Chapter 1 Introduction to VB 2010

❖ Introduction

In computing, a Visual Programming Language (VPL) is any programming language that lets users create programs by manipulating program elements ***graphically*** rather than by specifying them ***textually***. A VPL are based on the idea of boxes and other objects, where boxes or other screen objects are treated as entities which connected in some ways by relations.

VPLs may be further classified, according to the type and extent of visual expression used, into *icon-based languages*, *form-based languages*, and *diagram languages*. Visual programming environments provide graphical or iconic elements which can be manipulated by users in an interactive way according to some specific spatial grammar for program construction. An important benefit of learning Visual Basic and the Visual Studio *Integrated Development Environment* **(IDE)** is that you can use many of the same tools to write programs for Microsoft Visual C++ 2010, Microsoft Visual C# 2010, Microsoft Visual Web Developer 2010, and other popular products.

Visual programming languages are widely used for the rapid development of graphical applications. This subject will introduce students to the fundamental principles of the term **Event-Driven Programming** (*Events in this context include clicking a button, resizing a window, or changing an entry in a text box. The code that you write responds to these events. The program reacts to your movements and takes the necessary actions to complete your desired tasks*) and to programming using a **Visual Environment** through the use of the Visual C# programming language. An additional aim of this subject is to give students an understanding of the main ideas of Human Computer Interaction **(HCI)**.

❖ Visual Basic vs. VB .NET

Visual Basic **(VB)** was initially introduced in 1991 as the first programming language that directly supported programmable *Graphic User Interface* (**GUI)** using language supplied object. VB language, greatly make the creation of windows applications easy especially, by facilitating the use of re-usable components.

Visual Basic.NET **(VB.Net)** a programming language based on VB 6.0 working on the .NET framework of the Microsoft Corporation. This framework represents a platform for cross-language development (C#, VB. NET, C++, F#) and includes a large standard library called the Base Class Library **(BCL).**

Visual Studio intended mainly for ***Windows Applications and Web Applications***. We will use Visual Studio 2010 to create all of our programs.

Visual Basic is an Object Oriented Language **(OOL)** that let you program with many different program groups called ***Modules*** and ***Classes***. Visual Basic consists of two fundamental parts: a **visual part** *consists of set of objects* and **language part** *which consists of high level procedural programming language*. These two parts are used together to create **Application** which *is simply a visual basic program that can run under the windows operating system*.

❖ Prerequisites

All you need one version of Visual Studio 2010 Professional, Visual Studio 2010 Premium, or Visual Studio 2010 Ultimate to execute the exercises, if you don't have, you can complete most of the exercises by downloading Visual Basic 2010 Express from the Web for free. This will give you an opportunity to learn Visual Basic programming and see for yourself if you want to upgrade to a full release of the Visual Studio software.

To download Visual Basic 2010 Express, complete the following steps:

1. Open a Web browser and go to http://www.microsoft.com/express.
2. Follow the instructions on the screen to download Visual Basic 2010 Express. On the Express Web site, you will also see an Express product feature chart that compares the Express product to the full versions of Visual Studio. Although there are some key differences between the full versions and Visual Basic 2010 Express, many of these differences have no effect on how you learn the essential techniques and features of Visual Basic programming. After you experiment with the Express product, you can decide whether you want to upgrade to one of the full versions of Visual Studio or not.

❖ Microsoft Visual Studio & Visual Basic (.NET Version)

Microsoft's Visual Studio (also called Visual Studio.NET) includes several different programming languages:

- **Visual Basic**
- **Visual C#**
- **Visual C++**
- **Visual F#**
- **Jscript**
- Web Development (called **ASP.NET**)
- It also includes the **.NET 4 Framework** upon which these languages operate.

All of these languages **compile**. This means they are translated from human readable-form to machine readable-form to the same Microsoft Intermediate Language (MSIL),

then, MSIL run within the Common Language Runtime (CLR) – a component of the .NET Framework.

VB is also termed an event-driven programming language because you will write program code that responds to events that are controlled by the system user. Example events include:

- Clicking a button or menu.
- Opening or Closing a form.
- Moving the mouse over the top of an object such as a text box.
- Moving from one text box to another.

In order to work with VB, you need to understand "object" terminology as defined below.

- **Object** A thing – like a noun in English like forms and controls you place on forms such as buttons, text boxes, and icons.
- **Property** Objects have properties – like adjectives in English. Properties describe object behaviors. For examples Text, Name, BackColor, Font, and Size. Refer to a property by the notation ObjectName.PropertyName (use the .dot notation) – like TotalTextBox.Text or AccountLabel.ForeColor. Each property has value.
- **Method** Like a verb in English – these are the actions that objects exhibit. For examples methods to Show or Hide forms and methods to Print or Close forms. Refer to a method with the notation ObjectName.MethodName – example Me.Close will close the current form.
- **Event** Events are actions usually triggered by the system user such as clicking a button; however, events can also be triggered by the actions of objects. For example, closing a form can trigger an event.
- **Class** This is a really abstract term – it is a sort of template for an object. For example, all forms belong to the Form class of object. All buttons belong to the Button class of object. Classes include definitions for object properties, methods, and associated events. Each class is assigned an identifying namespace within the .NET Framework Class Library. Each new object you create is defined based on its class – the new object is called a class instance.

❖ Getting Started with Visual Studio

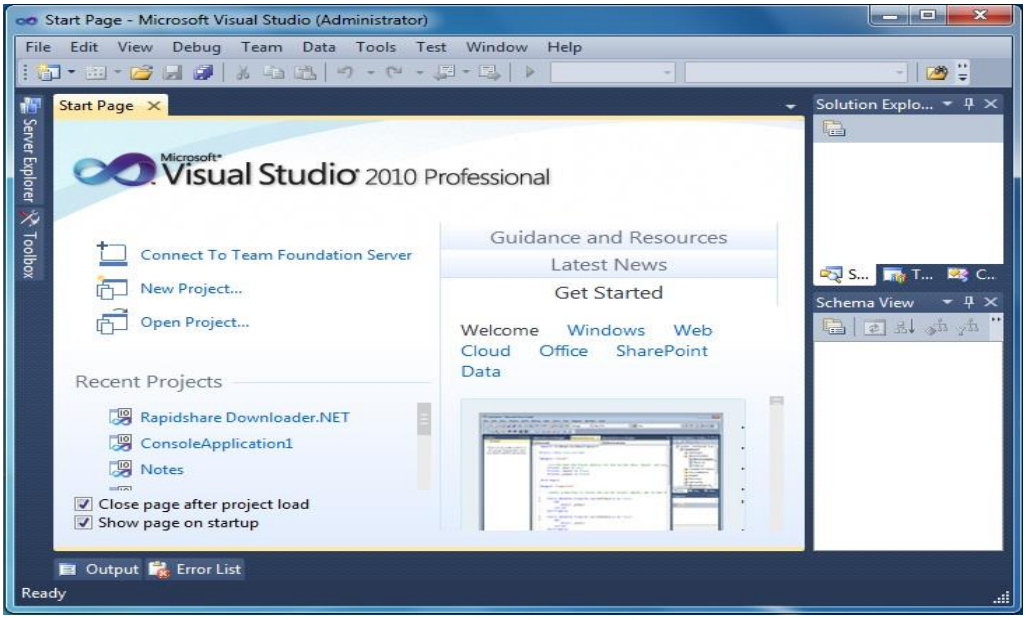
You will use the VB component of Visual Studio to create and test projects.

- In VB, programming applications you will design and develop are called solutions.
- A solution can actually contain more than one project.
- Each solution is stored in a folder identified by the solution name.

➢ Steps to Open VB 2010 and Create a New Project

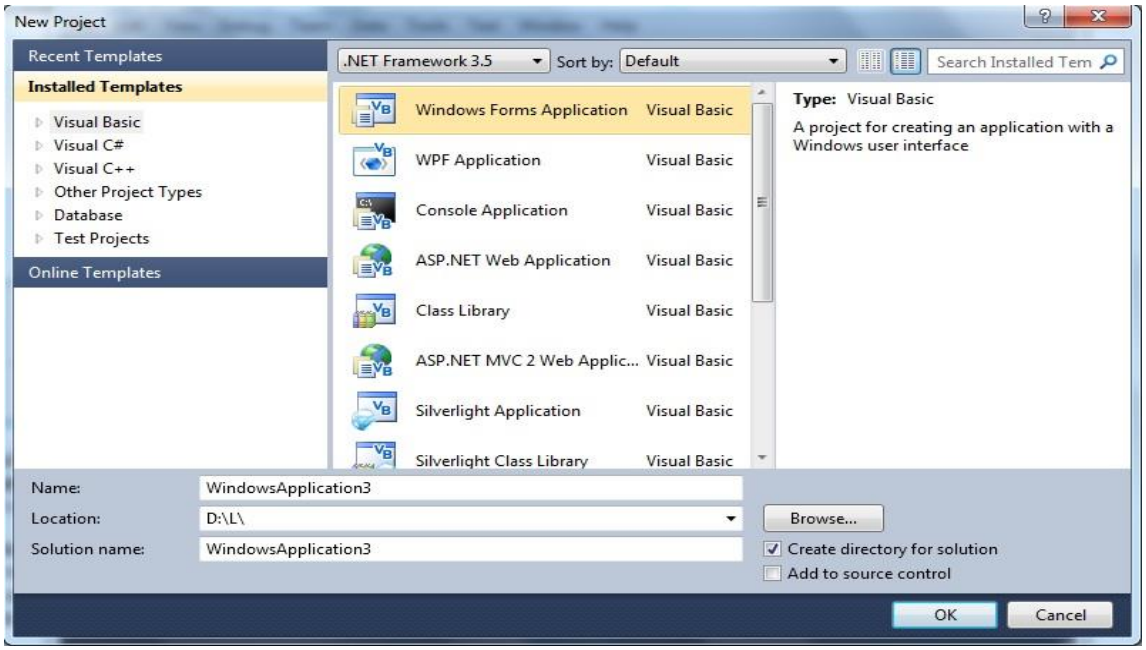
1. Start Visual Studio 2010. Figure1
2. On the *Visual Studio File* menu, click *New Project*, or you can start a new project by clicking the blue *New Project link* on the *Start Page*. The New Project dialog box opens, as shown Figure 2.

Figure1: Start Page for Visual Basic 2010 Express Edition.



The Visual Studio Start Page in this figure above is slightly different for Visual Studio 2010 (Professional or Ultimate Edition) and Visual Basic 2010 Express Edition.

Figure 2



- Your first project will be a Visual Basic Project using a Windows Forms Application template as shown in the figures above.
- The default name is WindowsApplication1 – You can change the project name
- Click the OK button – it takes several seconds to create the project files from the template.

➢ The Integrated Development Environment

Familiarize yourself with the Integrated Development Environment (IDE). Figure 3.

- Across the top are menus with different options used in the designing of an application.
- Toolbars with shortcut icons are shown below the menus.
- Form Designer (also termed the Document Window).
- Solution Explorer Window – displays filenames for files that comprise a project. The solution name is also shown here.
- Properties Window – displays properties of the currently selected object – in the figure the properties displayed are those of the Form object.
- ToolBox Window – this is shown along the left edge of the figure in collapsed display.

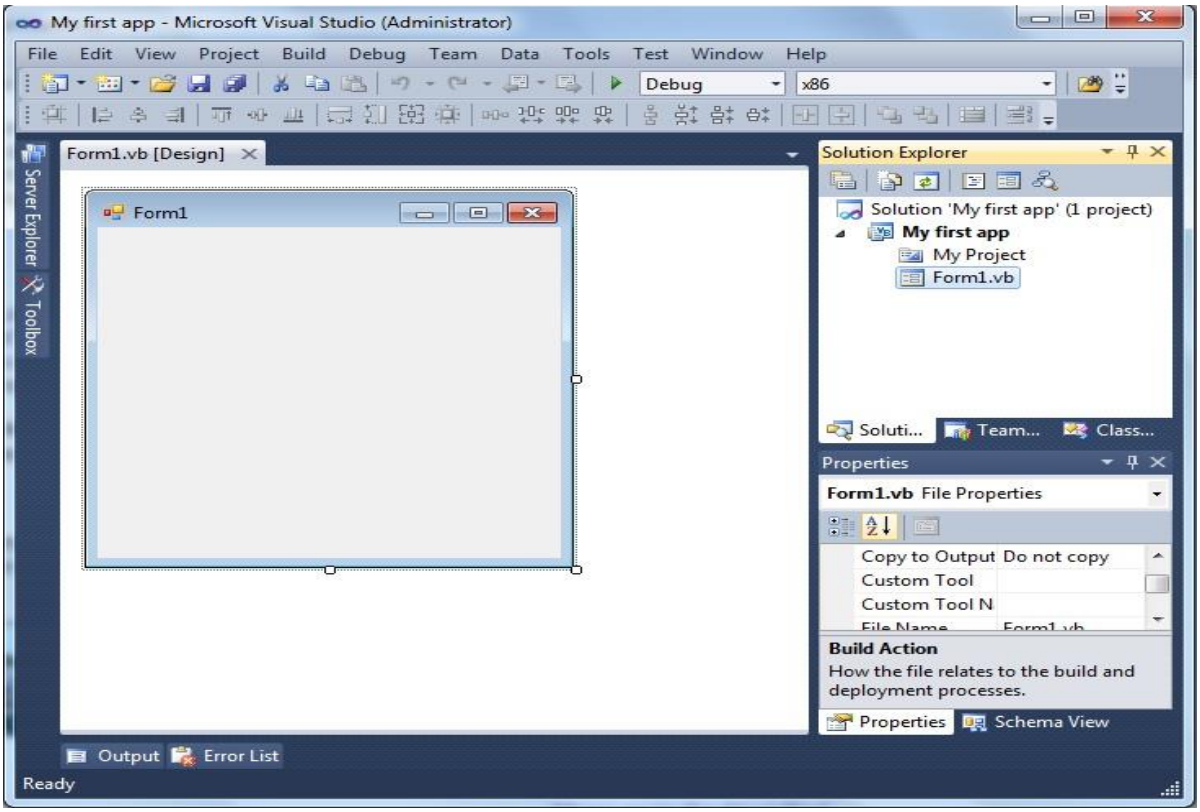


Figure 3

➢ Saving and Running a Project

To save a project, use **File-Save**. AVOID using **File-Save As**. You can click the **Save All** button on the tools bar. This will save all files in the project.

To run project there are several ways. Use the **Debug menu**, **Start Debugging** option, or press **F5 function key** to run the project, or **Click the shortcut green arrow** on the shortcut toolbar.

➢ Naming Rules and Conventions

Visual Basic automatically assigns a value to the Name property of each control by default, for example, Label1, Label2 or TextBox1, TextBox2 or Button1, Button2, etc. However, it is difficult to remember the difference between them so, it is best to rename them to a more meaningful name.

- An object name can begin with an uppercase letter or with "underscore" character.
- An object name can include letters, digits, and underscores.
- An object name CANNOT include a space or a punctuation mark.
- An object name CANNOT be a VB reserved word such as Button, Close, etc.
- Select object names that are meaningful.
- There are different types of rules for writing name of an object name. In this course we will use Hungarian Naming rules.

Figure 4: Examples of valid Hungarian Naming Conventions:

		Hungarian Naming Conventions
Form	Frm	FrmMain, FrmStudent, FrmTeachers
TextBox	Txt	txtName, txtMajor
Button	Btn	btnShipping, btnExit, btnReset
Label	Lbl	lblName, lblOutput
Listbox	Lst	LstName
Image	Img	ImgBox
Option Button	Opt	OptSelect

➢ Adding Program Code

Most VB objects may have code (programs) attached to it. When the user clicks on the object, VB will execute this program. *Double click on any objects on the form (at design time) will transfer system to another window (View Code Window or Editor Window).* You can also access the code view from the *View menu or the View icon in the Solution Explorer window*. Each procedure should begin with:

Private Sub object name_Events name ( ) …… Header Line

VB commands and statements here ……

End Sub

Notes :

* Commands are the lines of code you type in order to perform what you desire.
* A header line begins with optional keyword **Private** and must contain the keyword **Sub**. Private Sub & End Sub are key words automatically included by VB.
* For event procedures, the name consists of *object identification*, an underscore *(_)*, a *valid event* for the specific object, **parameters** between parentheses, the keyword *Handle* followed by *object identification*, *dot Character*, and a *valid event*.
* A Remark statement is a line of code shown in green. It is used to document your program code. Remarks are very useful for large programs. Remark statements are non-executable. In VB you can begin your comments or any remark statement with an apostrophe (‘).
* Often you will need to switch between View Designer and View Code Windows. **Click** one of the tabs to display either the View Designer or View Code windows, **Click** the View Code icon in the Solution Explorer to open the coding window, **Click** the View Designer icon in the Solution Explorer to display the form for additional design layout, or **Click** the View menu and select either the Code or Designer submenu options.
* VB also provides an Auto List Members feature (Editor-Intellisense). This feature presents a drop-down list of available members (such as properties and methods). It is use to accelerating your typing and help you when you aren’t sure which property or method are available for a specific control.

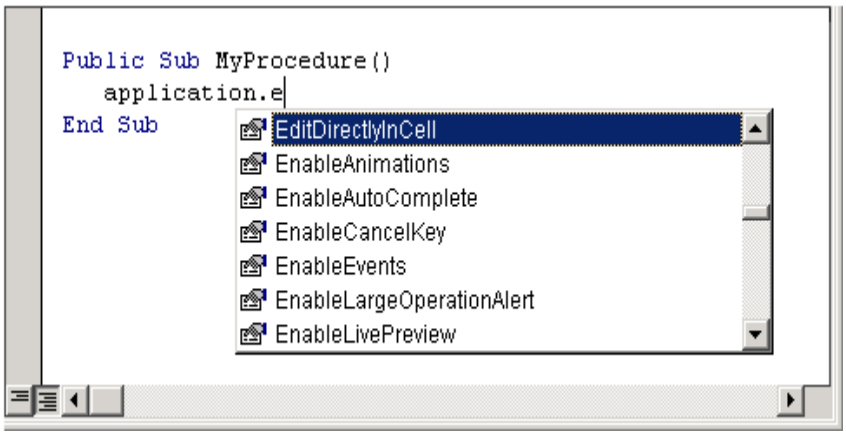


Figure 5

Chapter 2 Writing Software

❖ Identifiers

Identifiers are names given to namespaces, types (enumerations, structures, classes, standard modules, etc.), type members (methods, events, constants, and properties), and variables. Identifiers must begin with:

- An alphabetic or underscore character (_).
- May be of any length
- After the first character must consist of only alphanumeric and underscore characters.
- Namespace declarations may be declared either with identifiers or qualified identifiers. Qualified identifiers consist of two or more identifiers connected with the dot character (.). Only namespace declarations may use qualified identifiers.
- It should not be a reserved keyword. Ordinarily, identifiers may not match Visual Basic .NET keywords. If it is necessary to declare or use an identifier that matches a keyword, the identifier must be enclosed in square brackets ([]). For example :

Public Class [Public]

❖ Literals

Literals are representations of values within the text of a program. For example,**" x = y * 10"** where 10 is a literal, but x and y are not.

There are many types of literals.

➢ Numeric Literals

Represents any number within specific range of either ***Integer Literals*** (like: byte, short integer, or long integer) by default Visual Basic .NET interprets integer literals as type of integer, or ***Floating Point Literals*** (like: Single, Double, or Decimal). By default Visual Basic .NET interprets floating point literals as type of Double.

➢ String Literals

Literals of type **String** consist of characters enclosed within quotation-mark (**"**) characters. For example, in the following line of code, "hello world" is a literal of type String:

➢ Character Literals

Visual Basic .NET's **Char type** represents a single character. This is not the same as a one-character string; Strings and Chars are distinct types. Literals of type Char consist of a single character enclosed within quotation-mark characters.

➢ Date Literals

Literals of type Date are formed by enclosing a date/time string within number-sign characters. For example:

Dim MyDate As Date MyDate = #11/15/2001 3:00:00 PM#

Date literals in Visual Basic .NET code must be in the format m/d/yyyy, regardless of the regional settings of the computer on which the code is written.

➢ Boolean Literals

The keywords **True** and **False** are the only Boolean literals. They represent the true and false Boolean states. For example:

Dim MyBoolean As Boolean MyBoolean = True

➢ Nothing

There is one literal that has no type: the **keyword Nothing**. Nothing is a special symbol that represents an uninitialized value of any type. It can be assigned to any variable and passed in any parameter. When used in place of a value type, it represents an empty value of that type. For numeric types, this is 0. For the String type, this is the empty string (""). For the Boolean type, this is False.

❖ Variables and Constants

In VB as in other programming language, a program consists of **Instructions** that tells the computer what to do and **Data** that the program uses when it is running. The Data consists of **Constants** or *fixed values that never change* and **Variable** *store values during a program's execution*. Usually, both constants and variables are defined as certain **Data Types**. A variable has **Name** and **Value** which can be change depending on conditions or on information passed to the program**.** When you declaring variable you should be aware of naming conventions as mentioned previously.

➢ Types of Variables

VB.Net provides a wide range of data types. The following table shows the fundamental VB.NET types:

Data Type	Storage Allocation	Value Range
Boolean		**True** or **False**
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)

Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/- 79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/- 7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values
String	Depends on implementing platform	0 to approximately 2 billion Unicode characters

➢ Variable Declaration

The Variable declaration involves giving the variable a name and defining the data type to which it belongs. **Dim** statement is used for variable declaration and storage allocation for one or more variables. Dim is used at module, class, structure, procedure, or block level. You can use the word Dim with one of the following options:

[Shared] [Static] [ReadOnly] Dim [withEvents]

Where,

Shared: declares a shared variable, which is not associated with any specific instance of class or structure. **Optional**
Static: Variable will retain its value, even when the procedure terminated. **Optional**
ReadOnly: Variable can be read but not written. **Optional**
WithEvents: Variable is used to respond to events raised by the instance assigned to the variable. **Optional**
For declaring variables use the following syntax:

Dim Variable-Name boundslist As New data type = initializer

Where,
Variable Name: is name of variable.
Boundlist: [optional]. Is provides list of bounds of each dimension of an array variable.
New: [optional]. It creates a new instance of the class when the Dim statement runs.
Data type: Required if option strict is on. It specifies the data type of the variable.
Initializer: [optional if New is not specified]. It represents the value that assigned to the variable when it is created.
Example:

Static Dim x As Integer

In the above example, 'x' is the variable name while Integer is the data type to which variable x belongs.
Vb.Net also allows defining other value type of variable like **Enum** and reference types of variables like **class.**

➢ Constant Declaration

The constants refer to the fixed values that the program may not alter during its execution that means can't modify after definition. These fixed values are also called **Literals.** Constant can be of any of the basic data types like an integer, double, char or string literals. For declaring constants use the following syntax:

Const ConstName As DataType = Initializer

Where
ConstName: Name of constant.
DataType: Type of constant.
Initializer: The value assigned to the constant.
Example:

Public Const message1 As String = "Hello…"

➢ Variable Initialization

Initializing a variable means assigning a value to the variable. The following example demonstrates this:

```
Dim x As Integer
   x = 10
```

Here is another example:

```
Dim name As String
   name = "John"
Dim checker As Boolean
   checker = True
Static Dim flag As Boolean = False
Public Dim ch1 As Char = “A”
```

➢ Declaring Enumerations

The **Enum** statement declares an enumeration and defines the values of its members. The Enum statement can be used at the module, class, structure, procedure, or block level. The syntax for the Enum statement is as follows:

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

where

- ***Enumerationname***: Name of the enumeration. **Required**
- ***Datatype***: Data type of the enumeration and all its members. **Optional**
- ***Memberlist***: List of member constants being declared in this statement. **Required**.

Each member in the memberlist has the following syntax and parts:

MemberName [= initializer]

- ***Initializer*** − value assigned to the enumeration member. **Optional**.

Example:

```
Enum  Colors
    Red = 1
    Orange = 2
    Yellow = 3
    Azure = 4
    Blue = 5
    Violet = 6
End Enum
```

➢ Type of Conversion Functions

There are functions that we can use to convert from one data type to another. The following table includes some of them.

Function Name	Functions Description
CBool	Converts the expression to Boolean data type.
CByte	Converts the expression to Byte data type.
CChar	Converts the expression to Char data type.
CDate	Converts the expression to Date data type
CDbl	Converts the expression to Double data type.
CDec	Converts the expression to Decimal data type.
CInt	Converts the expression to Integer data type.
CLng	Converts the expression to Long data type.
CShort	Converts the expression to Short data type.
CSng	Converts the expression to Single data type.
CStr	Converts the expression to String data type.

➢ Print and Display Constant

Constant	Description
vbCrLf	Carriage return / linefeed character combination.
vbCr	Carriage return character.
vbLf	Line feed character.
vbNewLine	New line character.
vbTab	Tab character.
vbBack	Backspace character.

➢ The Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. VB.Net is rich in built-in operators and provides following types of commonly used operators:

- Arithmetic Operators
- Comparison Operators
- Logical Operators
- Assignment Operators

Arithmetic Operators

Following table shows all the arithmetic operators supported by VB.Net. Assume variable **A** holds 2 and variable **B** holds 7, then −

Show Examples

Operator	Description	Example
^	Raises one operand to the power of another	B^A will give 49
+	Adds two operands	A + B will give 9
-	Subtracts second operand from the first	A - B will give -5
*	Multiplies both operands	A * B will give 14
/	Divides one operand by another and returns a floating point result	B / A will give 3.5
\	Divides one operand by another and returns an integer result	B \ A will give 3
MOD	Modulus Operator and remainder of after an integer division	B MOD A will give 1

Comparison Operators

Following table shows all the comparison operators supported by VB.Net. Assume variable **A** holds 10 and variable **B** holds 20, then −

Show Examples

Operator	Description	Example
=	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A = B) is not true.
<>	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A <> B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Logical Operators

Following table shows all the logical operators supported by VB.Net. Assume variable A holds Boolean value True and variable B holds Boolean value False, then −

Show Examples

Operator	Description	Example
And	It is the logical as well as bitwise AND operator. If both the operands are true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A And B) is False.
Or	It is the logical as well as bitwise OR operator. If any of the two operands is true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A Or B) is True.
Not	It is the logical as well as bitwise NOT operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	Not(A And B) is True.
IsFalse	It determines whether an expression is False.	
IsTrue	It determines whether an expression is True.	

Assignment Operators

There are following assignment operators supported by VB.Net −

Show Examples

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (floating point division)	C /= A is equivalent to C = C / A
\=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (Integer division)	C \= A is equivalent to C = C \A
^=	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.	C^=A is equivalent to C = C ^ A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.	Str1 &= Str2 is same as Str1 = Str1 & Str2

Chapter 3 Working with Controls

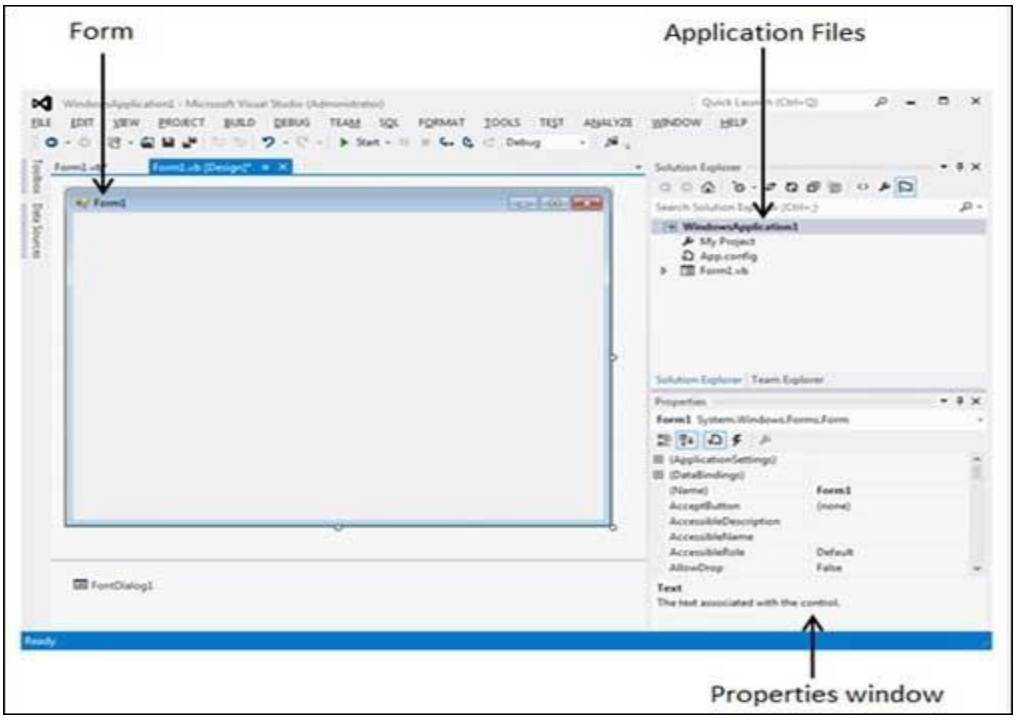
❖ Designing the Form

➢ Form Definition

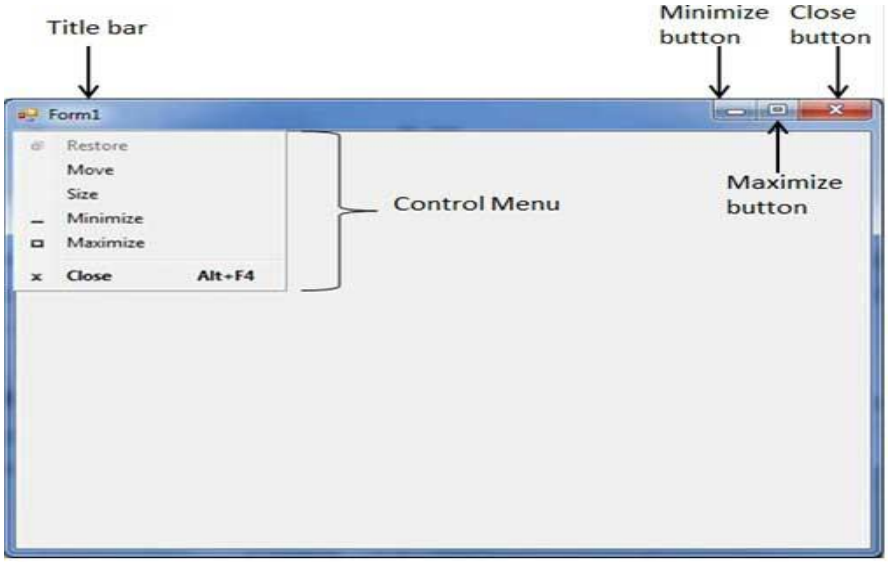
A Form is a container upon which controls are placed. When an application is executed, the form becomes a window. This graphical interface allows the user to see the Input and Output provides by application. The GUI is instructed by placing a set of visual objects on a blank window or Form. Let's start with creating a **Window Forms Application** by following the following steps in Microsoft Visual Studio:

File → New Project → Windows Forms Applications

Microsoft Visual Studio creates your project and displays following window Form with a name **Form1**.



Every form will have title bar on which the form's caption is displayed and there will be buttons to close, maximize and minimize the form shown below. If you click the icon on the top left corner, it opens the control menu, which contains the various commands to control the form like to move control from one place to another place, to maximize or minimize the form or to close the form.



➢ Form Events

Following table lists down various important events related to a form.

Sr.No.	Event	Description
1	**Activated**	Occurs when the form is activated in code or by the user.
2	**Click**	Occurs when the form is clicked.
3	**DoubleClick**	Occurs when the form control is double-clicked.
4	**Enter**	Occurs when the form is entered.
5	**KeyDown**	Occurs when a key is pressed while the form has focus.
6	**KeyPress**	Occurs when a key is pressed while the form has focus.
7	**KeyUp**	Occurs when a key is released while the form has focus.
8	**Load**	Occurs before a form is displayed for the first time.

➢ Form Properties

Following table lists of important properties related to a form. These properties can be set or read during application execution.

Properties	Description
1- BackColor	Sets the form background color.
2- Enabled	If True, allows the form to respond to mouse and keyboard events; if False, disables form.
3- Font	This property specify font type, style, size
4- Height	This is the height of the Form in pixels.
5- Width	This is the width of the form in pixel.
6- MinimizeBox	By default, this property is True and you can set it to False to hide the Minimize button on the title bar.
7- MaximizeBox	By default, this property is True and you can set it to False to hide the Maximize button on the title bar.
8- Text	The text, which will appear at the title bar of the form.
9- StartPosition	This property determines the initial position of the form when it's first displayed. It will have any of the following values − • **CenterParent** − The form is centered in the area of its parent form. • **CenterScreen** − The form is centered on the monitor. • **Manual** − The location and size of the form will determine its starting position. • **WindowsDefaultBounds** − The form is positioned at the default location and size determined by Windows. • **WindowsDefaultLocation** − The form is positioned at the Windows default location and has the dimensions you've set at design time.

➢ Form Methods

The following are some of the commonly used methods of the Form class.

Sr.No.	Method Name & Description
1	**Activate :-** Activates the form and gives it focus.
2	**ActivateMdiChild** :- Activates the MDI (Multi Documents Interface) child of a form.
3	**Close :-** Closes the form.
4	**Focus :-** Sets input focus to the control.
5	**Hide :-** Conceals the control from the user.
6	**Refresh :-** Forces the control to invalidate its client area and immediately redraw itself and any child controls.
7	**Select :-** Activates the control.
8	**Show :-** Displays the control to the user.
9	**ShowDialog :-** Shows the form as a modal dialog box.

❖ What is Controls

Controls are objects that can be inserted into the **Form** of the VB 2010 IDE for various purposes. These controls help in creating a GUI Based Applications in VB.Net quickly and easily. You can drag any control to the **Form** using the Control toolbox in the IDE. Each Control has some **properties**, **events**, and **methods.** You can write relevant code for them to perform certain tasks.

- **Properties** describe the object
- **Methods** are used to make the object do something
- **Events** describe what happens when the user/Object takes any action.

Once you have added a VB.NET control to the form, you can change its appearance, its text, its default values, position, size, etc. using its properties. The properties can be changed via the ***Properties pane*** or by adding the specific values of properties into the ***Code editor***. Using the following syntax:

Object. Property = Value

➢ Common Controls in VB.NET

VB.NET has a variety of controls, below given are the commonly used controls.

- ***Text Box***

It is used to accept textual input from the user. The user can add strings, numerical values and a combination of those, but Images and other multimedia content are not supported. The following are some of the commonly used properties, Methods, and Events of the TextBox control.

Sr.No.	Property & Description
1	**AcceptsReturn** Gets or sets a value indicating whether pressing ENTER in a multiline TextBox control creates a new line of text in the control or activates the default button for the form.
2	**CharacterCasing** Gets or sets whether the TextBox control modifies the case of characters as they are typed.
3	**Font** Gets or sets the font of the text displayed by the control.
4	**Font.Height** Gets or sets the height of the font of the control.
5	**ForeColor** Gets or sets the foreground color of the control.
6	**Multiline** Gets or sets a value indicating whether this is a multiline TextBox control.
7	**PasswordChar** Gets or sets the character used to mask characters of a password in a single-line TextBox control.
8	**ReadOnly** Gets or sets a value indicating whether text in the text box is read-only. *for displaying the results
9	**TabIndex** Gets or sets the tab order of the control within its container.
10	**Text** Gets or sets the current text in the TextBox.
11	**TextLength** Gets the length of text in the control.

12	**TextAlign** Gets or sets how text is aligned in a TextBox control. This property has values 1- Left 2- Right 3- Center
13	**WordWrap** Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.

Sr.No.	Method Name & Description
1	**AppendText** Appends text to the current text of a text box.
2	**Clear** Clears all text from the text box control.
3	**ResetText** Resets the Text property to its default value.
4	**Focus** Set input focus to the control.

Sr.No.	Event & Description
1	**Click** Occurs when the control is clicked.
2	**DoubleClick** Occurs when the control is double-clicked.
3	**TextAlignChanged** Occurs when the TextAlign property value changes.

TextBox is Change الحدث الافتراضي لل

- ***Label***

It is used to show any text to the user, typically the text in a label does not change while the application is running.

- ***Button***

It is used as a standard Windows Button. In most cases, the Button Control is used to generate a click event, its name, size and appearance are not changed in the runtime. The following are some of the commonly used Properties, Methods, and Events of the Button control.

Sr.No.	Property & Description
1	**AutoSizeMode** Gets or sets the mode by which the Button automatically resizes itself.
2	**BackColor** Gets or sets the background color of the control.
3	**BackGroundImage** Gets or sets the image that is displayed on a button control.
4	**TabIndex** Gets or sets the tab order of the control within its container.
5	**Text** Gets or sets the text associated with this control.

Sr.No.	Method Name & Description
2	**NotifyDefault** Notifies the Button whether it is the default button so that it can adjust its appearance accordingly.(True, False)
3	**Select** Activates the control. It will response to keybord & mouse

Use the same Events as in TextBox

Example:

```
Public Class Form1
    Private Sub ButtonExmaple_Load( )
        Me.Text = "educuba.com"
    End Sub
    Private Sub quitBTN _Click( )
        Application.Exit()
     End Sub
End Class
```

- ***ListBox***

As the name suggests, this control works as a way to display a list of items on the application. It allows adding items at design time by using the properties or at the run time by using some methods. List Boxes are best used for displaying large

number of choices. There are situations in which you can display choices with either a group of check boxes or a list boxes. In general, use a group of check boxes when the number of choices is small. Users can select any options from the list. The following are some of the commonly used properties, Methods, and Events of the ListBox control.

S.No	Property & Description
1	**Locked {Both}** Determined if we can move or resize the control (True, False).
2	**DropDownStyle {Combo}** Show the style of comboBox (simple, DropDown, DropDownList).
3	**Items {Both}** Gets the items of the list box.
4	**MultiColumn {ListBox}** Gets or sets a value indicating whether the list box supports multiple columns.
5	**ScrollAlwaysVisible {Listbox}** Gets or sets a value indicating whether the vertical scroll bar is shown at all times.
6	**SelectionMode {Listbox}** Gets or sets the method in which items are selected in the list box. This property has values – doesn't found in the combo box 1- None 2- One 3- MultiSimple 4- MultiExtended
7	**Sorted {Both}** Gets or sets a value indicating whether the items in the list box are sorted alphabetically.
8	**Text {Combo}** Gets or searches for the text of the currently selected item in the list box.

Sr.No.	Method Name & Description
1	**GetItemText** Return the text according to the specified item.
2	**Add** Insert item at the end of list.
3	**Remove** Remove items from the list
4	**Remove At** Remove the selected item or item in the specific location.

5	**SetSelected** Selects or clears the selection for the specified item in a ListBox.(No, T or F)
6	**Count** Return Number of items in the list
7	**Clear** Delete all the items in the list.

Sr.No.	Event & Description
1	**Click** Occurs when the control is clicked.
2	**DoubleClick** Occurs when the control is double-clicked.
3	**SelectedValueChane** Occurs when you move between items.
	SelectedIndexChanged Occurs when you select item

Example:

```
Private Sub dropexmaple_Load( )
    ListBox1.Items.Add("India")
    ListBox1.Items.Add("Pakistan")
    ListBox1.Items.Add("USA")
End Sub
Private Sub BTN1_Click( )
    MsgBox("The country you have selected is " +
    ListBox1.SelectedItem.ToString())
End Sub
Private Sub ListBox1_SelectedIndexChanged( )
    Label1.Text = ListBox1.SelectedItem.ToString()
 End Sub
```

- ***Combo Box***

It is similar to the ListBox control in the sense that it contains multiple items and user may select one, but typically it occupies less space on form because it works as a dropdown for the user. Combo Box is practically an expandable ListBox that can grow when the user want to add new items which doesn't found in the list and react after the selection is made, that means a user can specify items don't exist in the list by enter new item in the box **OR** he can click on the downwards on the

right side and select any item. Combo Box control doesn't allow multiple-item selection.

Example:

```
Private Sub Form2_Load( )
    ComboBox1.Items.Add("India")
    ComboBox1.Items.Add("Pakistan")
    ComboBox1.Items.Add("USA")
End Sub
Private Sub Button1_Click( )
     'Label1.Text = ComboBox1.Text
     MsgBox(ComboBox1.Text)
End Sub
```

Note: Most of Properties, Methods, and Events are applying for Combo Box.

- ***Radio Button***

Radio Button is one of the popular ways of limiting the user to pick just one option. The programmer can set any of the buttons as default if needed. These buttons are grouped together. The following are some of the commonly used properties, Methods, and Events of the Radio Button control.

Sr.No.	Property & Description
1	**Appearance** Determine the appearance of the radio button.(Normal, Button)
2	**AutoCheck** Gets or sets a value indicating whether the Checked value and the appearance of the control automatically change when the control is clicked.
3	**CheckState** Indicate the state of component. (Unchecked, Checked, Indeterminate)
4	**Checked** Returns either true or false and indicates or sets a value indicating whether the control is checked.
5	**Text** Gets or sets the caption for a radio button.

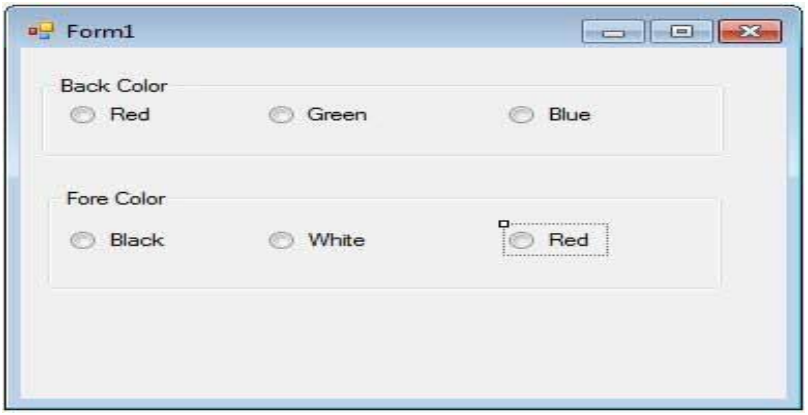
Sr.No	Event & Description
1	**AppearanceChanged** Occurs when the value of the Appearance property of the Radio Button control is changed.

2	**CheckedChanged** Occurs when the value of the Checked property of the RadioButton control is changed.

Sr.No.	Method Name & Description
1	**PerformClick** Generates a Click event for the control, simulating a click by a user.

Example:

In the following example, let us create two groups of radio buttons and use their CheckedChanged events for changing the BackColor and ForeColor property of the form.



Let's double click on the radio buttons and put the follow code in the opened window.

```
Public Class example
   Private Sub Form1_Load( )
       Me.Text = "educba.com"
   End Sub
   Private Sub Example_RadioButton1_CheckedChanged( )
       Me.BackColor = Color.Black
   End Sub
   Private Sub Example_RadioButton2_CheckedChanged( )
       Me.BackColor = Color.White
   End Sub
   Private Sub Example_RadioButton3_CheckedChanged( )
       Me.BackColor = Color.Brown
    End Sub
End Class
```

- ***Checkbox***

Checkboxes are similar to radio buttons in the way that they are also used in groups however, a user can select more than one item in the group. It takes most of the same Events, Properties, and Methods.

Example:

```
Public Class Form1
    Private Sub Submit_Click( )
        Dim msg As String = ""
        If ExampleCheckBox1.Checked = True Then
            msg = " ExampleCheckBox1 Selected"
        End If
        If ExampleCheckBox2.Checked = True Then
            msg = msg & "  ExampleCheckBox2 Selected "
        End If
        If ExampleCheckBox3.Checked = True Then
            msg = msg & ExampleCheckBox3 Selected"
        End If
        If msg.Length > 0 Then
            MsgBox(msg & " selected ")
        Else
            MsgBox("No checkbox have beenselected")
        End If
        CheckBox1.ThreeState = True
    End Sub
End Class
```

- ***PictureBox***

This VB.Net control is used to show images and graphics inside a form. The image can be of any supported format and we can select the size of the object in the form too.

```
Private Sub PictureBox1_Click( )
    PictureBox1.ClientSize = New Size(500, 500)
    PictureBox1.SizeMode = PictureBoxSizeMode.StretchImage
End Sub
```

- ***Date Time Picker***

In cases where you need to ask the user about date and time, VB.NET has a readymade control that lets the user pick the date and time via a Calendar and a clock. This saves the hassle of creating multiple text boxes for one input.

Chapter 4 Types of 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.

❖ Declaration Statements

When you declare a programming element, you can also define its data type, access level, and scope. The programming elements you may declare include variables, constants, enumerations, classes, structures, modules, interfaces, procedures, procedure parameters, functions return etc.

Following are the declaration statements in VB.Net −

Sr.No	Statements and Description	Example
1	**Dim Statement** Declares and allocates storage space for one or more variables.	`Dim x As Integer = 22`
2	**Const Statement** Declares and defines one or more constants.	`Const no1 As Long = 10`

3	**Enum Statement**
	Declares an enumeration and defines the values of its members.
		`Enum CoffeeMugSize`
		`  Large`
		`  Medium`
		`End Enum`

4	**Class Statement**
	Declares the name of a class and introduces the definition of the variables, properties, events, and procedures that the class comprises.
		`Class Box`
		`Public lng As Double`
		`Public hght As Double`
		`End Class`

5	**Sub Statement**
	Declares the name, parameters, and code that define a Sub procedure.
		`Sub muSub(By Val s As String)`
		`  Return`
		`End Sub`

6	**Structure Statement** Declares the name of a structure and introduces the definition of the	`Structure Box` `Public length As Double`

	variables, properties, events, and procedures that the structure comprises.	Public breadth As Double End Structure
7	**Module Statement** Declares the name of a module and introduces the definition of the variables, properties, events, and procedures that the module comprises.	Public Module myModule Sub Main() Dim user As String = InputBox("What is your name?") MsgBox("User name is" & user) End Sub End Module
8	**Function Statement** Declares the name, parameters, and code that define a Function procedure.	Function myFunction (ByVal n As Integer) As Double Return 5.87 * n End Function

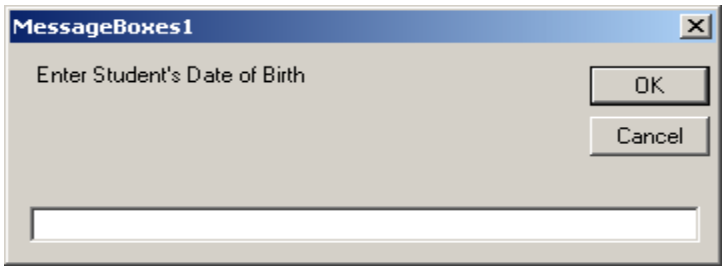
❖ Executable Statements

An executable statement performs an action. Statements calling a procedure, decision statements, looping through several statements, or evaluating an expression are executable statements. An assignment statement is a special case of an executable statement.

➢ Input Operations

In VB2010, there are no explicit instructions or statement for input operation. Vb2010 used some controls to execute the input operation.

1- **InputBox:** An input box is a specially designed dialog box that allows the programmer to request a value (some time huge data values) from the user and use that value as necessary. When calling the **InputBox( )** function, if you pass only the first two arguments, the input box would display the title for the InputBox in the title bar, a prompt message to indicate the requested value, a text box for the user, and two buttons: OK and Cancel.

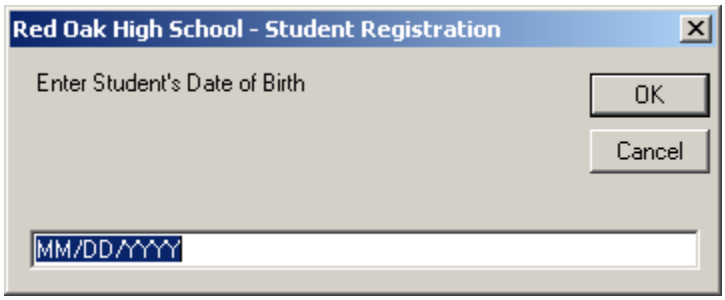


After using the input box, the user can change his or her mind and press **Esc** or **click Cancel**. If the user provided a value and want to acknowledge it, he or she can click **OK** or press **Enter**. This would transfer the contents of the text box to the application that displayed the input box.

To assist the user of type a value that you are expecting, you can give an example by using the third argument as a string. When passing it, you can provide a sample value that the user would follow.

Example:

```
InputBox("Enter Student's Date of Birth", "Red Oak High School - Student Registration", _
    "MM/DD/YYYY")
```



The last two arguments, *XPos* and *YPos*, allow you to specify the default position of the input box when it comes up the first time.

Example:

```
InputBox("Enter Student's Date of Birth", "Red Oak High School - Student Registration", _
    "MM/DD/YYYY", 100, 100)
```

2- TextBox: Is one of the common controls that use to accept data form user. It's good for small amount of input data. The Text Property can be used for both setting and retrieving text. *See CH3.P21 for more information.*

Example:

```
Dim mysalary As Integer
mysalary = Cint (TextBox1.Text)
```

Where,

mysalary: Integer variable
Cint : A function used to convert text value to numeric value to be compatible with the variable "mysalary" which is declared as integer type.
TextBox1: Is a control that used to accept data.
Text : Is the property that used for this purpose.

➢ Output Operations

1- **Label:** It is generally used to display some informative text on the GUI which is not changed during runtime by assigning the value to the ***Text*** property.

2- **Message Boxes:** A message box is a special dialog box used to display a piece of information to the user. To support message boxes, the Visual Basic language provides a ***MsgBox*** function, also the .NET Framework, provide the Show( ) method of the ***MessageBox*** class.

- ***MsgBox:*** It takes one or more arguments and the results of the function can be assigned to a variable. The syntax for MsgBox..

```
MsgBox (Prompt, Buttons, Title)
```

Where, Prompt: Is the text displayed in the message box.
Button: Is a number (0, 1, 2, 3, 4, 5) that specifies the buttons.
Title : Is the text displayed in the message box title bar.

If you create a simple message box by providing only the message, it would appear with only one button labeled OK. If you want the user to be able to make a decision and communicate it to you, provide a second argument. The second argument must be based on the **MsgBoxStyle** enumeration. When it comes to buttons, some members of this enumeration are:

To Display	MsgBoxStyle	Integral Value
OK	**OKOnly**	0
OK Cancel	**OKCancel**	1
Abort Retry Ignore	**AbortRetryIgnore**	2
Yes No Cancel	**YesNoCancel**	3
Yes No	**YesNo**	4
Retry Cancel	**RetryCancel**	5

To use any of these buttons, call the **MessageBoxStyle** enumeration and access the desired combination. Here is an example:

```
MsgBox("Now move to the next step", MsgBoxStyle.OkCancel, “Lesson Objective”)
```

OR

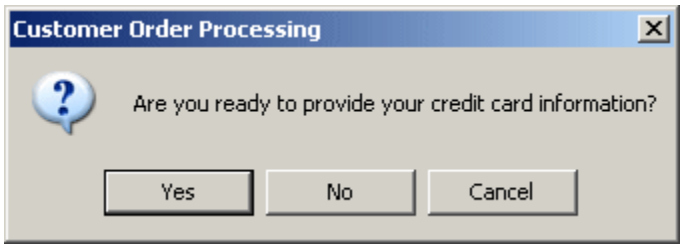
```
MsgBox("Now move to the next step", 1, “Lesson Objective”)
```

To enhance the appearance of a message box, you can display an icon on it. To support icons, the **MsgBoxStyle** enumeration provides the following additional members . To applying one of these buttons, combine its style with that of the button, using the **OR** operator.

To Display	MsgBoxStyle	Integral Value
	**Critical**	16
	**Question**	32
	**Exclamation**	48
	**Information**	64

Example:

MsgBox("Are you ready to provide your credit card information?", MsgBoxStyle.YesNoCancel Or MsgBoxStyle.Question, "Customer Order Processing")

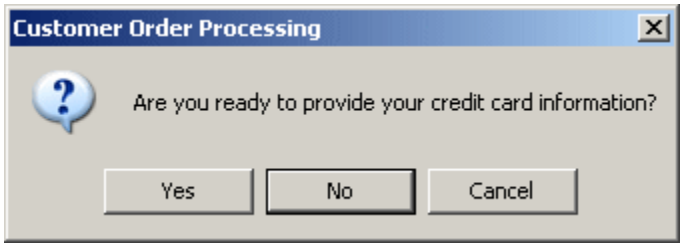


When a message box is configured to display more than one button, the operating system is set to decide which button is the default. The default button has a thick border that sets it apart from the other button(s). If the user presses Enter, the message box would behave as if the user had clicked the default button. If the message box has more than one button, you can decide what button would be the default. To support the default button, the **MsgBoxStyle** enumeration provides the following additional options:

MsgBoxStyle	Integral Value	If the message box contains more than one button, the default button would be
DefaultButton1	0	the first
DefaultButton2	256	the second
DefaultButton3	512	the third

Example:

MsgBox("Are you ready to provide your credit card information?", MsgBoxStyle.YesNoCancel Or MsgBoxStyle.Question Or MsgBoxStyle.DefaultButton2, "Customer Order Processing")



The value returned by a message box corresponds to a button the user would have clicked (on the message box). The return value of the MsgBox( ) function is based on the **MsgBoxResult** enumeration. The buttons and the returned values are as follows:

If the User Clicks	Button Caption	Integral Value
OK	**OK**	1
Cancel	**Cancel**	2
Abort	**Abort**	3
Retry	**Retry**	4
Ignore	**Ignore**	5
Yes	**Yes**	6
No	**No**	7

- ***MessageBox.Show( ):*** It takes one or more arguments. It has the following format:

MessageBox.Show(Text, Caption, Buttons, Icon)

```
Example:
 MessageBox.Show("Welcome to Microsoft VB", ”Program 1”,0,48)
```

Note: we can use **‘&’** operator to join a message and a variable value. The **‘&’** operator can also use to join strings together.

H.W.: Design and write a program to read your name by using InputBox and to display your name as the following example.

Your Name Is Baida’a

H.W.: Design and write a program to display the result of adding three numbers. Use 3 (TextBox to rea d three numbers), 2 (Buttons one for Exit and the other for Add), 1 (Label to display the result). Change back color and for color for all controls and use any picture as a back color for form.

➢ Decision Statements

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

VB.Net provides the following types of decision making statements. Click the following links to check their details.

Statement	Description
If ... Then	**If...Then statement** consists of a boolean expression followed by one or more statements.
If...Then...Else	An **If...Then statement** can be followed by an optional **Else statement**, which executes when the boolean expression is false.
nested If	You can use one **If** or **Else if** statement inside another **If** or **Else if** statement(s).
Select Case	A **Select Case** statement allows a variable to be tested for equality against a list of values.

1. If … Then Statement

It is the simplest form of control statement, frequently used in decision making and changing the control flow of the program execution. Syntax for if-then statement is −

```
If condition Then
    [Statement(s)]
End If
```

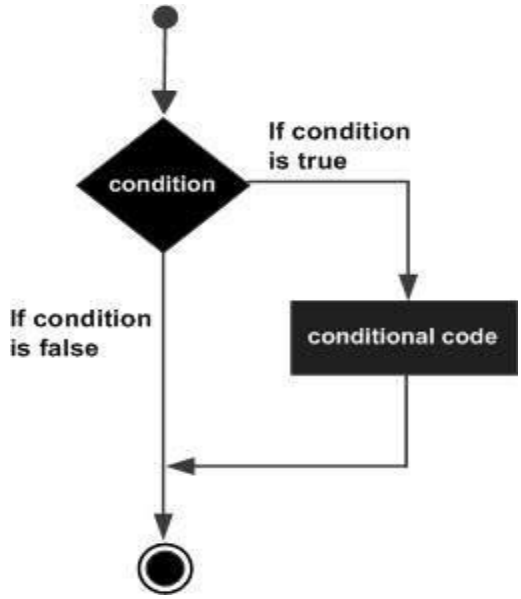
Where,

Condition is a Boolean or relational condition and *Statement*(s) is a simple or compound statement.

Example :

```
If (a <= 20) Then
    c = c+1
End If
```

If the condition evaluates to true, then the block of code inside the If statement will be executed. If condition evaluates to false, then the first set of code after the end of the If statement (after the closing End If) will be executed.

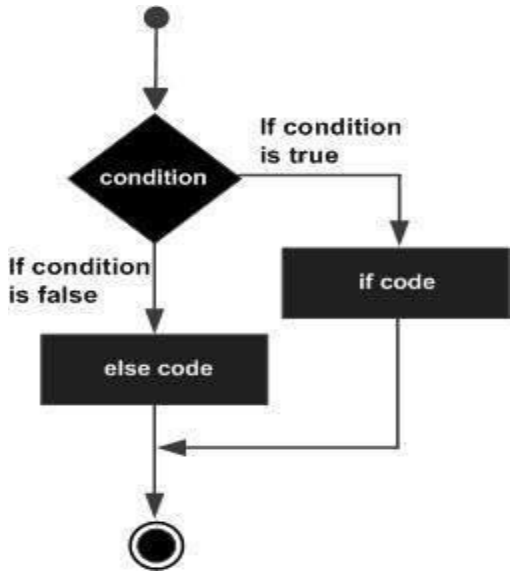


2. If … Then … Else Statement

An **If** statement can be followed by an optional **Else** statement, which executes when the Boolean expression is false. Syntax for if-then-else statement is –

```
If (boolean_ expression) Then
    Statement (s) will execute if then boolean expression is true
Else
    Statement (s) will execute if then boolean expression is false
End If
```

If the Boolean expression evaluates to **true**, then the If block of code will be executed, otherwise Else block of code will be executed.



3. Nested IF

It is always legal in VB.Net to nest If-Then-Else statements, which means you can use one If or ElseIf statement inside another If ElseIf statement(s). The syntax for a nested If statement is as follows −

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

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

Example: The following code, tests the score which is input by the user and display appropriate message according to the score.

```
Private Sub btnTestGrade_ Click( )
    Dim score As Single
    Dim result As String
    score = InputBox ("Enter Score .. ?")
    If score < 50 Then
       result = "Failed"
    Else if score < 70 then
       result = "Pass"
     Else if score < 80 then
       result = "Good"
     Else if score < 90 then
       result = "V.Good"
```

```
        Else
           result = "Excellent"
        End If
        MsgBox (result)
    End Sub
```

4. Select Case Statement

A **Select Case** statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each select case. The syntax for a Select Case statement is as follows –

```
Select [ Case ] expression
    [ Case expressionlist
            [ statements ] ]
    [ Case Else
            [ elsestatements ] ]
End Select
```

Where,

- ***expression*** − is an expression that must evaluate to any of the elementary data type in VB.Net, i.e., Boolean, Byte, Char, Date, Double, Decimal, Integer, Long, Short, Single, String.
- ***expressionlist*** − List of expression clauses representing match values for *expression*. Multiple expression clauses are separated by commas.
- ***statements*** − statements following Case that run if the select expression matches any clause in *expressionlist*.
- ***elsestatements*** − statements following Case Else that run if the select expression does not match any clause in the *expressionlist* of any of the Case statements.

H.W.: Rewrite the above segment of code related to If statement using Select statement.

5. IIf ( ) Function

It is a built in function accepts as an argument an expression and two values, evaluates the expression, and returns the first value if the expression is **true** or the second value if the expression is **false**. The IIf has the following syntax:

```
Iff (expression, True Part, False Part)
```

Iff function is compact notation for simple If statement, and you can use it to shorten If .. Then .. Else statement.

Example: Let's say you want to display one of the strings "CLOSE" or "FAR" depending on the value of distance variable. Instead of a multiple if statement, you can call IIf ( ) Function as follows.

```
Private Sub Button1_Click( )
        Dim distance As Single
        Dim result As String
        distance = InputBox( "Enter Distance   ?")
        result = IIf ( distance >1000, "FAR", "CLOSE")
        MsgBox (result)
End Sub
```

H.W: Read statement grade using an InputBox, test the grade then print "Pass" of "Fail" by using a message box.

Note: Tools that used with selection statements are CheckBox and RadioButtons.

H.W: Design and write a program to read length and width in Centimeter then convert them meter or inch or both. Use appropriate controls in your design.

➢ Loop Statements

A loop statement allows us to execute a statement or group of statements multiple times. VB.Net provides following types of loops to handle looping requirements.

Loop Type	Description
Do Loop	It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.
For...Next	It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.
While... End While	It executes a series of statements as long as a given condition is True.
Nested loops	You can use one or more loops inside any another While, For or Do loop.

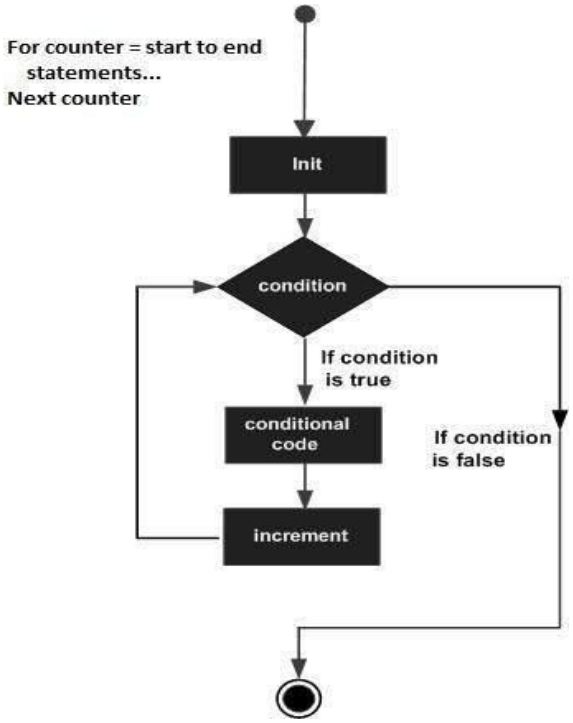
1. For .. Next Loop

It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes. It could be terminated at any time with the Exit For statement.

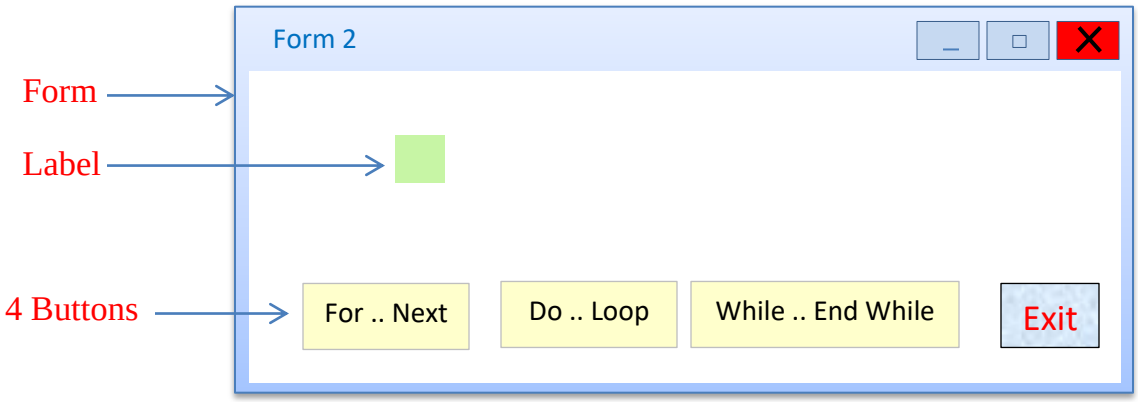
The syntax for this loop construct is −

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

Flow Diagram



Example: Design the following figure then write appropriate codes using VB2010



```
Public Class Form2
    Private Sub Button1_Click(… …)
        Label1.text = “ “
        For i As Integer = 1 To 21 Step 2
            Label1.Text = Label1.Text & i & Space(3)
        Next
    End Sub
End Class
```

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

```
1 3 5 7 9 11 13 15 17 19 21
```

Using Exit For

```
Public Class Form2
    Private Sub Button1_Click( )
        Label1.Text = ""
        For i As Integer = 1 To 21 Step 2
            Label1.Text = Label1.Text & i & Space(3)
            If i >= 15 Then
               Exit For
            End If
        Next
    End Sub
End Class
```

2. Do .. Loop Statement

It repeats the enclosed block of statements while a boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.

The syntax for this loop construct is −

```
Do { While | Until } condition
    [ statements ]
Loop
```

OR

```
Do
    [ statements ]
Loop { While | Until } condition
```

Example: Referencing to the For .. Next example. Do the following modification.

```
Private Sub Button2_Click()
  Dim i As Integer
  i = 1
  Label1.Text = ""
  Do While i <= 21
    Label1.Text = Label1.Text & i & Space(3)
    i = i + 2
  Loop
End Sub
```

```
Private Sub Button2_Click()
  Dim i As Integer
  i = 1
  Label1.Text = ""
  Do Until i > 21
    Label1.Text = Label1.Text & i & Space(3)
    i = i + 2
  Loop
End Sub
```

```
Private Sub Button2_Click()
  Dim i As Integer
  i = 1
  Label1.Text = ""
  Do
    Label1.Text = Label1.Text & i & Space(3)
    i = i + 2
  Loop While i <= 21
End Sub
```

```
Private Sub Button2_Click()
  Dim i As Integer
  i = 1
  Label1.Text = ""
  Do
    Label1.Text = Label1.Text & i & Space(3)
    i = i + 2
  Loop Until i > 21
End Sub
```

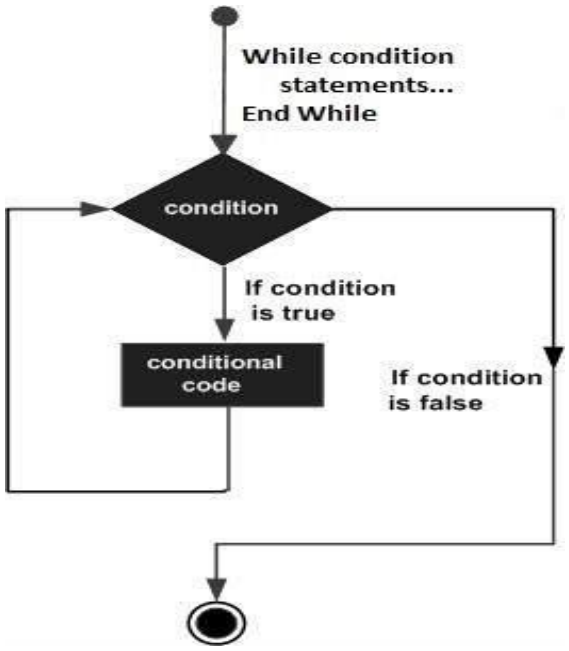
Using Exit Do

```
Public Class Form2
   Private Sub Button2_Click( )
       Dim i As Integer
       i = 1
       Label1.Text = ""
       Do While i <= 21
          Label1.Text = Label1.Text & i & Space(3)
          i = i + 2
          If i >= 15 Then
              Exit Do
          End If
       Loop
   End Sub
End Class
```

H.W: What the expected results for the above code?

3. While .. End While

It executes a series of statements as long as a given condition is True.



The syntax for this loop construct is −

```
While condition
   [ statements ]
End While
```

Here, statement(s) may be a single statement or a block of statements. The condition may be any expression. The loop iterates while the condition is true.

When the condition becomes false, program control passes to the line immediately following the loop.

Note: Here, key point of the *While* loop is that the loop might not ever run. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

H.W: Rewrite one of the above codes related to Do..Loop statement using While statement instead.

4. Nested Loops

VB.Net allows using one loop inside another loop. Following section shows few examples to illustrate the concept.

The syntax for a **nested For loop** statement in VB.Net is as follows −

```
For counter1 [ As datatype1 ] = start1 To end1 [ Step step1 ]
    For counter2 [ As datatype2 ] = start2 To end2 [ Step step2 ]
        Statements
    Next [ counter2 ]
Next [ counter 1]
```

The syntax for a **nested While loop** statement in VB.Net is as follows −

```
While condition1
    While condition2
        Statements
    End While
End While
```

The syntax for a **nested Do...While loop** statement in VB.Net is as follows −

```
Do { While | Until } condition1
    Do { While | Until } condition2
        Statements
    Loop
Loop
```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example, **For-Loop** can be inside a **While Loop** or vice versa.

Note: Tools that used with selection statements are ListBox and ComboBox.

H.W: By using nested loop concepts design and write the proper codes to display a structure of stars "*" in triangle format using label control.

```
*
**
***
****
*****
******
```

❖ Some of Important Functions & Properties & Methods

➢ String Functions

1. **Len:** Return length of the string. Example: Len("Ira q") = 5
2. **Mid:** Return substring containing a specified number of characters.
   Mid(string, position, length)
   Example: mid(" bai da", 3, 2) = ai رمز الفراغ يحتسب من ضمن طول الخيط الرمز
3. **Ltrim:** Return a copy of string without any spaces from left side.
   Example: Left(" bai da ") = "baida "
4. **Rtrim:** Return a copy of string without any spaces from right side.
   Example: Left(" bai da ") = " baida"
5. **Trim:** Return a copy of string without any spaces from both sides left & right
   Example: Left(" bai da ") = "baida"
6. **Ucase & L case:** Ucase convert all characters of string to the capital letters, while the Lcase convert all characters of string to the small letters.
   Ucase("baida") = BAIDA
7. **Space:** used to insert number of spaces as required.
   Space(5)

➢ Math Functions

1. **Fix:** Returns the integer part of the number.
   Example : Fix (7.1234) = 7
2. **Rnd:** Returns a random value between 0 & 1. Random numbers often need to be converted into integers in programming. For example if we wish to obtain a random output of 6 integers ranging from 1 to 6, we need to convert the random numbers to integers using the following formula Int(Rnd*6)+1.

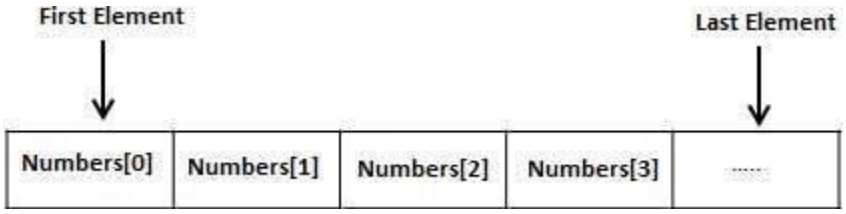
➢ Date & Time

1. **Now:** Returns date & time according to your system
2. **Today:** Returns date according to your system
3. **TimeOfDay:** Returns time according to your system
4. **Month:** Returns the month of any date. Example: Month ("1996/6/25") = 6
5. **Year:** Returns the year of any date. Example: Year (now) = 2020
6. **Day:** Returns the day of any date. Example: Day (today) = 25

❖ The Arrays

➢ One Dimension Arrays

An array stores a fixed-size sequential collection of elements of the same type. It is used to store a collection of data. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Declaring Arrays in VB.Net

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

```
Dim intData(30)                    ' an array of 31 elements
Dim strData(20) As String          ' an array of 21 strings
Dim twoDarray(10, 20) As Integer   'a two dimensional array of integers
Dim ranges(10, 100)                'a two dimensional array
```

You can also initialize the array elements while declaring the array. For example,

```
Dim intData( ) As Integer = {12, 16, 20, 24, 28, 32}

Dim names( ) As String = {"Karth", "Sandy", "Shivangi", "Shwitha", "Somnath"}

Dim Names(2) as String
    Names(0) = "Ali"
    Names(1) = "Ahmed"
    Names(2) = "Sara"
```

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this –

```
Private Sub Button4_Click( )
   Dim n(10) As Integer ' n is an array of 11 integers '
   Dim i, j As Integer
   ' initialize elements of array n '
   For i = 0 To 10
      n(i) = i + 100 ' set element at location i to i + 100
   Next i
   ' Output each array element's value '
   For j = 0 To 10
      Label1.Text = Label1.Text & "Element (" & j & ")" & Space(4) & n(j) &_
      vbNewLine
   Next j
End Sub
```

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

```
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
```

Appendix 1

1. Listbox1.Items.Add (“ALI”) Or (InputBox(“EnterValue ..?”) Or others {it will insert the items at the end of list}
2. Listbox1.Items.Remove (“ALI”) {it will delete specified item}
3. Listbox1.Items.RemoveAt (Listbox1.selectedindex) Or (2) or (Listbox1.items.count-i) Or others {it will delete item after select it
4. Listbox1.Items.count. {retrieve number of items in the list}
5. Listbox1.Items.Insert (i, ”any text”) {It will insert item at the position i}
6. Listbox1.Items.contain (“ any text”) {it will return True if found specific text}
7. Listbox1.sorted = True {it will arrange the list}