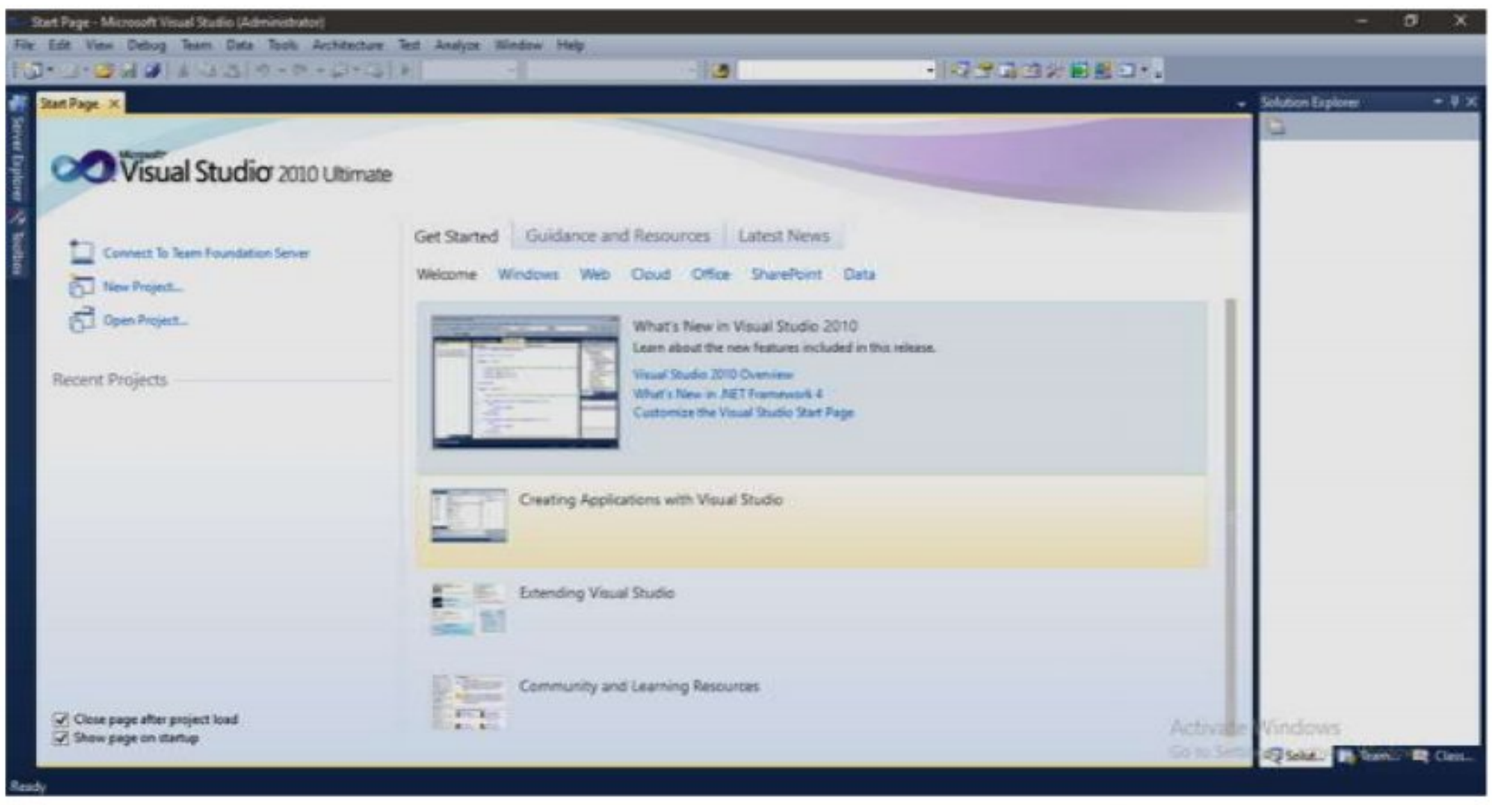
Unit 2. Programming in Visual Basic .NET

2.1.Visual Studio .NET IDE

Before going to the .NET programming, let us learn about the Integrated Development Enviroment(IDE) which helps in programming. We will study about the IDE with respect to Visual Studio 2010.

Start up Screen:

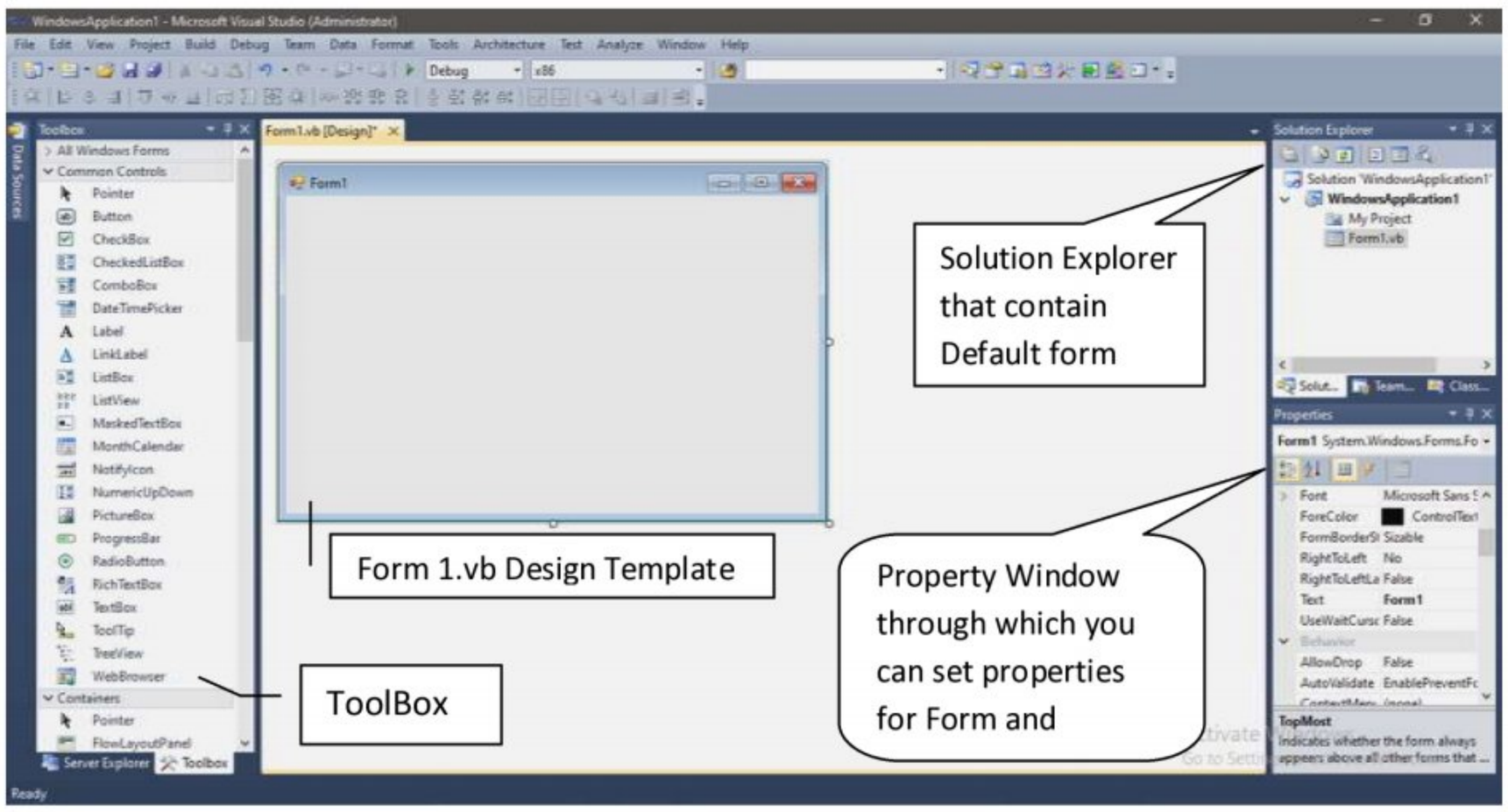


In the above screen we are able to see that it is divided into different panels:

- **Recent Projects**: It displays the list of the projects which we have created recently. In this panel, it also provides link to open existing project which is not in the list or we can create the new project.
- **Getting started**: It displays the link to the topic which provides the help for the Visual Studio 2010.
- **Visual Studio News**: It displays the links for the latest news for the Visual Studio from the internet.

Visual Studio 2010 IDE also contains some tabs such as Server Explorer, Toolbox,Solution Explorer, Class View, Error List, and Output.

- **Visual Studio .NET IDE Screen**



User Interface element of Visual studio .NET IDE

Default Form:

- When you create a new Windows Project in VB.NET, a Form is automatically added to the Project.
- A Form is the basic unit in any Windows-based application and is used to accept user input and perform some action based on the input.
- The Form designer allows you to design the user interface for a Project.
- It allows you to add controls to a Form, arrange them as per your requirements, and add code to perform some action.

The Solution Explorer Window:

- The solution explorer window lists the solution name, Project name, and all the Forms that are used in the Project (see in above figure).
- You can open a particular file existing in a Project by double clicking on it in the solution explorer window.

The Property Window:

- The properties window displays the properties that are associated with an object (see in above figure).
- For example, on selecting Form from the Designer window, the property window list, all the properties associated with Form, such as Name, Text, BackColor, etc.

Toolbox

- The toolbox display number of tabs such as Common, Data, Containers, Menus & Toolbars, and Dialogs etc.
- Each tab contains several items, as shown in above figure.
- At a time, the items of only a single tab are visible.
- The Common tab contains items, such as pointer, label, Textbox, and Button etc.

2.2. Variables

- A variable is the data name given to a memory location that holds value during program execution.
- Value of variable may change during the program execution.

  Variable naming conventions

  Variable name follow certain rules:

  1. First character must be a letter
  2. It can contains digits and alphabets
  3. No special character allowed except underscore.

Example:
(1) Valid variable names: temp, emp_name, book1.
(2) In valid variable name: emp&name, emp no.

Variable Declaration

Dim keyword is used to declare variable and data type of variable is defined by **As** clause.
The general syntax for variable declaration is as follow:
Syntax:
{Dim | Public | Private | Friend | Protected Friend | Static}
Syntax: variable_names [As datatype [= expression]]

Examples

```
Dim intl As Integer          'declared variable of type Integer
Public dt1  As Date          'declared public variable of type Date
Dim name As String           'declared variable of type String
Dim sngl As Single           'declared variable of type Single
Dim x, y As Integer          'declare variable x and y of type Integer
```

Moreover, user can assign value to the variable as follow:

```
intl = 5                     'assign Integer value
name = " visual studio"      'assign String value.
dtl = #01/01/2013#           'assign Date value.
sngl = 5.4                   'assign Single value
```

A user can also initialize variable at the time of declaration as follow:

```
Dim x As Integer = 5                          'assign value 5 to x
Dim name As String = "visual studio"          'assign string to name.
```

If variable is declared without **As** clause, then it will be considered as object type variable.

Examples

```
Dim obj1          'declare variable of type object (default datatype)
```

Scope

The following table shows the list of access modifiers used in VB.NET.

Access modifier	Scope
Private	It is accessible in the module, class or structure in which they are declare.
Public	It is accessible in all procedures of all classes, modules and structures in application
Static	Retains their values within the scope of the method in which they are declared.
Protected	It is accessible in the class in which they are declared or derived class
Friend	It is accessible in any class or module within the assembly
Shared	Shared members such as properties, procedures or fields that are shared by all instances of a class.

Both **Dim** and **private** are considered as private modifier.

❖ Constants

- Constant is an entity whose value never changes during program execution.
- **Cons t**keyword is used to declare a constant.
- Constants must be initialized at the time of declaration.
- The general syntax for constant declaration is as follow:

Syntax:
[{Public | Private | Friend | Protected Friend}] **Const** constant_name [As datatype] = expression

Examples

```
Const pi As Single = 3.14
Const Val = 500       'declared constant of type object
```

2.2 Datatype

- The data type of a programming element refers to what kind of data it canhold and how that data is stored.
- There are mainly four categories of it.

  Numeric Data Types (Short, Integer, Long, Single, Double, Decimal)
  Character Data Types (Char, String)
  Miscellaneous Data Types (Boolean, Byte, Date, Object)
  User-Defined Data Types (Structure, Array, Enum)

- **Numeric Data Types**

Short

- It occupies 2 bytes.
- The range of it -32,768 to 32,767.
- Prefix we can use 'sht' and store only whole number
- Example: Dim shtRollNo As Short
  shtRollNo=5

Integer

- It occupies 4 bytes.
- The range of it -2,147,483,648 to 2,147,483,647
- Prefix we can use 'int' and store only whole number
- Example: Dim intRollNo As Integer
  intRollNo=5

Long

- It occupies 8 bytes.
- The range of it -9,223,372,036,854,775,808 to 9, 223, 372, 036, 854, 775,807.
- Prefix we can use 'lng' and store only whole number
- Example: Dim lngBankIncome As Long
  lngBankIncome = 50000000

Single

- It occupies 4 bytes.
- The range of it -3.4028235E+38 to -1.401298E-45 for negative values;
- 1.401298E-45 to 3.4028235E+38 for positive values
- Prefix we can use 'sng' and store whole and fractional both
- Example: Dim sngSal As Single
  sngSal= 5000.5555

Double

- It occupies 8 bytes.
- The range of it -1.79769313486231570E+308 to -4.94065645841246544E-324 for negative values; 4.94065645841246544E-324 to 1.79769313486231570E+308 for positive values.
- Prefix we can use 'dbl' and store whole and fractional both
- Example: Dim dblSatyamFraud As Double
  dblSatyamFraud = 2222222222222.5557

Decimal

- It occupies 16 bytes.
- The range of it 0 to +/-79,228,162,514,264,337,593,543,950,335 with nodecimal point;
- 0 to +/-7.9228162514264337593543950335 with 28 placesto the right of the decimal; smallest nonzero number is +/- 0.0000000000000000000000000001 (+/-1E-28).
- Prefix we can use 'dec' and store whole and fractional both
- Example: Dim decIncome As Decimal
  decIncome=1.2321334433243232E+19

- **Character Data Types**

Char

- It occupies 2 bytes.
- The range of it 0 to 65535.
- Prefix we can use 'ch' and we need to write (" ").
- Example: Dim chGen As Char
  chGen = "M"

String

- It occupies 2 bytes.
- The range of it 0 to approximately 2 billion Unicode characters.
- Prefix we can use 'str' and we need to write (“ ”).
- Example: Dim strAdd As String
  strAdd = "30, Sundarvan Row House, Adajan, Surat”

- **Miscellaneous Data Types**

Boolean

- It occupies 2 bytes.
- It can store either "True" or "False"
- Prefix we can use 'bin'
- Example: Dim binGen As Boolean
  binGen = True

Byte

- It occupies 1 byte.
- The range of it 0 to 255.
- Prefix we can use ‘byt'
- Usually Byte data type is used to access binary files, image and sound files etc.
- Example: Dim bytNo As Byte
  bytNo=223

Date

- It occupies 8 bytes.
- Minimum value is 0:00:00 on January 1, 0001 and Maximum value11:59:59 PM on December 31, 9999.
- Prefix we can use 'dt' and we need to write (# #)
- The format of Date variable is #mm/dd/yyyy#
- Example: Dim dtDob As Date
  dtDob = #12/21/1982#

Object

- It occupies 4 bytes.
- It can store any type of data. Prefix we can use 'obj'.
- Example:
  Dim objTemp As Objects
  objTemp = 5
  objTemp = "Ram"
  objTemp = # 10/17/2008#

- Use Object at compile time when you do not know what data type thevariable might point to.
- Whatever data type it refers to, an Object variable does not contain thedata value itself, but rather a pointer to the value.
- The Object data type is slower than using explicit data types. Butwhy?
- When we create any variable it stores data into that variable only.
- For example, if we create Dim A As Integer=5
- In memory box is created the name of the box is A, the address of thatbox is XYZ and it contains value 5
- But when we create Dim objA As Object=5

5
XYZ
A

- So when we get or set the value of variable A, it directly deals with thatmemory box only.
- In memory, 2 boxes are created 1st one is A it stores the address of 2nd boxwhich contains value 5
- Object data type is slower than other data type .It takes more time tofetch and store the data, because it first has to point other memory blockand then perform operation.
- The Object data type requires more memory than other data types.

2.2.1 Boxing and Unboxing

- 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**
  The conversion of a value type instance to an object,
- **Unboxing**
  The conversion of an object instance to a value type.

Example

```
Dim no As Integer=10
Dim obj As Object = no            ' boxing
Dim ans As Integer = CInt(obj)    ' unboxing
```

2.2.2 Enumerations

Enumeration is a user-defined data type used to define a related set of constants as a list using the Keyword **enum** statement.

Syntax:

```
Enum   Enumenumeration_name [As Data Type]
Member 1    [-initial expression]
MemberN    [-initial expression]
End Enum
```

- If you do not use initial expression then memberl gets initial value 0, member2 gets initial value 1 and so on.
- Even you can also partially assign the initial value to the members of Enum.

Example

```
Enum  FilePermission
        Create=1
        Read=2
        Write=3
        Delete=4
End Enum

Enum  SecurityLevel
        IllegalEntry=-1
        MinimumSecurity=0
        MaximumSecurity=1
End Enum
Enum abc
        A
        B
        C
End Enum
Sub Main()
        Console.Write(FilePermission.Create)
        Console.Write(SecurityLevel.IllegalEntry)
        Console.Write(abc.B)
        Console.ReadKey()
End Sub
```

2.2.3 Data Type Conversion Functions

- Type conversion is used for convert one data type to another.
- There are 2 type of conversion: Implicit and Explicit
  - **An implicit (Widening) conversion** : does not require any special syntax in the source code.

    Dim A As Integer
    Dim B As Double
    A = 4499
    B = A
  - As above, The VB.NET will convert smaller data type to larger (Integer -> Double) automatically is also called as **Narrowing** to **Widening** conversion
  - **An explicit conversion** requires function.

The following table shows the available conversion functions used for explicit conversion.

Functionname	Converts into
CBool	Boolean
CByte	Byte
CChar	Char
CDate	Date
CDbl or Val	Double
CDec	Decimal
CInt	Integer
CLng	Long
Cobj	Object
CShort	Short
CSng	Single
CStr	String

Syntax: Function Name (argument)
Example:

```
Dim str As String
Dim no As Integer
str = "5"
no = CInt(str)
```

```
Dim str As String
Dim dt As Date
str = "12/11/1982"
dt = CDate(str)
```

- **CTYPE Function**
  - It uses to convert one type to another type.
  - Instead of remember all conversion functions, remember only CTYPE function.

    Syntax: Ctype (expression, Type name)
    Example:

```
Dim no1 As Integer
Dim no2 As Double
no2 = 66.7
no1 = CType (no2, Integer)
Console.Writeline(no1)
```

  - Execution is faster because there is no call to a procedure to accomplish the conversion.
- **Type Checking Functions**
  - VB.NET provides number of data verification or data type checking functions:
    - IsDate
    - IsNothing
    - IsNumeric
    - IsDBNull
    - IsArray

- **IsDate**

  Returns True or False.

  Example:

```
Dim dt1, dt2 As Date
Dim strTemp As String
Dim bolAns As Boolean
dt1 = "February 17, 2007"
dt2 = # 1/17/1982#
strTemp = "India is great"
bolAns = IsDate(dt1)
Console.Writeline(bolAns)              ‘Returns True.
Console.Writeline(IsDate(dt2))         ‘Returns True.
Console.Writeline(IsDate(strTemp))     ‘Returns False.
```

- **IsNothing**

  Returns True if the Object variable that currently has no assigned value; otherwise, it returns False.

  Example:

```
Dim objTemp As Object                      'No instance assigned to this variable yet.
Dim bolAns As Boolean
bolAns = IsNothing (objTemp)               ‘Returns True.
Console.Writeline(bolAns)

objTemp = "Dolly"                          ‘Assign a string instance to the variable.
Console.Writeline(Is Nothing (objTemp))    ‘Returns False.

objTemp = Nothing                          ‘Assign Nothing means write now no value
Console.Writeline(IsNothing (objTemp))     'Returns True.
```

- **IsNumeric**

  Returns True if the value is numeric; otherwise, it returns False.

  Example:

```
Dim objTemp As Object
Dim str As String
Dim bolAns As Boolean

objTemp = 53
bolAns =
Console.Writeline(IsNumeric(objTemp))          ‘Returns True.

str = "459.95"
Console.Writeline(IsNumeric(str))              'Returns True.

objTemp = "45 Help"
Console.Writeline(IsNumeric(objTemp))          ‘Returns False.
```

- **IsDBNull**
  - IsDBNull returns True if the value evaluates to the DBNull type; otherwise, returns False.
  - The System.DBNull value indicates that the Object represents missing Or nonexistent data.
  - **DBNull is not the same as Nothing or null string.**

  Example:

```
Dim objTemp As Object
Dim bolAns As Boolean

bolAns=IsDBNull(objTemp)                   ‘Returns False.
Console.Writeline(bolAns)

objTemp=”Karna”
Console.Writeline(IsDBNull(objTemp))       ‘Return False
objTemp = System.DBNull.Value
Console.WriteLine(IsDBNull(objTemp))       'Return True
```

2.2.4 Option Statements

There are four Option Statements available in VB.NET:

(i) Option Explicit
(ii) Option Strict
(iii) Option Compare
(iv) Option Infer

(i) Option Explicit

- It ensures that variables are declared before referring in code. In other words, Option Explicit forces to declare variable before it is used in the program.
- The general syntax option explicit statement is as follow:

Syntax

Option Explicit (On | Off)

- There are two modes of Option Explicit: On and Off mode.
- If Option Explicit is on, variable must be declared before used in the program. Otherwise, it will issue a compile-time error.
- By default the Option Explicit is **On**.

Example

If Option Explicit is set On, then following code cause the compile time error :
name="Visual studio" compiler error, variable name is not declared It must be written as:

Example:

```
Option Explicit Off
Module Module2
   Sub main()
      name = "Visual Studio"
      Console.WriteLine(name)
      Console.ReadKey()
   End Sub
End Module
```

(ii) Option Strict

- It ensures that conversion errors are avoided in the code.
- Option Strict On allows widening conversion statements, but a compile error occurs if a narrowing conversion is attempted that will cause data loss.
- The general syntax of option strict statement is as follow:

Syntax

Option Strict {On | Off}

- By default Option Strict is **off**.

Example

```
Option Strict On
Module Modulel
        Sub Main()
        Dim x As Integer = 7
        Dim f As Decimal
        Dim b as Byte
        f=x                 'does not gives error
        b=x                 'gives compile time error
        End Sub
End Module
```

(iii) Option Compare

- It is use to specify whether the conversion string comparison is in binary or text.
- Its value can be Binary or Text. By default, it is **Text**.
- The general syntax of option compare statement is as follow:

Syntax

Option Compare {Binary | Text }

Example:

```
Dim a As String = "Hi"
Dim b As String = "HI"
If a = b Then
        Console.Writeline("TRUE")      'when Option Compare Text
Else
        Console.Writeline("FALSE")     ‘when Option Compare Binary
End If
```

(iv) Option Infer

- It enables local type inference (Anuman) in declaring variables.
- When you set On, you can declare variables without explicitly stating a data type.
- The compiler infers the data type of a variable from the type of its initialization expression.
- When you set Off, and you do not specify any data type then it is taken as Object type.
- The general syntax of option infer statement is as follow:

Syntax

```
Option Infer (On / Off)
```

Example

```
Option Infer On
Module Modulel
    Sub Main()
        Dim varl=2     ‘it is considered as Dim var1 As Integer
    End Sub
End Module
```

- Option Infer Off

```
Module Modulel
    Sub Main()
        Dim varl=2      'it is considered as Dim varl As Object
    End Sub
End Module
```

- **Imports statement**
  - It is used to import a namespaces so that you do not have to qualify items in code by listing the entire qualifying path of namespace when you referred them.

Syntax

```
Imports {namespace_name}
```

Example

```
Console.Writeline(math.abs(-5))      'code without imports statement
Imports System.Math
    Console.Writeline(abs(-5))       'code with imports statement
```

- **Comments**
  - Comments are optional part of coding.
  - It is displayed at design time.
  - It's very important for any software.
  - It never compiles. VB ignore whatever statements follows the in the comment
  - We add comments in our program to make clear particular concepts.
  - Like why we take this variable? What is the use of this Procedure? Programmer of this form, Date when we create this form. Etc.

    There are 3 ways to comment

    1. Using (Single quote)
    2. Using Rem: keyword (Remark)
    3. Using Toolbar Icon

Example:

```
Dim a is String                 ‘a is variable of type String
REM: VB.NET Notes
```

❖ Operators in VB.Net

- An operator is a symbol or character.
- It allows us to manipulate data. An operator performs a function on one or more operands.

  Example:

  Dim Ans As Integer

  Ans=2+5

  Here 2 and 5 are operands and + is an operator.
- Operators that take one operand are called **unary** operators.
- Operators that take two operands are called **binary** operators.

Unary Operators

- Used with a single operand.

  Example:

  Ans = -10

Binary Operators

- Used with a two operands.

  Example:

  Ans=10/5

❖ Category of operators

Type of Operator	Operators
Arithmetic	^, -, *, /, \, Mod, +
Concatenation	&, +
Assignment	=, ^=, *=, /=, \=, +=, -=, &=
Comparison/Relational	=, <>, <, >, <=, >=, Like, Is
Logical/bitwise operations	Not, And, Or, Xor, AndAlso, OrElse
Miscellaneous operations	GetType

- **Concatenation Operators**
  - The & (ampersand) and + (plus) characters is used for string concatenation.

    Example:

```
Dim str As String
str = "India is great"
Console.WriteLine(str)
str &= " I love my India"
Console.WriteLine(str)
str &= " I am Indian" & "I am proud to be an Indian "
Console.WriteLine("Hey..." & str)
Console.ReadLine()
```

  - We can also use + operator as a concatenation operator.

    Example:

```
Dim ans As String
ans = "Damyanti" + "Patel"          'String Concatenation
Console.Writeline(ans)
ans = 5 + 7                         'Arithmetic addition
Console.Writeline(ans)
```

- **Is**
  - The Is operator determines whether two object reference variables refer the same object instance.

Example:

```
Dim tmp1, tmp2 As New Object
Dim a As Object
a = tmp1
Console.WriteLine(tmp1 Is tmp2)     'False
Console.WriteLine(a Is tmp1)        'True
Console.WriteLine(a Is tmp2)        'False
```

- **Like Operator**
  - It compares a string to a pattern.
  - The Like operator can work **only** with operands of type **String and Character**.

Syntax:

Operand Like Pattern

Example:

```
Dim bolAns As Boolean
bolAns = "S" Like "s"                ‘Returns False
Console.WriteLine(bolAns)
bolAns = "S" Like "S"                'Returns True
Console.WriteLine(bolAns)
bolAns = "S" Like "SSS"              'Returns False
Console.WriteLine(bolAns)
```

We can also use below characters in pattern.

Characters in pattern	Matches in String
?	Any single character
*	Zero or more characters means any Alphanumeric or symbols
#	Any single digit (0-9)
[charlist]	Any single character in charlist means in between **Example**: [A-C], [D-M]
[!charlist]	Any single character not in charlist means not in between **Example**: [!A-C], [ID-M]

Example:

```
bolAns= "Ritu" Like "R*u"                    'Returns True
Console.WriteLine(bolAns)
Console.WriteLine("R" Like "[A-Z]")          ‘Returns True, means between A....z to
```

Example:

```
Console.WriteLine("R" Like "[!A-C]")         'Returns True, means not between A,B,C
```

Example:

```
Console.WriteLine("R2R" Like "R#R")          'Returns True
```

Example:

```
Console.WriteLine("aM5a2" Like "a[B-T]#?#")  'Returns True
```

- **Logical/bitwise Operations**

  And Also - with this operator, if the first operand is False, the second operand is no tested.
  OrElse -with this operator, if the first operand is True, the second is not tested.

  Example:

```
Dim no1, no2 As Integer
no1 = 10
no2 = 20
If no1 < 5 AndAlso no2 > 10 Then
MsgBox("Match")
Else
MsgBox("Not match")
End If
```

- In above example, first condition is False so **AndAlso** operator doesn't wastetime to check second one.

  Example:

```
Dim no1, no2 As Integer
no1 = 10
no2 = 20
If no1> 5 OrElse no2 < 10 Then
MsgBox("Match")
Else
MsgBox("Not match")
End If
```

- In above example, first condition is True so OrElse operator doesn't waste time to check second one.

- **Xor**
  - Performs a Boolean exclusive or operation on the bits.(1 1 0, 0 0 0, 1 0 1, 0 1 1)

Example:

```
Dim a,b,c As Integer
Dim k as boolean
a = 10
b = 20
c=30
MsgBox( b > a Xor c> b)                'False
k = a > b Xor b> c                     'False
MsgBox(k)
```

- **GetType**
  - Returns a Type object for the specified type.

Syntax: GetType(typename)

Example:

```
Dim a As Integer = 5
Dim b As String = "Hello"
Console.WriteLine(a.GetType)
Console.WriteLine(b.GetType)
Console.ReadKey()
```

2.3 String & Date Functions and Methods

❖ String Class Functions

- **Length**
  - It's a property not a function.
  - It returns the length of the string.
  - Len() is function also.
  - We can use any of this 2 options to check length of the string.

Example:

```
Dim Str As String = "Tanay and Shiv are good boys."
Dim ans As Integer
ans = Str.Length()
MsgBox(ans)
MsgBox(Len(Str))
```

- **Compare**
  - It's use for comparing 2 strings.
  - It returns less than zero, zero or greater than zero.
  - Less than zero : str1 is less than str2.
  - Zero : str1 is equal to str2.
  - Greater than zero : str1 is greater than str2.

Syntax:

Compare(String str1,String str2, Boolean) As Integer

Where, Boolean True/False Indication to check the String with case sensitive or without case sensitive.

Example:

```
Imports System.String
Dim a As Integer
 a = String.Compare("abc", "ABC", True)          '0
MsgBox(a)
MsgBox(Compare("ABC", "abc"))                    '1
MsgBox(Compare("ABC", "ABC"))                    '0
```

- **Concat**
  - Combined 2 strings.
  - Places them one after the other and forms a new string.
  - We can use &,+ operators also instead of this.

    Example:

```
Dim str1 As String= "My name is Damyanti Patel. "
Dim str2 As String= " My contact no is 9033793520"
Dim S As String
S= String.Concat(str1, str2)
MsgBox(s)
```

- **Copy**
  - Copy string. From source to destination.
  - Instead of this we can use directly operator like a=b means copy from b to a also.

    Example:

```
Dim str1 As String= "My name is Damyanti Patel. "
Dim str2 As String
str2 = String.Copy(str1)
MsgBox(str2)
```

- **Equals**
  - Determines whether two String objects have the same value.

    Example:

```
MsgBox(String.Equals("ABC", "ABC"))        'True
MsgBox(String.Equals("ABC", "abc"))        'False
```

- **Trim**
  - Removes starting and ending spaces from the string .
  - We can also pass parameter and remove particular character from starting and ending of the string.

    Example:

```
Dim S As String = "     Hello     "
MsgBox(S.Length)                '16
S = S.Trim()                    'It is not part of String class
MsgBox(S.Length)                ' 5
S = "### Tanay is good boy. ####"
MsgBox(S.Length)                '27
Dim S1 As String
S1 = S.Trim("#")
MsgBox(S1)              'Tanay is good boy.
MsgBox(S1.Length)               '20
```

- **Ends With**
  - Checks whether the end of this instance matches the specified String.
  - The return type is Boolean.
  - It is overload function.
  - We can pass second parameter as case sensitive.

    Example:

```
Str= “hellooooooo ”
If str.EndsWith("oo") Then
MsgBox("Yes")
Else
MsgBox("No")
End If
```

    Output:

    Yes

- **StartsWith**
  - Exactly opposite of EndsWith. It checks character of starting.

Example:

```
Str = "hellooooooo"
If str.StartsWith("hel") Then
    MsgBox("Yes")
Else
    MsgBox("No")
End If
```

Output:

Yes

- **Indexof**
  - Reports the index of the first occurrence of a String. (index start with 0)

Example:

```
Dim str As String
Str = "The Complete Reference VB.net is an Excellent Book"
Dim a As Integer
a = Str.IndexOf("E")                    '36
MsgBox("Place No -> " & a)
a = str.IndexOf("e")                    '2
MsgBox("Place No -> " & a)
a = str.IndexOf("e", 10)                '11
MsgBox("Place No -> " & a)
```

- **Ucase OR ToUpper**
  - Converts a string to uppercase

- **Lcase OR ToLower**
  - Converts a string to lowercase

Example:

```
Dim str As String
Str = "The Complete Reference VB.net is an Excellent Book"
Console.WriteLine(str.ToLower())
Console.WriteLine(str.ToUpper())
Console.WriteLine(LCase(str))
Console.WriteLine(UCase(str))
```

- **Insert**
  - Inserts a specified instance of String at a specified index position in this String.
  - Two arguments it (Star Index, Value).
  - The startIndex means the index position of the insertion and the Value means the String to insert.

Example:

```
Dim str As String = "Visual .net"
Str = Str.Insert(7, "Studio")
Console.WriteLine(Str)
```

- **PadLeft, PadRight**
  - Both are used for alignment purpose see the below example.

Example:

```
str = "I have $ 5000"
str = str.PadLeft(25, "*")
Console.WriteLine(str)
```

Output:

**********I have $ 5000

Example:

```
str = "I have $ 5000"
str = str.PadRight(25, "*")
Console.WriteLine(str)
```

Output:

I have $ 5000**********

- **Remove**
  - Deletes a specified number of characters from this instance beginning at a specified position.
  - Two arguments of it (StartIndex, Count).
  - **StartIndex** means the position in this instance to begin deleting characters and **count** means the number of characters to delete.

Example:

```
Dim Str As String
Str = "Vb.net is very interesting Subject"
Str = Str.Remove(10, 5)
MsgBox(Str)
```

Output:

Vb.net is interesting Subject

- **Replace**
  - Replaces all occurrences of a specified character or String.
  - There are 2 types of arguments of it (Char, Char) and (String, String).
  - It means overloaded function.

Example:

```
Dim Str As String
Str = "Jingle bells Jingel bells, Jingle All the ways"
Str = Str.Replace("J", "B")
MsgBox(Str)
```

Output:

Bingle bells Bingel bells, Bingle All the ways

Example:

```
Dim Str As String
Str = "Jingle bells Jingel bells, Jingle All the ways"
Str = Str.Replace("bells", "tells")
MsgBox(Str)
```

Output:

Jingle tells Jingel tells, Jingle All the ways

- **Substring(IMP)**
  - Retrieves a substring from this instance.
  - There are 2 types of arguments it. (Start Index) and (Start Index, Count).
  - StartIndex means from where to start and Count means up to how many character

Example:

```
Dim Str As String
Str = "Jingle bells Jingel bells, Jingle All the ways"
Str = Str.Substring(7)
  MsgBox(Str)
```

Output:

bells Jingel bells, Jingle All the ways

Example:

```
Dim Str As String
Str = "Jingle bells Jingel bells, Jingle All the ways"
Str = Str.Substring(7, 13)
MsgBox(Str)
```

Output:

bells Jingel

Char class's Method

Name	Description of methods
IsDigit	Checks a character is decimal digit or not
IsLetter	Checks a character is an alphabetic letter or not
IsLower	Checks a character is lowercase letter or not.
IsNumber	Checks a character is number or not.
IsPunctuation	Checks a character is punctuation mark or not.
IsSeprator	Checks a character is separator character or not.
IsSymbol	Checks a character is symbol character or not.
IsUpper	Checks a character is an uppercase letter or not.
IsWhiteSpace	Checks a character is white space or not.
ToLower	Converts the value of a character to its lowercase.
ToUpper	Converts the value of a character to its uppercase.

Example:

```
Sub main()
'IsDigit():
Console.WriteLine(IsDigit("8"))                                  'True
Console.WriteLine(IsDigit("sample string", 7))                   'False
'IsLetter():
Console.WriteLine(IsLetter("7"))                                  ' False
Console.WriteLine(IsLetter("sample string", 7))                  ' True
'IsLower():
Console.WriteLine(IsLower("d"))                                  'True
Console.WriteLine(IsLower("sample String", 7))                   ' False
'IsNumber():
Console.WriteLine(IsNumber("8"))                                 'True
Console.WriteLine(IsNumber("non-numeric", 3))                    'False
'IsPunctuation()
Console.WriteLine(IsPunctuation("."))                            ' True (.,?;:!)
Console.WriteLine(IsPunctuation("no punctuation", 3))            'False
'IsSeparator()
Console.WriteLine(IsSeparator("a"))                              'False
Console.WriteLine(IsSeparator("sample  String", 6))              'True
'IsSymbol()
Console.WriteLine(IsSymbol("+"))                                  'True
Console.WriteLine(IsSymbol("sample String", 6))                  'False
'IsUpper ()
Console.WriteLine(IsUpper("A"))                                  'True
Console.WriteLine(IsUpper("Sample String", 7))                   'True
'IsWhiteSpace()
Console.WriteLine(IsWhiteSpace("A"))                             'False
Console.WriteLine(IsWhiteSpace("Sample String", 6))              'True
'ToLower ()
Console.WriteLine(Char.ToLower("D"))                              'd
'ToUpper ()
Console.WriteLine(Char.ToUpper("d"))                             'D

Console.ReadKey()

End Sub
```

❖ **Other inbuilt functions**

- **IIf**
  - Returns one of two objects, depending on the evaluation of an expression.
  - It is like Ternary operator in C or C++.

  Syntax:

  IIf(Expression, Truepart, Falsepart)

  Example:

```
Dim salary As Integer = 50000
Dim strAns As String
strAns = IIf(salary> 30000, "Good", "Bad")
MsgBox(strAns)
```

  Output:

  Good

- **Mid**
  - Returns a string containing a specified number of characters from a string.
  - Equivalent to Substring method of String class.

  Syntax:

  Mid(String, StartIndex, Length)

  Example:

```
Dim str As String
str = "Jamnagar is good city"
Console.WriteLine(Mid(str, 10))          'Returns is good city
Console.WriteLine(Mid(str, 1, 3))        'Returns Jam
Console.WriteLine(Mid(str, 10, 7))       'Returns is good
Console.ReadKey()
```

- **StrReverse**
  - Returns a string in which the character order of a specified string is reversed.

  Syntax:

  StrReverse(String)

  Example:

```
Dim str As String ="Tanay is Good Boy."
Dim rev As String
rev =StrReverse(str)              ‘Returns ".yoB dooG si yanaT"
```

- **ASC**
  - Returns ASCII value for the input character.

  Syntax:

  Asc(char) or Asc(string)

  Example:

```
Dim ans As Integer
ans = Asc("a")              'Returns 97
ans =Asc("A")               'Returns 65
ans = Asc("0")              'Returns 48
ans = Asc("9")              'Returns 57
```

- **Chr**
  - Returns return the character associated for the input ASCII number, Excatally opposite of ASC function.

  Syntax:

  Chr(number)

  Example:

```
Dim ans As Char
ans = Chr(97)          'Returns a
ans = Chr(65)          'Returns A
ans = Chr(57)          'Returns 9
ans = Chr(48)          'Returns 0
```

❖ Date-Time Functions

- In **DateTime** class, Date datatype stores date values, time values or date, and time values.
- Further more, to perform the date and time function, we need to import the **System.DateTime** class.
- The default value of DateTime is between 00:00:00 midnight, Jan 1, 0001 to 11:59:59 P.M., Dec 31, 9999.

Properties and method of DateTime

- **Date:** It is used to return the date component of the DateTime Object.
- **Day:** It is used to return the day of the month represented by the DateTime object.
- **DayOfWeek:** It is used to return a particular day of the week represented by the DateTime object.
- **Minute:** The Minute property is used to return the minute component by the DateTime object.
- **DateOfYear:** It is used to return a day of the year represented by the DateTime object.
- **Hour:** It is used to return the hour of the component of the date represented by the DateTime object.
- **Now:** It is used to return the current date and time of the local system.
- **Month:** The Month property is used to return the month name of the Datetime object.
- **MonthName:** Returns the month name from given datetime object.
- **Second:** It is used to return the second of the DateTime object.
- **Today:** It is used to return the current date of the system.
- **Year:** It is used to return the year of the date represented by the DateTime object.
- **TimeOfDay:** It is used to return the time of the day represented by the DateTime object.

Syntax:

```
Dim obj_name As DateTime = New DateTime()
```

Here, **DateTime** is a class for creating objects with the new keyword, and **obj_name** is the name of the object.

Example:

```
' Create DT as an instance of DateTime
Dim DOB As DateTime = New DateTime()
Console.WriteLine("Default Date and Time is :" & DOB)
Console.WriteLine()
'Dim DT As DateTime = New Date(1998, 9, 7, 12, 22, 30)
DOB = Now
Console.WriteLine("Current Date and Time is :" & DOB)
Console.WriteLine(" Different function of DateTime")
Console.WriteLine("================================")
Console.WriteLine(" Date is : {0}", DOB.Date)
Console.WriteLine(" Date is Using Today : {0}", Today)
Console.WriteLine(" Day is : {0}", DoB.Day)
Console.WriteLine(" Month is : {0}", DOB.Month)
Console.WriteLine(" Month Name is : {0}", MonthName(DOB.Month))
Console.WriteLine(" Year is : {0}", DOB.Year)
Console.WriteLine(" Hour is : {0}", DOB.Hour)
Console.WriteLine(" Minute is : {0}", DoB.Minute)
Console.WriteLine(" Second is : {0}", DoB.Second)
Console.WriteLine(" Millisecond is : {0}", DoB.Millisecond)
Console.WriteLine(" Day of Week is : {0}", DoB.DayOfWeek)
Console.WriteLine(" Day Of year is : {0}", DoB.DayOfYear)
Console.WriteLine(" Time of Day is : {0}", DoB.TimeOfDay)

Console.ReadKey()
```

- **Methods**

The following are the most commonly used methods of the DateTime.

- **DateDiff**: It returns the number of intervals between two dates.
- **DaysInMonth:** The DaysInMonth method is used to return the total number of days in the specified month of the year.
- **AddHours:** It is used to return a new time by adding the hours to the value of the Datetime object.
- **AddYears:** It is used to return the year by adding the year to the value of the DateTime object.
- **AddDays:** It is used to return the new Day by adding the days to the value of the DateTime object.
- **AddMinutes:** It is used to display the new time by adding the minutes to the Datetime object.
- **AddMonths:** It is used to return the new time by adding the months to the value of the Datetime object.
- **AddSeconds:** It is used to return the new time by adding the seconds to the value of the Datetime object.
- **IsLeapYear:** It uses a Boolean value that represents whether the particular year is a leap year or not.

Example:

```
Imports System.DateTime

 Dim DOB As Date
 DOB = Now()
 Dim dt As Date = #8/18/1992#
Console.WriteLine("DateDiff:" & DateDiff(DateInterval.Day, dt, Now()))
Console.WriteLine("Age:" & DateDiff(DateInterval.Day, dt, Now()) / 365)
Console.WriteLine("Days in Month:" & DaysInMonth(DOB.Year, DOB.Month))

Dim d1 As DateTime = New DateTime(2020, 11, 2, 9, 0, 10)
Console.WriteLine("Assign Date:" & d1)
Console.WriteLine("AddYears:" & d1.AddYears(4))
Console.WriteLine("AddDays:" & d1.AddDays(10))
Console.WriteLine("AddMonths:" & d1.AddMonths(3))
Console.WriteLine("AddHours:" & d1.AddHours(2))
Console.WriteLine("AddMinutes:" & d1.AddMinutes(30))
Console.WriteLine("AddSeconds:" & d1.AddSeconds(25))
Console.WriteLine("IsLeapYear:" & IsLeapYear(d1.Year))
If IsLeapYear(d1.Year) Then
        Console.WriteLine("Yes, This is Leap Year")
Else
        Console.WriteLine("No, This is not a Leap Year")
End If
```

- **Format Function**

Returns a string formatted according to Instructions contained in a format String expression.
There are 2 arguments of it an expression and style.

Example:

```
 Console.WriteLine(Now)
 Console.WriteLine(Format(Now, "M-d-yy"))
 Console.WriteLine(Format(Now, "MM-dd-yyyy"))
 Console.WriteLine(Format(Now, "MMMM-d-yyy - dddd"))
 Console.WriteLine(Format(Now, "hh:mm:ss tt"))
```

2.4 Modules, Procedures and Functions

▪ Modules

- It is a file to store group of functions in a single unit. User can write several procedures in the module and that can be used on different forms and classes in the application.

Adding module in the project

(a) In solution explorer, Right click on project
(b) Select Add from popup menu
(c) Select Module
(d) Give Proper name
(e) Click on Add button.

Another way to add Module in the project is as follow:

(a) In Solution Explorer, Right Click on project
(b) Select Add from popup menu
(c) Select New Items
(d) From Template select Module
(e) Give Proper name
(f) Click on Add button.

- Write and WriteLine write a string to the console.
- ReadLine reads input from the console, in this case a string.
- ReadKey() pauses the app and waits for a keypress.

▪ Procedures

- A procedure is a set of one or more pogrom statements that can be call by referring to the procedure name.
- We can divide the big program into small procedures.
- Due to Procedure the application is easier to debug and easier to find error.
- Reusability of procedure is the one of the great advantage of it.
- Procedure have 2 categories In-built or User-defined procedures.
- There are **four** types of Procedures in VB.Net
  - Sub procedures does not return value.
  - Event-handling procedures are Sub procedures that execute in response to an event triggered by user action. Like button1_Click
  - Function procedures return a value.
  - Property procedures used in object oriented concept.

- **Modular Coding**

There are 2 approaches for doing modular coding in VB.net

1. Sub Routines or Sub Procedure
2. Function Procedure

The main concept is breaking large application into smaller.

If your procedure does not return any value then it should be implemented as a sub procedure else function.

1. Sub Procedures

- A sub procedure is nothing more than a small block of program in itself,It is also known as Sub Routine.
- There are 2 parts of it declaring and calling.
- We should give proper name to it through which we can remember it easily.
- Sub procedure may or may not have arguments. Arguments are not compulsory.
- Sub procedure does not return the value.
- **By Val** is by default argument type.
- So, do not worry to type it. It will automatically appear if we set parameter in the Sub procedure.
- 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

Example: **Create Sub Procedure when user click on button call it**

```
Private Sub btnAns_Click(ByVal sender As System. Object)
        Call magic()
End Sub

Sub magic()
   MsgBox("Hi good morning")
End Sub
```

We can call sub procedure using **Çall keyword or without Call keyword.**

Examples with argument sub procedure.

```
Sub magic(ByVal str As String)
  MsgBox("Hello how are u?" & str)
End Sub
```

Calling: Call magic(“Sutex”) or Call magic(Textbox1.Text)

- **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.

```
Sub swap(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

Sub Main()
   Dim a As Integer = 100
  Dim b As Integer = 200
  Console.WriteLine("Before swap, value of a : {0}", a)
   Console.WriteLine("Before swap, value of b : {0}", b)
  swap(a, b)
  Console.WriteLine("After swap, value of a : {0}", a)
  Console.WriteLine("After swap, value of b : {0}", b)
  Console.ReadLine()
  End Sub
```

Output: −

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

It shows that there is **no change** in the values though they had been changed inside the function.

- **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 swap(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
  Sub Main()
   Dim a As Integer = 100
   Dim b As Integer = 200
   Console.WriteLine("Before swap, value of a : {0}", a)
   Console.WriteLine("Before swap, value of b : {0}", b)
   swap(a, b)
   Console.WriteLine("After swap, a : {0}", a)
   Console.WriteLine("After swap, b : {0}", b)
   Console.ReadLine()
 End Sub
```

Output: −

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100
```

2. Functions

- A Function 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.
- Functions return a value, whereas Subs do not return a value.
- By default functions are public

- **Defining a Function**

The syntax for the Function statement is −

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

Where,

- ***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

Function Returning a Value

In VB.Net, a function can return a value to the calling code in two ways −

- By using the return statement
- By assigning the value to the function name

Output:

```
Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
    Dim result As Integer
    If (num1 > num2) Then
       result = num1
    Else
       result = num2
    End If
    Return result   'Or    FindMax=result
End Function
  Sub Main()
    Dim a As Integer
    a = FindMax(25, 85)
    Console.WriteLine("Maximum Number:" & a)
    Console.ReadKey()  'Output: Maximum Number:85
End Sub
```

2.4.2. Optional Argument/Default Argument

- We can specify procedure argument as an optional.
- In C++ this concept is known as default argument.
- Optional means not to pass argument compulsory when the procedure is called.
- They are indicated by the Optional keyword in the procedure default value else if we pass then default value is overwritten.
- If we do not pass the value of optional argument then it will take its default
- **Some rules of the optional argument:**
- Remember that if we make one argument optional, all the following arguments must be optional.
- We must specify a default value for every optional argument.

Syntax:

```
Sub subProcName(para1 as <type>,optional para2 as <type>=value)
End Sub
```

Example:

```
Function funcName(paral as <type>,optional para2 as <type>=value)as<type>
End Function
```

Example: Create Add sub procedure which contains second argument as Optional argument

```
Sub add(ByVal no1 As Integer, Optional ByVal no2 As Integer = 0)
        MsgBox(no1+no2)
End Sub
```

Calling:

```
add(5,2)
add(5)  ' It will take value of no2 is 0 which is default value.
```

2.6. Control Flow Statement

- It is used to declare the flow of program.
- Control statements enable us to execute a certain set of statements based on condition.
- The control structures are an important concept in high-level programming languages.
- They are blocks of code that dictate the flow of control.

2.6.1 Conditional / Branching Statement

- A branch is a point at which your program must make a choice.
- A branch statement, or conditional statement, is a code that allows you to test whether statements are true or false and then execute some code based on comparisons.

- **<u>If...Then...ElseIf Statement</u>**
  - The If..Then..Elself statement is the conditional statements.
  - There are cases when we would like to execute some when a condition is TRUE and some other logic when the condition is FALSE.

Syntax

```
If condition1 Then
    [statement(s)]
Elself  condition2 ..n   Then
   [statement(s)]
Else
 [statement(s)]
End If
```

Example

```
Dim marks As Integer = 45
If marks >= 35 Then
        MsgBox("Hey!!! I am Passs.")
Else
        MsgBox("Ohhh!!! I am Fail.")
End If
```

```
Dim a As Integer = 1
 If a = 0 Then
    MsgBox("I am Zero.")
 ElseIf a > 0 Then
    MsgBox("I am Positive")
 Else
    MsgBox("I am Negative")
 End If
```

- **Select...Case Statement**
  - Select...Case statement is used to avoid long chains of If...Then...Elself statement.
  - When you are comparing the same expression with several different values, we can use the Select... Case
  - statements as an alternative to the If....Then....Else statements.
  - If...Then...Else block are time consuming then Select ... case block.
  - If no one case statement is satisfied then case Else clause's logic will run.
  - We can use **To** keyword to specify a range of values.
  - We can also use **IS** keyword. The **IS keyword is used with any comparison operators.**

Syntax

```
Select Case testexpression
Case expression1
      statement(s).....n
Case Else
       statement(s)
End Select
```

Example

```
Dim a As String
a = Console.ReadLine()
Select Case a
   Case "I"
      MsgBox("Thank you.")
   Case 2
      MsgBox("That's fine.")
   Case 3
      MsgBox("OK.")
   Case 4 To 7
      MsgBox("In the range 4 to 7.")
   Case Is > 7
      MsgBox("Definitely too big.")
   Case Else
      MsgBox("Not a number I know.")
End Select
```

- **Loop Statements**
  - Looping allows user to run one or more lines of code repetitively.
  - You can repeat the statements in a loop until a condition is True.
  - Once a condition is False execution of those statements stops.
  - VB .NET supports
    1. While...End While
    2. Do ....Loop
    3. For..Next
    4. for-each loops.
- **While..... Loop**
  - While Loop runs set of statements as long as the condition specified with **While** loop is True.

Syntax:

```
 While condition
   [Statements]
End While
```

Example

```
Dim i As Integer = 1
While i <= 10
    Console.WriteLine(i)
 i += 1
End While
```

- **Do..... Loop**
  - Do Loop allows you to test a condition at either the beginning or at the end of a loop.
  - You can also specify whether to repeat the loop while the condition remains True or False.
  - There are two variations of Do loop: entry-control loop and exit-control loop.

  Syntax

```
Do {While/Until}  Condition
    Logic....
Loop
```

OR

```
Do
  Logic....
Loop {While/Until}  Condition
```

  - We can use either While and Until.
  - **While**: Repeat the loop until condition is False
  - **Until** : Repeat the loop until condition is True

  Example 1:

```
Dim i As Integer = 1
Do While i <= 10
    Console.WriteLine(i)
 i += 1
Loop
```

```
Dim i As Integer = 1
Do Until 10 < i
    Console.WriteLine(i)
     i += 1
Loop
```

  Example 2:

```
Dim i As Integer = 1
Do
    Console.WriteLine(i)
 i += 1
Loop While i <= 10
```

```
Dim i As Integer = 1
Do
    Console.WriteLine(i)
     i += 1
Loop Until 10 < i
```

- **For .... Next**
  - The For loop executes a block of statements a specified number of times.
  - It's most commonly and most popular loop used in any programming language.
  - Remember Next keyword should be used in For....Next at the end.

  Syntax

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

  - Here counter variable is initialized when the loop starts and runs up to the end values.
  - We can set increment and decrement by step keyword.
  - Step keyword is optionally to write.
  - By default loop will increment by 1.

  Example

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

```
Dim i As Integer
For i = 1 To 10 Step 2
  Console.WriteLine(i)
Next
```

- **For each Loop**

  We will discuss in topic 2.5 Array and Collection

❖ MsgBox(), MessageBox.Show() and InputBox()

• MsgBox()

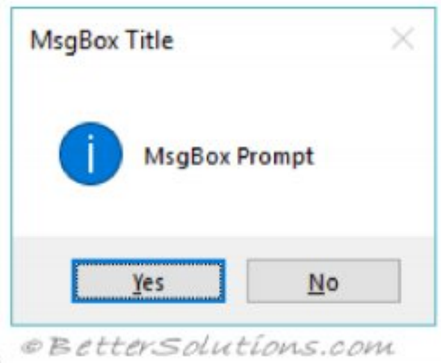
- Displays a message in a dialog box, waits for the user to click a button, and returns an **Integer** indicating which button the user clicked.

Syntax

Ans=MsgBox (prompt, [buttons+Icon] ,[title])

- Prompt is compulsory argument.
- Buttons and Icon is optional argument.
- By default title is Project Name

Example: **MsgBox("MsgBox Prompt", MsgBoxStyle.YesNo, "MsgBox Title")**



The list of button below and the return value of it.

Value	Button
1	vbOK
2	vbCancel
3	vbAbort
4	vbRetry
5	vbIgnore
6	vbYes
7	vbNo

Some of the values of argument style as given below:

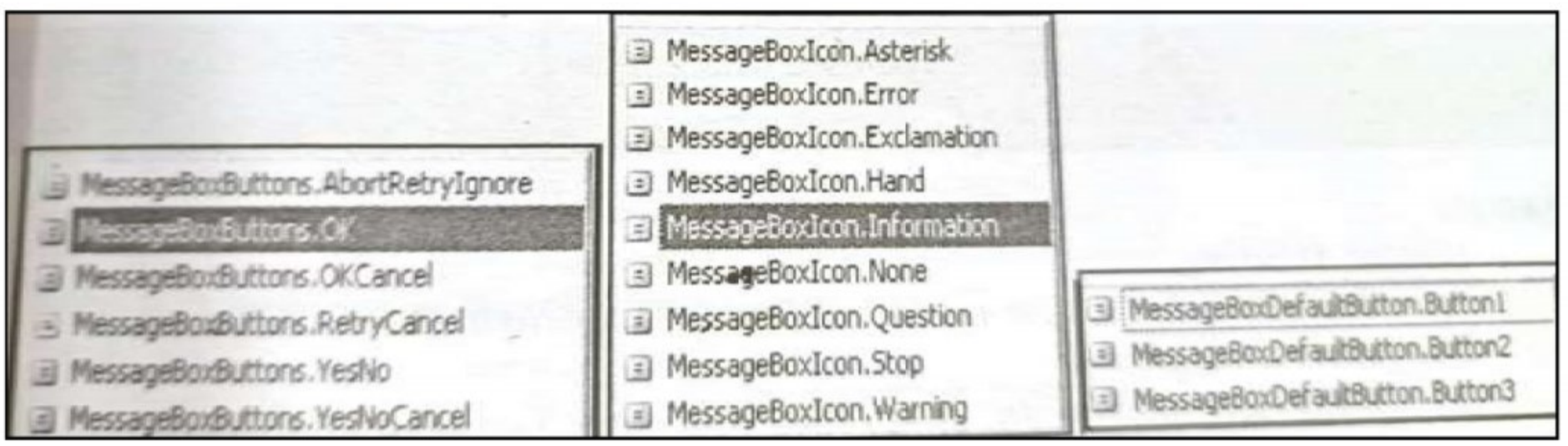
Constant	Description
OKOnly	Display OK button only.
OKCancel	Display OK and Cancel buttons.
AbortRetryIgnore	Display Abort, Retry, and Ignore buttons.
YesNoCancel	Display Yes, No, and Cancel buttons.
YesNo	Display Yes and No buttons.
RetryCancel	Display Retry and Cancel buttons.
Critical	Display Critical Message icon.
Question	Display Warning Query icon.
Exclamation	Display Warning Message icon.
Information	Display Information Message icon.
DefaultButton1	First button is default.
DefaultButton2	Second button is default.
DefaultButton3	Third button is default.
DefaultButton4	Fourth button is default.

MessageBox.Show

- It's an advance version of MsgBox function.
- MessageBox is the class and Show is the method of it.
- Show method has more arguments then MsgBoxfunction.It containsseparate arguments for icon and button.
- It also provides alignment option, helpbutton and default button options.

Syntax:

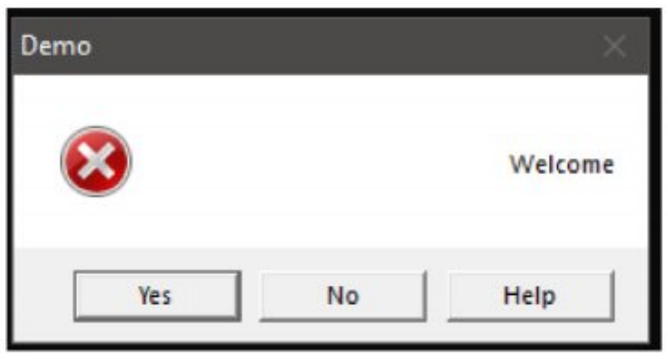
MessageBox.Show(Text,caption,button,icon,defaultbutton, options,help,Button)



Example:

MessageBox.Show("Welcome", “Demo”, MessageBoxButtons. YesNo,
MessageBoxIcon.Error,MessageBoxDefaultButton.Button1,

Output:



❖ InputBox

- InputBox function is to accept data from the user.
- An InputBox function will display a prompt box where the user can enter avalue or a message in the form of text.
- Instead of taking Textbox on the form we can get the user value usingInputBox.
- The Inputbox function with return a value. We need to declare the datatype for values to be accepted in the Inputbox

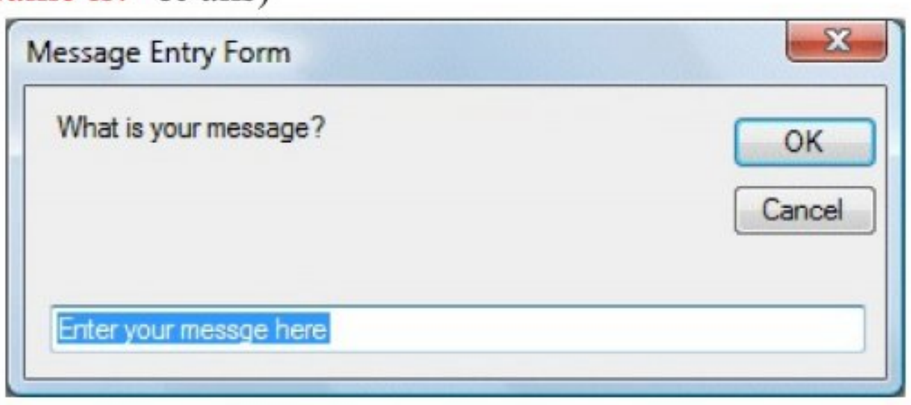
Syntax:

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

- The **prompt** argument will be displayed as message when a question is asked.
- The**title** argument is the title of the Input Box.
- The **default_text** argument appearsthe Input field where users can use it as his intended input or he may change to the message he wish to key in and x-position and y-position are the position or the coordinate of the Inputbox.

Example: **Accept the message from InputBox and display name in the messagebox.**

```
Private Sub btnAns_Click(ByVal sender As System.Object, ByVal e As System)
Dim ans As String
 ans = InputBox("What is your message?", "Message Entry Form", "Enter your message here")
 MsgBox("My Name is:" & ans)
 End Sub
```



2.5 Using Arrays and Collection

- **Array**
  - An array is a collection of elements of same data type.
  - Array elements are accessed using a single name and an index number, representing position of the element within the array.

Syntax

Dim Array_name(no_of_elements) As Data type

Example

Dim Num(9) As Integer 'It creates 10 elements with index 0-9.

- There is another way to create an array. A user can initialize array elements at the time of declaration.
- The general syntax to initialize an array at the time of declaration is as follow:

Syntax

Dim Array_name() As Data type = New Data type(no_of_elements) {[list of constant separated by comma]}

- Here Curly braces are used to initialize array elements.

Example:

Dim Num() As Integer=New Integer(){0,1,2,3,4,5,6,7,8,9}

- **For each Loop**
  - The For Each....Next loop is similar to the For...Next loop, but it executes the statement block for each element in a collection on array.
  - Repeats a group of statements for each element in a collection.

  Syntax:

```
For Each element In Group
  Logic....
Next
```

  Example:

```
Dim a() As Integer = {1, 2, 3, 4, 5}
Dim b() As Integer = New Integer() {11, 2, 3, 4, 5}
For Each i In b
  Console.WriteLine(i)
Next
```

- **Multidimensional Array**:
  - User can also create multidimensional array.
  - The general syntax of a multidimensional array declaration is as follow:

  Syntax

  Dim Array_name ([Lower Bound, Upper_Bound]) As Integer

  Example

```
Dim Mat(2,2) As Integer
```

  - User can initialize 2-Dimensional Array as

  Example

```
Dim d(,) As Integer = New Integer(,) {{0, 0, 0}, {1, 1, 1}, {2, 2, 2}}
For i As Integer = 0 To 2
  For j As Integer = 0 To 2
    Console.Write(c(i, j) & "  ")
  Next
  Console.WriteLine()
Next
```

- **Assigning values to Array elements**
  - After declaring an array, user can assigned a value to an array element.
  - The general Syntax for assigned a value to an array element is as follow:

  Syntax:

  Array_name(index)=value

  Example

```
Dim Num(9) As Integer                    '1 Dimensional Array
Num(0)=5
Dim Mat(2,2) As Integer                  '2 Dimensional Array
Mat(0,0)=5
```

- **Resizing Array**
  - You can resize an array using ReDim Statement. The general syntax of resizing a is as follow:

  Syntax

  ReDim [Preserve] array_name(new size of an array)

  Example

  If you want to resize Num Array to hold 20-elements then

  ReDim Num(20)

  - When you use ReDim Statement, the existing contents are erased.
  - You can use keyword to ensure that the existing contents of an array are not lost.

  Example

  ReDim Preserve Num(20)

  - If you use **Preserve** keyword while resizing an array, only the last dimension of the array can be resized.

Example:

```
Dim a(4) As Integer
a(0) = 101
a(1) = 102
a(2) = 103
a(3) = 104
a(4) = 105
For Each i In a
   Console.WriteLine(i)
Next
Console.WriteLine("==================================")
ReDim a(10)              'change it with " ReDim Preserve a(10)" and check Output
a(5) = 106
a(6) = 107
a(7) = 108
a(8) = 109
a(9) = 110
a(10) = 111
For Each i In a
   Console.WriteLine(i)
Next
```

- **Erase Statement**
  - It is used to release array variables and deal locate the memory used for their elements.
  - The general syntax of Erase element is as follow:

Syntax

Erase Array_namel [,array_name2 [, array_name3...]]

Example

```
Dim Num(9) As Integer
Erase Num
```

- **Traversing an Array**
  - A user can access all elements of an array using the following code:

Example1(Using For loop)

```
Dim Num(3) As Integer
Num(0) = 1
Num(1) = 2
Num(2) = 3
Num(3) = 4
For i As Integer = 0 To 3
   Console.WriteLine(Num(i))
Next
```

Example2(Using For…Each Loop)

```
Dim Num(3) As Integer
Num(0) = 1
Num(1) = 2
Num(2) = 3
Num(3) = 4
For Each i In Num
   Console.WriteLine(i)
Next
```

- **Built-In Function used with Array**

User can perform different operations on an array. Table 1 shows the list of functions that are applied on an array.

Functions	Description
GetUpperBound	It gets the upper bound of the specified dimension in the Array.
GetLowerBound	It gets the lower bound of the specified dimension in the Array.
GetLength	It gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
GetType	It gets the data type of the Array.
First	It gets the first element of an Array.
Last	It gets the last element of an Array.
Max	It gets the maximum element of an Array.
Min	It gets the minimum element of an Array.
IndexOf	It gets the index of the specified element of Array.

Example

```
Dim Num(3) As Integer
 Num(0) = 101
 Num(1) = 202
 Num(2) = 303
 Num(3) = 404
 Console.WriteLine("GetUpperBound:" & Num.GetUpperBound(0))        '3
 Console.WriteLine("GetLowerBound:" & Num.GetLowerBound(0))        '0
 Console.WriteLine("Length:" & Num.Length)                         '4
 Console.WriteLine("GetLength:" & Num.GetLength(0))                '4
 Console.WriteLine("GetType:" & Num.GetType().ToString)            'System.Int32
 Console.WriteLine("First Element:" & Num.First)                   '101
 Console.WriteLine("Last Element:" & Num.Last)                     '404
 Console.WriteLine("Max:" & Num.Max)                               '404
 Console.WriteLine("Min:" & Num.Min)                               '101
 Console.WriteLine("IndexOf:" & Array.IndexOf(Num, 303))           '2
```

Collections

- A collection is a one-dimensional storage area which holds value of mixed data types.
- Collections are increased in size simply by adding items to it.
- A collection is declared with a Dim statement that creates a new Collection object.
- The general syntax of Collection is as follow:

Syntax

Dim Collection_name As New Collection

Example

Dim Subject As New Collection

- **Adding Items in Collection**
  - Add() Method is used to add items in the collection.

Syntax

Collection_Name.Add(Value, Key)

Example

Subject.Add("VB.Net", "VB")

- **Accessing Items of Collection**
  - Item() method is used to access the items within the collections.

Syntax

Collection Name.Item(Index)

Items in a collection are indexed beginning with 1.

Or

Collection_Name.Item(Key)

Example

Subject.Item("VB")

Or

Actors.Item(1)

- **Traversing Collection**
  - For...Each Loop is used to traverse the collection.

Syntax

```
For Each element In Collection Name
    [statement(s)]
Next [element]
```

Example:

```
Dim Subject As New Collection
Subject.Add("Database", "db")
Subject.Add("Java Programming", 2)
Subject.Add("Vb.net Programming", "vb")
Subject.Add("OOPs Concepts", 3.14)
Subject.Add("Python Programming")
Subject.Add(6, "DK")

For Each i In Subject
   Console.WriteLine(i)
Next
```

Property and Built-In functions used with Collection

Count property and other methods Remove() and Clear() are also used with collection.

Example

```
Dim cnt As Integer
cnt = Subject.Count
Console.WriteLine("Count of Subject=" & cnt)
Subject.Remove("vb")                    'it remove the item with key "vb"
```

❖ Difference between Array and Collection

Sr. No.	Key	Arrays	Collection
1	Size	Arrays are fixed in size i.e once the array with the specific size is declared then we can't alter its size afterward.	The collection is dynamic in size i.e based on requirement size could be get altered even after its declaration.
2	Memory Consumption	Arrays due to fast execution consumes more memory and has better performance.	Collections, on the other hand, consume less memory but also have low performance as compared to Arrays.
3	Data type	Arrays can hold the only the same type of data in its collection i.e only homogeneous data types elements are allowed in case of arrays.	Collection, on the other hand, can hold both homogeneous and heterogeneous elements.
4	Primitives storage	Arrays can hold both object and primitive type data.	On the other hand, collection can hold only object types but not the primitive type of data.
5	Performance	Arrays due to its storage and internal implementation better in performance.	Collection on the other hand with respect to performance is not recommended to use.

❖ VB.NET Dynamic Array

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

Syntax:

```
Dim array_name() As Integer
```

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

Syntax:

```
ReDim {Preserve] array_name(subscripts)
```