.
    MyCheck = IsNumeric(MyVar)    ' Returns True.
MsgBox(MyCheck)

    MyVar = "45 Help"' Assign value.
    MyCheck = IsNumeric(MyVar)    ' Returns False.
MsgBox(MyCheck)
```

IsArray()

- Tests whether an object variable points to an array

```
        Dim s() As Integer = {1, 2}
        Dim t As Object
        t = s
    MsgBox(IsArray(t)) ' return true

    Dim strArr() As String
    Console.WriteLine(IsArray(strArr)) ' return false An uninitialized array
```

<u>**Enumerations**</u>

When you are in a situation to have a number of constants that are logically related to each other, you can define them together these constants in an enumerator list. An enumerated type is declared using the enum keyword.

Syntax:

```
Enum enumerationname [ As datatype ]
  memberlist
End Enum
```

Enum declaration:

```
Enumdayaction As Integer

awake = 0
asleep = 1
coding = 2

 End Enum
```

An enumeration has a name, an underlying data type, and a set of members. Each member represents a constant. It is useful when you have a set of values that are functionally significant and fixed.

Retrieve and check the Enum value:

```
    Private action Asdayaction = 2

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
Button1.Click

MessageBox.Show(action.ToString())
    End Sub
```

String Function:

1. The Len Function

The length function is used to find out the number of characters in any given string.

Syntax

Len(*string*)

2. The Mid function

The mid function is used to Return a substring 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. The Left Function

The Left function extract the left portion of a string.

Syntax

Left("string", n)

4. The Right Function

The Right function extract the right portion of a string.

SyntaxRight("string", n)

5. The Space Function

The space function is used to Return a string containing the specified number of blank spaces.

Syntax

Space (*number*)

6. The Replace Function

The replace function is used to replacing some text in a string with some other text.

Syntax

Replace(string, searchtext, replacetext)

7. The Trim function

The trim function trims the empty spaces on both side of the String.

Syntax

Trim ("String")

8. The Ltrim Function

The Ltrim function trims the empty spaces of the left portion of the string.

Syntax

Ltrim("string")

9. The Rtrim Function

The Rtrim function trims the empty spaces of the Right portion of the string.

Syntax

Rtrim("string")

10. The Ucase and the Lcase Functions

The Ucase function converts all the characters of a string to capital letters. On the other hand, the Lcase function converts all the characters of a string to small letters.

Syntax

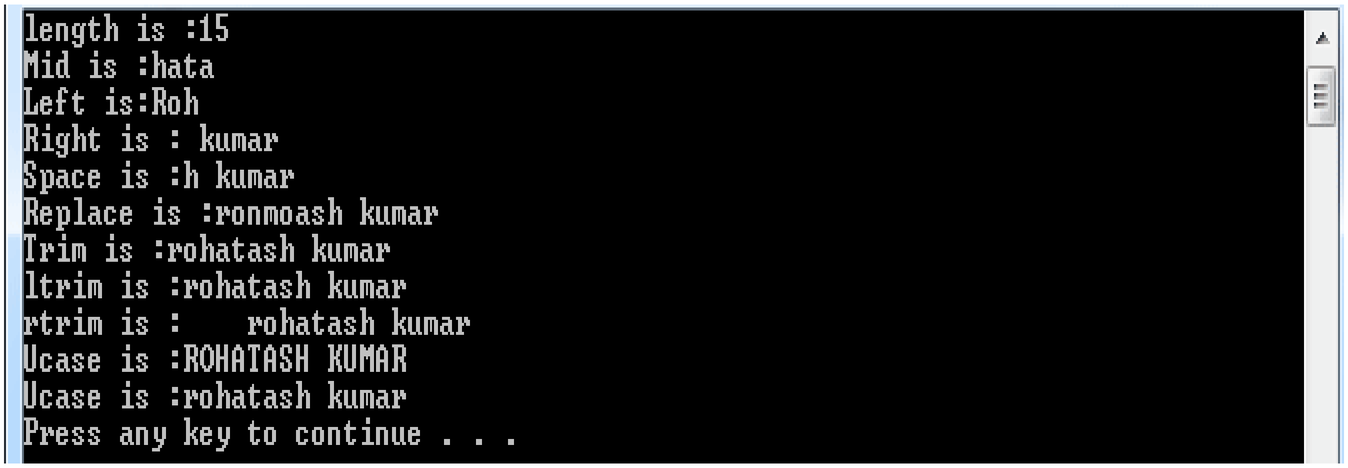
Ucase("string")

Lcase("string")

EXAMPLE

```
Module Module1
    Sub Main()
         Dim leng As String = Len(" Rohatashkumar")
        Console.WriteLine("length is :" &leng)
        Dim middle As String = Mid("Rohatash Kumar", 3, 4)
        Console.WriteLine("Mid is :" & middle)
        Dim leftf As String = Left("Rohatash Kumar", 3)
        Console.WriteLine("Left is:" &leftf)
        Dim rightr As String = Right("rohatashkumar", 6)
        Console.WriteLine("Right is :" &rightr)
        Dim spaces As String = Right("rohatashkumar", 7)
        Console.WriteLine("Space is :" & spaces)
        Dim replaces As String = Replace("rohatashkumar", "hat", "nmo")
        Console.WriteLine("Replace is :" & replaces)
        Dim trimt As String = Trim("   rohatashkumar  ")
        Console.WriteLine("Trim is :" &trimt)
        Dim ltriml As String = LTrim("    rohatashkumar     ")
        Console.WriteLine("ltrim is :" &ltriml)
        Dim rtrimr As String = RTrim("    rohatashkumar     ")
        Console.WriteLine("rtrim is :" &rtrimr)
        Dim ucaseu As String = UCase("rohatashkumar")
        Console.WriteLine("Ucase is :" &ucaseu)
        Dim lcasel As String = LCase("ROHATASH KUMAR")
        Console.WriteLine("Ucase is :" &lcasel)
    End Sub
End Module
```

OUTPUT



```
length is :15
Mid is :hata
Left is:Roh
Right is : kumar
Space is :h kumar
Replace is :ronmoash kumar
Trim is :rohatash kumar
ltrim is :rohatash kumar
rtrim is :    rohatash kumar
Ucase is :ROHATASH KUMAR
Ucase is :rohatash kumar
Press any key to continue . . .
```

Date Time Class

- VB.NET's *DateTime* structure represents an instant in time and is usually expressed as a particular date and time of the day.
- **Properties**
  - Date: returns the date component of the DateTime value.
  - Day: returns the day of the month component of the DateTime value.
  - DayOfWeek: returns the day of the week component of the DateTime value.
  - DayOfYear: returns the day of the year component of the DateTime value.
  - Hour: returns the hour component of the DateTime value.

  - Millisecond: returns the milliseconds component of the DateTime value.
  - Minute: returns the minute component of the DateTime value.
  - Month: returns the month component of the DateTime value.
  - Now: returns a DateTime value that is the current local date and time on this computer.
  - Second: returns the seconds component of the DateTime value.
  - TimeOfDay: returns the time of day of the DateTime value.
  - Today: returns the current system date.
  - Year: returns the year component of the DateTime value.

- **Methods**
  - Add: adds the value of the specified TimeSpan to the DateTime value.
  - AddDays: adds the specified number of days to the DateTime value.
  - AddHours: adds the specified number of hours to the DateTime value.
  - AddMilliseconds: adds the specified number of milliseconds to the DateTime value.
  - AddMinutes: adds the specified number of minutes to the DateTime value.
  - AddMonths: adds the specified number of months to the DateTime value.
  - AddSeconds: adds the specified number of seconds to the DateTime value.
  - AddYears: adds the specified number of years to the DateTime value.
  - DaysInMonth: returns the number of days in the specified month of the specified year.
  - IsLeapYear: returns an indication of whether the specified year is a leap year.
  - Subtract: subtracts the specified time or duration from the DateTime value.
  - ToLocalTime: converts the current Coordinated Universal Time (UTC) to local time.
  - ToLongDateString: converts the value of this instance to its equivalent long date string representation.

- ToLongTimeString: converts the value of this instance to its equivalent long time string representation.
- ToShortDateString: converts the value of this instance to its equivalent short date string representation.
- ToShortTimeString: converts the value of this instance to its equivalent short time string representation.

Date Functions

DateAdd()

It is used to returns a date with a date,time value added with a specified time intreval.

Syntax: DateAdd(interval, number, date)

A date is added after an intreval of 10 days to the current date value.

DateDiff()

It is used to return a long value specifying the number of time intrevals between the specified date values.

Syntax: Datediff(interval,date1,date2)

The time intrevals between two same dates of different years are found using the 'DateDiff()'.

DatePart()

returns an integer value containing the specified component of the Date value.

Syntax: DatePart(interval,date)

the date value entered is in the **mm:dd:yy** format

DateSerial()

returns an date value for the specified year, month and day with the time set to the midnight.
If the month value is '0' or '-1' the month december or november of the previous year is taken.
If the month value is '1', january month of the calculated year is taken, if '13' january of the following year is taken.
If the Day value is '1' refers to the first day of the calculated month, '0' for the last day of previous month, '-1' the penultimate day of the previous month.

Syntax: DateSerial(Year,Month,Day)

DateValue()

returns an date value containing the date information as a string

Syntax: DateValue(Date)

the date information alone is displayed as a String using the **DateValue** function

IsDate()

checks if the given expression is a valid date and returns a boolean true or false.

Syntax: IsDate(Expession)

If the date given as the argument is valid, the **IsDate** function returns **True**.

Today()
Return today’s date
Syntax: Today()

Now()
Return today’s date with time
Syntax: now()

Month()
returns the month of the year as an integer value in the range of 1-12.
Syntax: Month(Date)
the month for the given date is returned using the Month function.

TimeSerial()
returns an date value with the specified hour, minute, second with the date information is set to January 1 of year 1.
Syntax: Timeserial(hour, minute, second)

Format()

Returns a string formatted according to instructions contained in a format string expression.

Syntax: format(date,format)

<u>Example:</u>

DateAdd

```
MsgBox("10 Days after the current date is::" &DateAdd(DateInterval.Day, 10, Now))
```

DateDiff

```
    Dim d1 As Date = #2/4/2009#
    Dim d2 As Date = #2/4/2010#
    Dim res As Long
    res = DateDiff(DateInterval.Day, d1, d2)
MsgBox("The number of time intrevals between the dates is::" & res)
```

DatePart

```
MsgBox("The date part of '01/10/2010' is::" &DatePart("d", "01/10/2010"))
```

DateSerial

```
    Dim a As Date
    a = DateSerial(2010, 2, 21)
MsgBox(a)
```

DateValue

```
MsgBox("Date information as a String is::"&DateValue("5/10/2010 12:00:01 AM"))
```

TimeSerial

```
MsgBox("Time displayed using Timeserial() is:: " &TimeSerial(4, 30, 23))
```

isdate

```
        Dim curdat As Date
curdat = "5/31/2010"
MsgBox("Is '5/31/2010' a valid date::" &IsDate(curdat))
```

Today

```
        Dim curdat As Date
curdat = Today()
MsgBox(curdat)
```

now

```
        Dim curdat As Date
curdat = Now()
MsgBox(curdat)
```

Month

```
        Dim dat As Date
dat = "6/23/2010"
MsgBox("Month value of the given date is::" & Month(dat))
```

Format

```
MsgBox(Format(Now, "M-d-yy"))
MsgBox(Format(Now, "MM-dd-yyyy"))
MsgBox(Format(Now, "MMMM-d-yyy-  dddd"))
MsgBox(Format(Now, "hh:mm:sstt"))
```

Design time and Run time

- The time during which you build an application in the development environment by adding controls, setting controls or form properties, and so on. For ex. Setting the password char property to textbox (*) design time
- The time, during which code is running. during run time, you can't edit the code

With…End

Executes a series of statements that repeatedly refer to a single object so that the statements can use a simplified syntax when accessing members of the object.

To make this type more efficient and easier to read, we use this block.

The use of it do not require calling again and again the name of the object

It allows us to set multiple properties and methods quickly and easily

Remember this is not a type of loop

Syntax	**Example (on button click)**
With object name [statements] End With	With Button1 .Text = "Click ME" .ForeColor = Color.Aqua .BackColor = Color.Yellow .Height = 50 .Width = 100 EndWith

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 −

- Sub procedures or Subs
- Functions

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

1. Sub procedures or Subs or Sub Routine

- A Sub procedure is a series of Visual Basic statements enclosed by the Sub and End Substatements. The Sub procedure performs a task and then returns control to the calling code, but it does not return a value to the calling code.
- Each time the procedure is called, its statements are executed, starting with the first executable statement after the Sub statement and ending with the first End Sub, Exit Sub, or Returnstatement encountered.
- You can define a Sub procedure in modules, classes, and structures. By default, it is Public, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it

  A Sub procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

The **Sub** statement is used to declare the name, parameter and the body of a sub procedure. The syntax for the Sub statement is −

```
[Modifiers] Sub SubName [(ParameterList)]
  [Statements]
End Sub
```

Where,

- ***Modifiers*** − specify the access level of the procedure; possible values are - Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***SubName*** − indicates the name of the Sub
- ***ParameterList*** − specifies the list of the parameters

```
SubtellOperator(ByVal task AsString)
Dim stamp AsDate
     stamp = TimeOfDay()
MsgBox("Starting "& task &" at "&CStr(stamp))
EndSub
```

```
tellOperator("file update")
```

2. Functions

- A Function procedure is a series of Visual Basic statements enclosed by the Function and End Function statements. The Function procedure performs a task and then returns control to the calling code. When it returns control, it also returns a value to the calling code.
- Each time the procedure is called, its statements run, starting with the first executable statement after the Function statement and ending with the first End Function, Exit Function, or Returnstatement encountered.
- You can define a Function procedure in a module, class, or structure. It is Public by default, which means you can call it from anywhere in your application that has access to the module, class, or structure in which you defined it.
- A Function procedure can take arguments, such as constants, variables, or expressions, which are passed to it by the calling code.

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

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
  [Statements]
End Function
```

Where,

- ***Modifiers*** − specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***FunctionName*** − indicates the name of the function

- ***ParameterList*** − specifies the list of the parameters
- ***ReturnType*** − specifies the data type of the variable the function returns

FunctionFindMax(ByVal num1 AsInteger, ByVal num2 AsInteger) AsInteger ' local variable declaration */ Dim result AsInteger If (num1 > num2) Then result = num1 Else result = num2 EndIf FindMax = result EndFunction	**Calling function** MsgBox("the maximum number is "&FindMax(10, 5))

Optional argument

Sub notify(ByVal company AsString, OptionalByValdesgAsString = "manager") Ifdesg = "manager"Then MsgBox("i am manager") Else MsgBox("i am not manager") EndIf EndSub	Call notify("abc") Call notify("abc", "worker")

Function overloading

Function overloading is where two or more functions can have the same name but different parameters.

Function overloading can be considered as an example of compile time polymorphism feature in Oop’s.

PublicOverloadsFunction add(ByVal a AsInteger, ByVal b AsInteger)
MsgBox("You are in function add(a,b)")
Return a + b
EndFunction
 	
PublicOverloadsFunction add(ByVal a AsInteger, ByVal b AsInteger, ByVal c AsInteger)
MsgBox("You are in function add(a, b, c)")
Return a + b + c
EndFunction
	
	MsgBox(add(4, 2))
	
	MsgBox(add(4, 5, 1))

ByVal&ByRef Methods for passing arguments to Subroutine & Functions

- The **ByVal** keyword indicates that an argument is passed in such a way that the called procedure or property cannot change the value of a variable passed as the argument in the calling code.
- The **ByRef** keyword indicates that an argument is passed in such a way that the called procedure can change the value of a variable passed as the argument in the calling code.

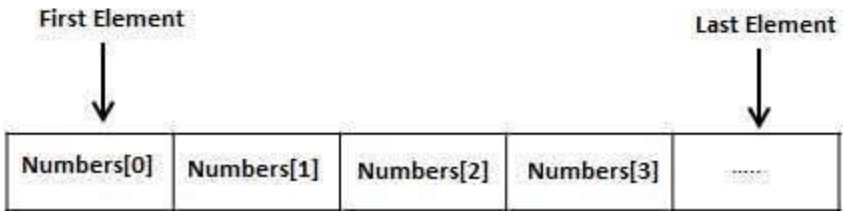
Dim value AsInteger = 1	
Pass by value Sub Example1(ByVal test AsInteger) test = 10 EndSub	**Pass by reference** Sub Example2(ByRef test AsInteger) test = 10 EndSub

' The integer value doesn't change here when passed ByVal.
 	
    Example1(value)
MsgBox("by val "& value)
	' The integer value DOES change when passed ByRef.
	
	    Example2(value)
	MsgBox("by ref "& value)

Output : 1	Output :10

Arrays in VB.Net

- **Declaration**
- An Array is a collection of values of similar data type.
- Technically, VB.Net arrays are of reference type.
- Each array in VB.Net is an object and is inherited from the System.Array class.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



- Arrays are declared as follows:

  Dim <identifier>(<size of array>) As <data type>

- Lets define an array of Integer type to hold 10 integers.

  Dim myIntegers(9) As Integer

- The above will create an array of 10 integers from the index of 0 to 9.
- The size of an array is fixed and must be defined before use.
- You can also use variables to define the size of array like so:

```
Dim size As Integer = 10

Dim myIntegers(10-1) As Integer
```

- Accessing the values stored in array
- To access the values in an Array, we use the indexing operator (Integer index) by passing an Integer to indicate which particular index value we wish to access.

DimmyIntArray() AsInteger
myIntArray = NewInteger() {5, 10, 15, 20}
Dim j AsInteger = myIntArray(2)
MsgBox(j)
	Output=2

Dim n(10) AsInteger' n is an array of 11 integers ' Dimi, j AsInteger ' initialize elements of array n ' Fori = 0 To 10 n(i) = i + 100 ' set element at location i to i + 100 Nexti ' output each array element's value ' For j = 0 To 10 MsgBox("Element({0} "& j) MsgBox("Element({1} "& n(j)) Next j	Element(0) = 100 Element(1) = 101 Element(2) = 102 Element(3) = 103 Element(4) = 104 Element(5) = 105 Element(6) = 106 Element(7) = 107 Element(8) = 108 Element(9) = 109 Element(10) = 110 output

Multidimensional Arrays

- A multidimensional array is an 'array of arrays'.
- A multidimensional array is the one in which each element of the array is an array itself.
- It is similar to tables of a database where each primary element (row) is the collection of other secondary elements (columns).
- Suppose we wish to create a two dimensional rectangular array with 2 rows and 3 columns. We can instantiate the array as follows.
- You can declare a 2-dimensional array of strings as −

- Dim twoDStringArray(10, 20) As String

or, a 3-dimensional array of Integer variables −

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

DimstrBooks(4, 1) AsString strBooks(0, 0) = "Learning Visual Basic" strBooks(0, 1) = "John Smith" strBooks(1, 0) = "Visual Basic in 1 Week" strBooks(1, 1) = "Bill White"	Learning Visual Basic John Smith Visual Basic in 1 Week Bill White Everything about Visual Basic Mary Green

```
strBooks(2, 0) = "Everything about Visual Basic"
strBooks(2, 1) = "Mary Green"
strBooks(3, 0) = "Programming Made Easy"
strBooks(3, 1) = "Mark Wilson"
strBooks(4, 0) = "Visual Basic 101"
strBooks(4, 1) = "Alan Woods"

For intCount1 = 0 To 4
For intCount2 = 0 To 1
MessageBox.Show(strBooks(intCount1, intCount2))
Next intCount2
Next intCount1
```

```
Programming Made Easy
Mark Wilson
Visual Basic 101
Alan Woods
```

output

Dynamic Array

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 for ReDim statement −

```
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.
- **subscripts** specifies the new dimension.

```
    Dim marks() AsInteger
        ReDim marks(2)
            marks(0) = 85
            marks(1) = 75
            marks(2) = 90
        ReDimPreserve marks(10)
            marks(3) = 80
         marks(4) = 76
            marks(5) = 92
         marks(6) = 99
            marks(7) = 79
         marks(8) = 75
        Fori = 0 To 10
        MsgBox(i&vbTab& marks(i))
        Nexti
```

```
0       85
1       75
2       90
3       80
4       76
5       92
6       99
7       79
8       75
9       0
10      0
```

Output

sr.No	Property Of Array(Name & Description)
1	**IsFixedSize** Gets a value indicating whether the Array has a fixed size.
2	**IsReadOnly** Gets a value indicating whether the Array is read-only.
3	**Length** Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	**LongLength** Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	**Rank** Gets the rank (number of dimensions) of the Array.
Sr.No	**Method Of Array (Name & Description)**
1	**Public Shared Sub Clear (array As Array, index As Integer, length As Integer)** Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	**Public Shared Sub Copy (sourceArray As Array, destinationArray As Array, length As Integer)** Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.

3	**Public Function GetLength (dimension As Integer) As Integer** Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
4	**Public Function GetLongLength (dimension As Integer) As Long** Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
5	**Public Function GetLowerBound (dimension As Integer) As Integer** Gets the lower bound of the specified dimension in the Array.
6	**Public Function GetType As Type** Gets the Type of the current instance (Inherited from Object).
7	**Public Function GetUpperBound (dimension As Integer) As Integer** Gets the upper bound of the specified dimension in the Array.
8	**Public Function GetValue (index As Integer) As Object** Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
9	**Public Shared Function IndexOf (array As Array,value As Object) As Integer** Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
10	**Public Shared Sub Reverse (array As Array)** Reverses the sequence of the elements in the entire one-dimensional Array.

Conditional statement

Conditional statement

Visual Basic allows you to test conditions and perform different operations depending on the results of that test. You can test for a condition being true or false, for various values of an expression, or for various exceptions generated when you execute a series of statements. The decision statements supported by Visual Basic include:

- **If...Then**

if *condition* then
 	
Statement1
 	
Statement2
end if
	Dim x AsInteger
	x = 10
	If (x Mod 2) = 0 Then
	MsgBox("x is an even number")
	
	EndIf

- **If...Then...Else**

if *condition* then *Statement1* *statement2* else *Statement3* *statement4* end if	Dim x AsInteger x = 10 If (x Mod 2) = 0 Then MsgBox("x is an even number") Else MsgBox("x is an odd number") EndIf

- **If...Then...ElseIf…Then..Else**

If *condition1* then *Statements…* Else If *condition*2 then *Statements…* Else *Statements…* End If	Dim x AsInteger x = InputBox(x) If x > 0 Then MsgBox("It is positive.") ElseIf x < 0 Then MsgBox("It is negative.") Else MsgBox("It is zeo.") EndIf

- **Select...Case**

Select case variable case val1 statements case val2 statements case val3 statements ………… case else statements End select	Dim x AsInteger x = InputBox("Please enter your number from 1 to 3", x) SelectCase x Case 1 MsgBox("You entered 1", x) Case 2 MsgBox("You entered 2") Case 3 MsgBox("You entered 3") Case Else MsgBox("Invalid!") EndSelect

Loop Constructs

Loop structures allow you to execute one or more lines of code repetitively. You can repeat the statements until a condition is true, until a condition is false, a specified number of times, or once for each object in a collection. The loop structures supported by Visual Basic include:

- **While**

While condition [statements] [Continue While] [statements] [Exit While] [statements] End While	Dim index AsInteger = 0 While index <= 10 MsgBox(index.ToString&" ") index += 1 EndWhile ' Output: 0 1 2 3 4 5 6 7 8 9 10

- **Do...Loop**

Do [statements] [Continue Do] [statements] [Exit Do] [statements] Loop { While | Until } condition	Dim index AsInteger = 0 Do MsgBox(index.ToString&" ") index += 1 LoopUntil index > 10 ' Output: 0 1 2 3 4 5 6 7 8 9 10

- **For...Next**

```
For counter [ As datatype ] = start To end [ Step step ]

         [ statements ]
         [ Continue For ]
         [ statements ]
         [ Exit For ]
         [ statements ]
      Next [ counter ]
```

```
For index AsInteger = 1 To 5
MsgBox(index.ToString&" ")
Next
' Output: 1 2 3 4 5
```

- **For Each...Next**

```
For Each element [ As datatype ] In group

         [ statements ]
         [ Continue For ]
         [ statements ]
         [ Exit For ]
         [ statements ]

Next [ element ]
```

```
' Create a list of strings by using a
' collection initializer.

DimlstAsNewList(OfString) _
From {"abc", "def", "ghi"}

' Iterate through the list.

ForEach item AsStringInlst
MsgBox(item &" ")
Next
'Output: abcdefghi
```

<u>What is an Exit Statement</u>

The **Exit** statement allows you to exit directly from any decision structure, loop, or procedure. It immediately transfers execution to the statement following the last control statement. The syntax for the **Exit** statement specifies which type of control statement you are transferring out of. The following versions of the **Exit** statement are possible:

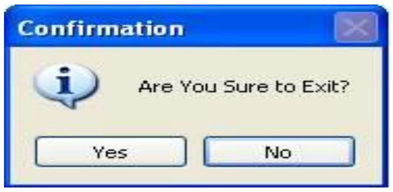
- **Exit Select**
- **Exit Try**
- **Exit Do**
- **Exit While**
- **Exit For**

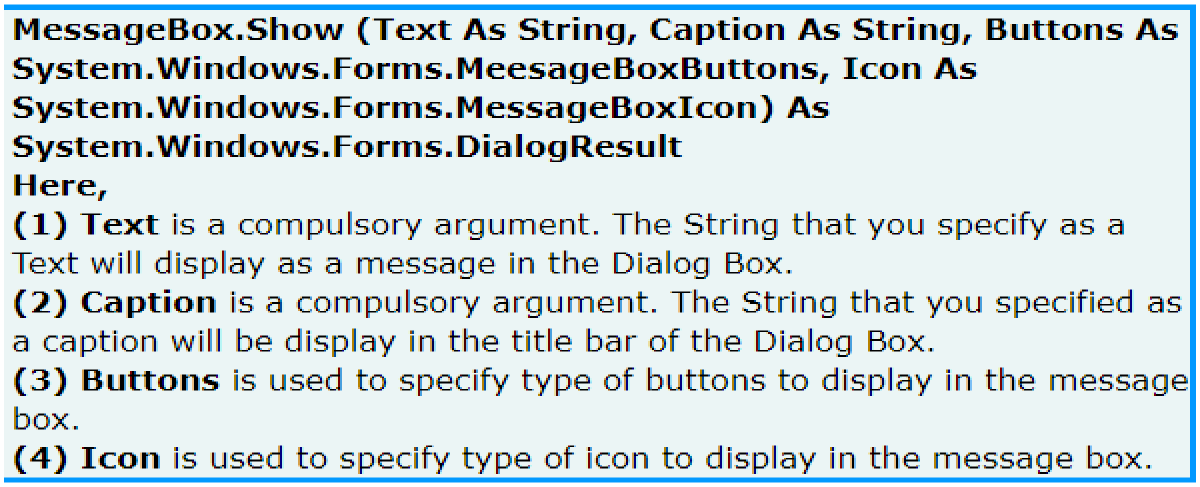
You can also exit directly from a **Function**, **Sub**, or **Property** procedure; the syntax is similar to that of **Exit For** and **Exit Do**:

- **Exit Sub**
- **Exit Function**
- **Exit Property**

Message Box

The show method of MessageBox is used to display User Specific message in a Dialog Box and waits for the user to click a button.It returns an integer value indicating which button is click by user.
MessageBox is shown in the figure below:





MessageBox.Show (Text As String, Caption As String, Buttons As System.Windows.Forms.MeesageBoxButtons, Icon As System.Windows.Forms.MessageBoxIcon) As System.Windows.Forms.DialogResult
Here,
(1) Text is a compulsory argument. The String that you specify as a Text will display as a message in the Dialog Box.
(2) Caption is a compulsory argument. The String that you specified as a caption will be display in the title bar of the Dialog Box.
(3) Buttons is used to specify type of buttons to display in the message box.
(4) Icon is used to specify type of icon to display in the message box.

Possible values for Button argument are

MessageBoxButtons.OKOnly	Display OK Button.
MessageBoxButtons.OKCancel	Display OK and Cancel Button.
MessageBoxButtons.AbortRetryIgnore	Display Abort, Retry and Ignore Button.
MessageBoxButtons.YesNoCancel	Display Yes, No and Cancel Button.

MessageBoxButtons.YesNo	Display Yes and No Button.
MessageBoxButtons.CancelRetry	Display Cancel and Retry Button.

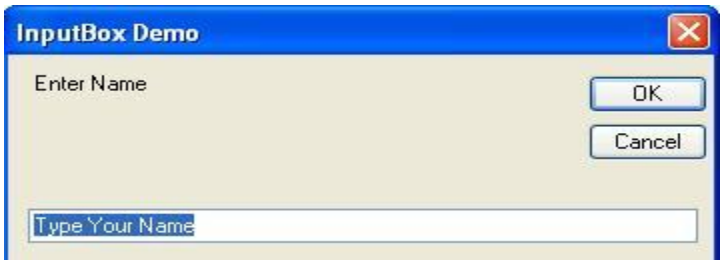
Possible values for Icon argument are:

MessageBoxIcon.Critical	Display Critical icon.
MessageBoxIcon.Question	Display Question icon.
MessageBoxIcon.Exclamation	Display Exclamation icon.
MessageBoxIcon.Information	Display Information icon.

InputBox

InputBox function display a prompt in the form of dialog box as shown below and waits for the user to input some value in textbox or click a button.

If user clicks on OK button then it will return content of textbox in the form of string.
If user clicks on Cancel Button then it will return a blank value.



InputBox (Prompt As String,[Title As String=""], [DefaultResponse As String=""], [XPos As Integer = -1], [YPos As integer = -1]) As String

Here,

(1) Prompt is a compulsory argument. The String that you specify as a Prompt will display as a message in the

InputBox Dialog.

(2) Title is an optional Argument. The String that you specify as a Title will display in the title bar of the InputBox. If you skip this argument then name of the application will display in the title bar.

(3) DefaultResponse is an Optional Argument. The String that you specify as a DefaultResponse will display as a default value in the textbox of the InputBox. If you skip this argument then Textbox of the InputBox is displayed empty.

(4) XPos is an Optional Argument. It Specify the distance (in pixel) of the left edge of the Input box from the left edge of the screen.

(5) YPos is an Optional Argument. It Specify the distance (in pixel) of the upper edge of the Input box from the top edge of the screen.

Note: If you skip XPos and YPos then InputBox will display in the centre of the screen.

What's difference between MsgBox() function and MessageBox.Show() in VB.net?

MsgBox() displays normal messagebox, but when using MessageBox.Show(), it displays a messagebox with Windows Classic Title Bar

Collections

- VB.NET Collections are data structures that holds data in different ways for flexible operations .
- The important datastructres in the Colletions are ArrayList ,HashTable, Sorted list,Stack and Queue etc.
- The main advantages of collections are:

1 .differentdatatype can be under one collection

2 .individual data can be deleted from the collection

3. Collection provides the features of searching,sorting and adding the data as and when needed

ArrayList

- is one of the most flixible data structure from.
- contains a simple list of values.
- easily we can add , insert , delete , view etc..
- It represents ordered collection of an object that can be **indexed** individually.
- very flexible because it grow dynamically and also shrink .
- Commonly used methods are:
  - Add(item) : add elements in arraylis
  - Insert(index, item) : add element to particular index
  - Remove(item) : remove item from array
  - RemoveAt(item_index) : remove item from particular index
  - Sort () : - sort elements in ascending order
  - Copy() : - copy one arraylist into an another arraylist

- Binary search(item) : search data using binary search technique so needed only sorted data
- Indexof(item) : search the first occurance of data in given array list. Returns position
- LastIndexOf (item):search the last occurance of data in given array list. Returns position

```
DimarAsNewArrayList

ar.Add(10)
ar.Add(3)
ar.Add(15)
ar.Add(4)

ar.Sort()
MsgBox(ar.BinarySearch(15))     →3
MsgBox(ar.IndexOf(4))           →1
ar.Insert(2, 8)
MsgBox(ar.Contains(8))          →true
```

HashTable

- stores a (Key, Value) pair type collection of data .
- keys are unique and value can be of any datatype.
- We can retrive items from hashTable to provide the key .
- Both key and value are Objects.
- Commonly used method are:
  - Add(key, Value) : adds keys and value to hash table
  - Item(key) : gives value of the specified key from hash table
  - Contains(key): check wheather the given key is in hash table or not
  - Remove(key) : remove the specified key from hash table
  - Keys() : retrives all the key from the hash table
  - Clear() : remove all the element from the hash table
  - Count() : returns the number of element in the hash table
  - Values () : retrives all the value from hash table

```
DimhtAsNewHashtable
ht.Add("amit", 70)
ht.Add("priya", 80)
ht.Add("chintan", 90)
ht.Add("madhu", 90)

MsgBox(ht.Count)                →4
MsgBox(ht.Item("amit"))         →70
ht.Remove("priya")
MsgBox(ht.Contains("madhu"))→true
ForEach t Inht.Keys
MsgBox(t &ht.Item(t) &ht(t))
Next
ForEach t Inht.Values
MsgBox(t)
Next
```

Sorted List

- It uses a **key** as well as an **index** to access the items in a list.
- A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable. The collection of items is always sorted by the key value.
  - Add(key, Value) : adds keys and value to sorted list. Key must be unique
  - Containskey(key): check wheather the given key is in sorted list or not
  - Getkeylist() : retrives all the key from the sorted ist
  - Clear() : remove all the element from the sorted list
  - Count() : returns the number of element in the sorted list

```
Dim s AsNewSortedList
s.Add("red", 10)
s.Add("green", 5)
s.Add("blue", 5)
s.Add("black", 4)

ForEach x Ins.GetKeyList()
MsgBox(x)                    → black,blue,green,red
Next

MsgBox(s.ContainsKey("blue"))        →true
s.Clear()

ForEach x Ins.GetKeyList()
MsgBox(x)
Next
```