What is Microsoft .NET?

- Microsoft .NET (pronounced "dot net") is a software component that runs on the Windows operating system. .NET provides tools and libraries that enable developers to create Windows software much faster and easier. .NET benefits end-users by providing applications of higher capability, quality and security. The .NET Framework must be installed on a user's PC to run .NET applications.
- .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services

What is VB.Net?

- Visual Basic .NET (VB.NET), is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB), which is implemented on the .NET Framework
- It is the next generation of the visual Basic language.
- It supports OOP concepts such as abstraction, inheritance, polymorphism, and aggregation.

.NET Framework Architecture :-

A programming infrastructure created by Microsoft for building, deploying, and running applications and services that use .NET technologies, such as desktop applications and Web services

1) It is a platform for application developers.
2) It is tiered, modular, and hierarchal.
3) It is a service or platform for building, deploying and running applications.
4) It consists of 2 main parts: Common language runtime and class libraries.

- The common language runtime is the bottom tier, the least abstracted.
- The .NET Framework is partitioned into modules, each with its own distinct responsibility.
- The architectural layout of the .NET Framework is illustrated in following figure:

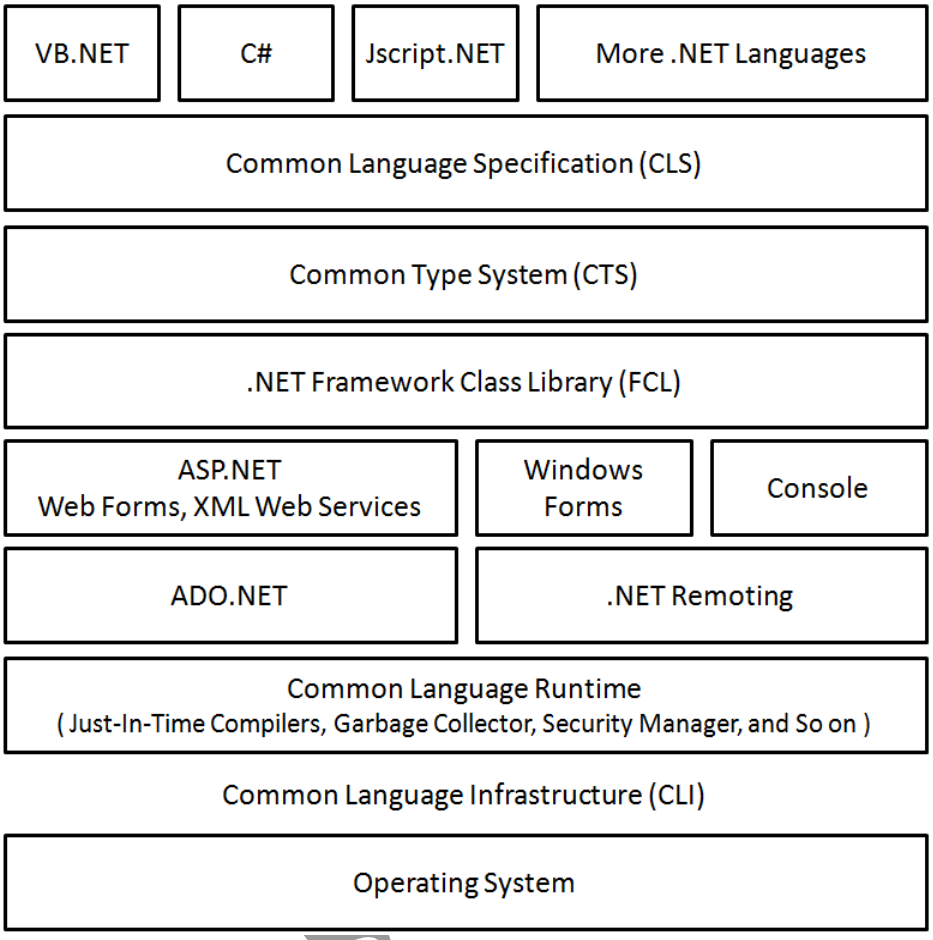

Figure 1 An overview of the .NET architecture.

Here we examine the following key components of the .NET Framework:

1) **Common Language Infrastructure (CLI):** The purpose of the Common Language Infrastructure (CLI) is to provide a language-neutral platform for application development and execution, including functions for Exception handling, Garbage Collection, security, and interoperability.
2) **Common Language Runtime (CLR):** The .NET Framework provides a runtime environment called the Common Language Runtime or CLR (similar to the Java Virtual Machine or JVM in Java), which handles the execution of code and provides useful services for the implementation of the program.

The CLR is the execution engine for .NET applications and serves as the interface between .NET applications and the operating system. The CLR provides many services such as:

- Loads and executes code
- Converts intermediate language to native machine code
- Manages memory and objects
- Enforces code and access security
- Handles exceptions
- Interfaces between managed code, COM objects, and DLLs
- Provides type-checking
- Provides code meta data (Reflection)
- Provides profiling, debugging, etc.
- Separates processes and memory

3) **Framework Class Library (FCL):** It is also known as a base class library. The FCL is a collection of over 7000 reusable classes, interfaces, and value types that enable .NET applications to :
   a) read and write files,
   b) access databases,
   c) process XML,
   d) display a graphical user interface,
   e) draw graphics,
   f) use Web services, etc.

- The .Net Framework class library (FCL) organized in a hierarchical tree structure and it is divided into Namespaces. Namespaces is a logical grouping of types for the purpose of identification. Framework class library (FCL) provides the consistent base types that are used across all .NET enabled languages. The Classes are accessed by namespaces, which reside within Assemblies.
- Other name of FCL is BCL – Base Class Library

4) Common Type System (CTS)

1) CTS allows written in different programming to easily share to information.
2) A class written in C# should be equivalent to a class written in VB.NET.
3) Languages must agree on the meanings of these concepts before they can integrate with one and other.
4) CLS forms a subset of Common type system this implies that all the rules that for apply to Common type system apply to common language specification.
5) It defines rules that a programming language must follow to ensure that objects written in different programming languages can interact which each other.
6) Common type system provide cross language integration.

- The common type system supports two general categories of types:
  a) Value Type
  b) Reference Type

a) **Value Type:** Stores directly data on stack. In built data type. For ex. Dim a as integer.
b) **Reference Type:** Store a reference to the value's memory address, and are allocated on the heap. For ex: dim obj as new oledbconnection.

- The Common Language Runtime (CLR) can load and execute the source code written in any .Net language, only if the type is described in the Common Type System (CTS)

7) Common Language Specification (CLS)

The CLS is a common platform that integrates code and components from multiple .NET programming languages. In other words, a .NET application can be written in multiple programming languages with no extra work by the developer (though converting code between languages can be tricky).

.NET includes new object-oriented programming languages such as C#, Visual Basic .NET, J# (a Java clone) and Managed C++. These languages, plus other experimental languages like F#, all compile to the Common Language Specification and can work together in the same application.

- CLS includes basic Language features needed by almost all the application
- It serves as a guide for Library Writers and Compiler Writer.
- The Common Language Specification is a subset of the common type system (CTS).
- The common language specification is also important to application to who are writing codes that will be used by other developers.
- CLS a series of basic rules that are required for language integration.

8) Microsoft Intermediate Language:

- When you compile your Visual Basic .NET source code, it is changed to an intermediate language (IL) that the CLR and all other .NET development environments understand.
- All .NET languages compile code to this IL, which is known as Microsoft Intermediate Language, MSIL, or IL.
- MSIL is a common language in the sense that the same programming tasks written with different .NET languages produce the same IL code.
- At the IL level, all .NET code is the same regardless of whether it came from C++ or Visual Basic.
- When a compiler produces Microsoft Intermediate Language (MSIL), it also produces Metadata.
- The Microsoft Intermediate Language (MSIL) and Metadata are contained in a portable executable (PE) file.
- Microsoft Intermediate Language (MSIL) includes instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations.

Advantages :

- It offers cross− language integration, including cross− language inheritance, which allows you to create a new class by deriving it from a base class written in another language.
- It facilitates automatic memory management, known as garbage collection.
- compilation is much quicker
- It allows you to compile code once and then run it on any CPU and operating system that supports the runtime.

Disadvantages:

- IL is not compiled to machine, so it can more easily be reverse engineered. Defense mechanisms for handling this are likely to follow shortly after the .NET Framework is officially released.
- While IL is further compiled to machine code, a tiny percentage of algorithms will require a direct unwrapped access to system resources and hardware.

- Figure: 2 shows what happens to your code from its inception in Visual Studio to execution.

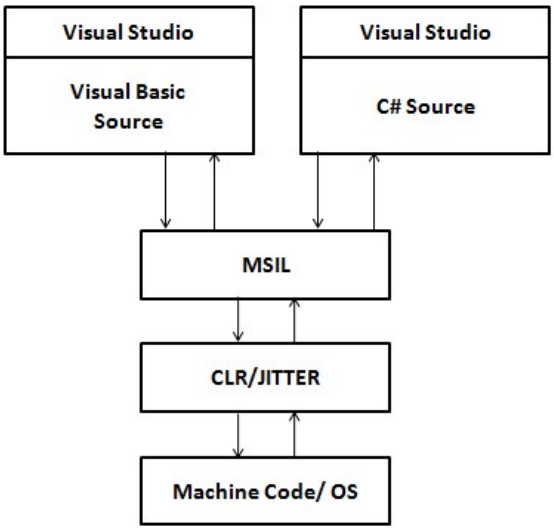

Figure 2 : Following the IL

The Just-In-Time Compiler

- Your code does not stay IL for long, however. It is the PE file, containing the IL that can be distributed and placed with the CLR running on the .NET Framework on any operating system for which the .NET Framework exists, because the IL is platform independent. When you run the IL, however, it is compiled to native code for that platform. Therefore, you are still running native code. The compilation to native code occurs via another tool of the .NET Framework: the Just-In-Time (JIT) compiler.
- With the code compiled, it can run within the Framework and take advantage of low level features such as memory management and security. The compiled code is native code for the CPU on which the .NET Framework is running. A JIT compiler will be available for each platform on which the .NET Framework runs, so you should always be getting native code on any platform running the .NET Framework.
- **Just-in-time compilation** (**JIT**), also known as **dynamic translation**, is a method to improve the runtime performance of computer programs. Historically, computer programs had two modes of runtime operation, either interpreted or static (ahead-of-time) compilation. Interpreted code is translated from a high-level language to a machine code continuously during every execution, whereas statically compiled code is translated into machine code before execution, and only requires this translation once.
- JIT compilers represent a hybrid approach, with translation occurring continuously, as with interpreters, but with caching of translated code to minimize performance degradation. It also offers other advantages over statically compiled code at development time, such as handling of late-bound data types and the ability to enforce security guarantees.
- The Common Language Runtime (CLR) provides various Just In Time compilers (JIT) and each works on a different architecture depending on Operating System. That is why the same Microsoft Intermediate Language (MSIL) can be executed on different Operating Systems without rewrite the source code.

- Just In Time (JIT) compilation preserves memory and save time during initialization of application.
- Just In Time (JIT) compilation is used to run at high speed, after an initial phase of slow interpretation.
- Just In Time Compiler (JIT) code generally offers far better performance than interpreters.
- **There are three types of JIT:**
    1) Pre JIT
    2) Econo JIT
    3) Normal JIT

1) **Pre JIT**: It converts all the code in executable code in a single cycle and it is slow.
2) **Econo JIT:** It will convert the called executable code only. But it will convert code every time when a code is called again.
3) **Normal JIT:** It will only convert the called code and will store in cache so that it will not require converting code again. Normal JIT is fast.

.NET Languages

- .Net languages are CLI computer programming languages that may also optionally use the .NET Framework Base Class Library and which produce programs that execute within the Microsoft .NET Framework. Microsoft provides several such languages, including C#, F#, Visual Basic .NET, and Managed C++.
- Generally .NET languages call into two main categories, TypeSafe Languages (such as C#) and Dynamic Languages (Such as Python). Type Safe Languages are built on the .NET Common Language Runtime and Dynamic Languages are built on top of the .NET Dynamic Language Runtime. The .NET Framework is unique in its ability to provide this flexibility.
- Regardless of which .NET language is used, the output of the language compiler is a representation of the same logic in an intermediate language named Common Intermediate Language (CIL).
- As the program is being executed by the CLR, the CLI code is compiled and cached, just in time, to the machine code appropriate for the architecture on which the program is running. This last compilation step is usually performed by the Common Language Runtime component of the framework "just in time" (JIT) at the moment the program is first invoked, though it can be manually performed at an earlier stage.

Microsoft Intermediate Language (MSIL)

- MSIL or IL(Intermediate Language) is machine independent code generated by .NET framework after the compilation of program written in any language by user.
- MSIL or IL is now known as CIL(Common Intermediate Language).
- One of the more interesting aspects of .NET is that when you compile your code, you do not compile to native code. But the compilation process translates your code into something called Microsoft intermediate language, which is also called MSIL or just IL.
- The compiler also creates the necessary metadata and compiles it into the component. This IL is CPU independent. After the IL and metadata are in a file, this compiled file is called the PE, which stands for either *portable executable* or *physical executable*. Because the PE contains your IL and metadata, it is therefore self-describing, eliminating the need for a type library or interfaces specified with the Interface.

.NET Assembly

- Whatever .NET language you create applications with, compilers generate an *assembly*, which is a file containing .NET executable code and is composed essentially by two kinds of elements: MSIL code and metadata.
- The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file.

- All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. The designers of .NET have worked a lot on the component (assembly) resolution.

The structure of an assembly: Assemblies contain code that is executed by the Common Language Runtime.

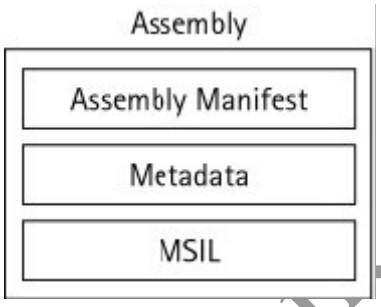

Figure 3 : A diagram of assembly

- Assemblies are made up of the following parts:
  a) The assembly manifest
  b) Type metadata
  c) Microsoft Intermediate Language (MSIL) code
- The assembly manifest is where the details of the assembly are stored. The assembly is stored within the DLL or EXE itself. Assemblies can either be single or multiple file assemblies and, therefore, assembly manifests can either be stored in the assembly or as a separate file. The assembly manifest also stores the version number of the assembly to ensure that the application always uses the correct version.
- The metadata contains information on the types that are exposed by the assembly such as security permission information, class and interface information, and other assembly information.

Contents of an Assembly:

a) Assembly Manifest
b) Assembly Name
c) Version Information
d) Types
e) Cryptographic Hash
f) Security Permissions

An assembly does the following functions:

- It contains the code that the runtime executes.
- It forms a security boundary. An assembly is the unit at which permissions are requested and granted.
- It forms a type boundary. Every type's identity includes the name of the assembly at which it resides.

- It forms a reference scope boundary. The assembly's manifest contains assembly metadata that is used for resolving types and satisfying resource requests.
- It forms a version boundary. The assembly is the smallest version able unit in the common language runtime; all types and resources in the same assembly are versioned as a unit.
- It forms a deployment unit. When an application starts, only the assemblies the application initially calls must be present. Other assemblies, such as localization resources or assemblies containing utility classes, can be retrieved on demand. This allows applications to be kept simple and thin when first downloaded.
- It is a unit where side-by-side execution is supported.

There are two kinds of assemblies in .NET

a) Private
b) Shared

a) Private assemblies is the assembly which is used by application only, normally it resides in your application folder directory.

b) Shared assemblies - It resides in GAC, so that anyone can use this assembly. Public assemblies are always share the common functionalities with other applications.

- An assembly can be a single file or it may consist of the multiple files. In case of multi-file, there is one master module containing the manifest while other assemblies exist as non-manifest modules. A module in .NET is a sub part of a multi-file .NET assembly. Assembly is one of the most interesting and extremely useful areas of .NET architecture along with reflections and attributes, but unfortunately very few people take interest in learning such theoretical looking topics.

The .NET Framework Namespaces

- . Net framework class library is a collection of namespaces.
- Namespace is a logical naming scheme for types that have related functionality.
- Namespace means nothing but a logical container or partition.
- For example: My computer contains C:, D:, E: and F: Each drive contains 1.txt file. The file 1.txt is available in all the drive so it is require to specify the drive name to locate the actual required file.
- At the top of the hierarchy is the System namespace.

- A namespace is just a grouping of related classes. It's a method of putting classes inside a container so that they can be clearly distinguished from other classes with the same name.
- A namespace is a logical grouping rather than a physical grouping. The physical grouping is accomplished by an assembly
- The .NET CLR consists of multiple namespaces, which are spread across many assemblies. For example, ADO.NET is the set of classes located in the System.Data namespace, and ASP.NET is the set of classes located in the System.Web namespace. In the CLR, the classes and structures contained in each of the namespaces represent a common theme of development responsibility.
- .NET Framework class library is collection of namespaces.
- Following table shows Common Namespaces supported by .NET

System	Contains fundamental classes and base classes.
System.IO	Contains classes for reading and writing data in file.
System.XML	Contains classes work with XML.

System.Windows.Forms	Contains classes for windows-based applications.
System.Data	Contains classes for the database connection.

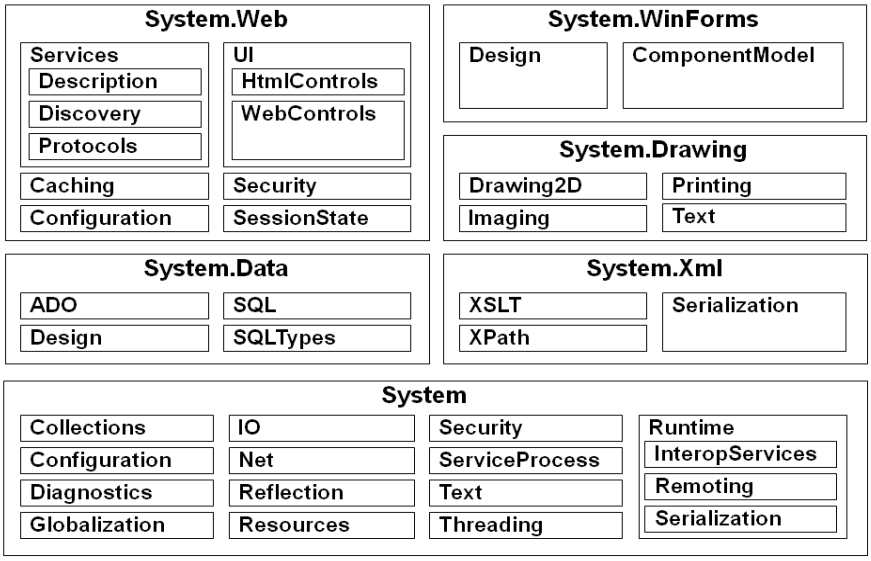

Figure 4: .Net namespaces

VB.NET - Introduction

- Microsoft .NET is a software component that runs on the Windows operating system.
- .NET provides tools and libraries that enable developers to create Windows software much faster and easier. .NET benefits end-users by providing applications of higher capability, quality and security.
- This is how Microsoft describes it: "NET is the Microsoft Web services strategy to connect information, people, systems, and devices through software. Integrated across the Microsoft platform, .NET technology provides the ability to quickly build, deploy, manage, and use connected, security-enhanced solutions with Web services.
- VB .NET is an object-oriented computer programming language that can be viewed as an evolution of the classic Visual Basic (VB), which is implemented on the .NET Framework.
- Visual Basic 2008 version 9.0 was released together with the Microsoft .NET Framework 3.5

Why .NET?

- Interoperability between language and execution environment.
- Uniformity in schema or formats for Data exchange using XML, XSL (Extensible Style Sheet Language)
- Extend or use existing code that is valid.
- Programming complexity of environment is reduced
- Multiplatform applications, automatic resource management simplification of application deployment.
- It provides security like – code authenticity check, resources access authorizations, declarative and imperative security and cryptographic security methods for embedding into user's application.
- The .Net platform is on integral component a new and simplified model for programming and deploying application on the windows platform.

- .Net development framework provides a new and simplified model for programming and deploying applications on the windows platform.

Compilation and Execution

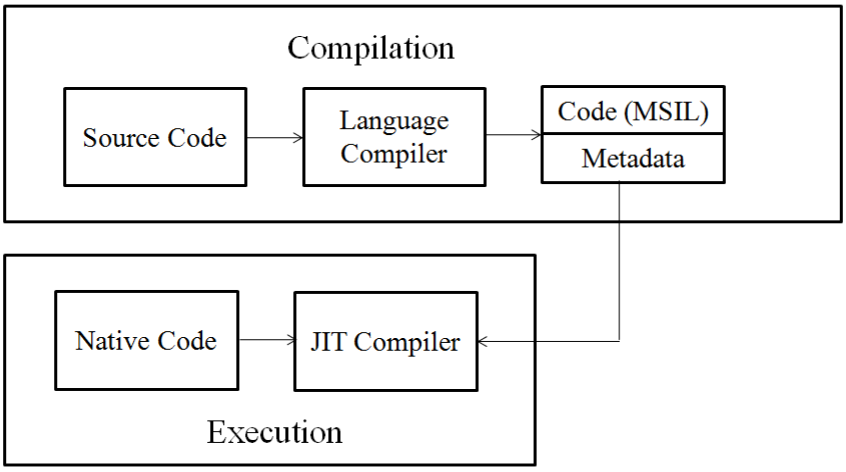

Figure 5: Compilation and Execution Process

Solutions and Projects

- In VB.Net project groups are known as solutions.
- By default, when you create a new project in VB.Net, then visual basic creates a new solution first and then adds a project to that solution.

File Extensions used in VB.Net

- When you save a solution, the file extension is ".sln" and all projects in the solution are saved with extension ".vbproj".
- Most popular file extension is ".vb"

Types of projects:

The following list provides a comparison of Visual Basic 6.0 and Visual Basic .NET project types.

Visual Basic 6.0	Visual Basic .NET
Standard EXE	Windows Application
ActiveX DLL	Class Library
ActiveX EXE	Class Library
ActiveX Control	Windows Control Library
ActiveX Document	No equivalent. Visual Basic .NET can interoperate with ActiveX Documents.
DHTML Application	No equivalent. Use ASP.NET Web Application.
IIS Application (Web Class)	No equivalent. Use ASP.NET Web Application.

The Visual Basic projects you can create are as follows:

- **Windows Application** Windows standard thick client applications based on forms (EXE)
- **Class Library** For individual classes or collections of classes (DLL)
- **Windows Control Library** Controls and components for Windows Forms (classic)

- **. ASP.NET Web Application** ASP.NET−based application composed of static or dynamic HTML pages
- **ASP.NET Web Service** For Web services to be used by clients communicating over the HTTP protocol
- **Web Control Library** Web−based controls for ASP.NET applications
- **Console Application** Your standard Console application
- **Windows Service** Create Windows services
- **Empty Project** Empty Windows application project
- **Empty Web Project** Empty Web server−based application

Visual Basic Integrated Development Environment:

Start Page

- User can use the start page to select from recent projects
- By default 'Get Started' item is selected in the start page.
- User can create new project or open existing project from recent project item.

Toolbars

- This feature is another handy aspect of the IDE.
- These appear near the top of the IDE.
- IDE displays tool tips, it becomes easy to know which button performs which operation.
- Toolbars provides a quick way to select menu item.

Graphical Designer

- VB.Net can display those elements which will look like at run time.
- Different types of graphical designers including.
    a) windows form designers
    b) web form designers
    c) compact designers
    d) XML designers
- From tools menu → select options → options dialog box will open from that select "Window Form Designer" folder display possible options.

The Object Explorer

- This tool lets you look at all the members of an object at once
- The Object Explorer helps open up any mysterious objects that Visual Basic has added to your code so you can see what's going on inside.
- To open the Object Explorer, select View →Other Windows →Object Explorer
- The Object Explorer shows all the objects in your program and gives you access to what's going on in all of them.

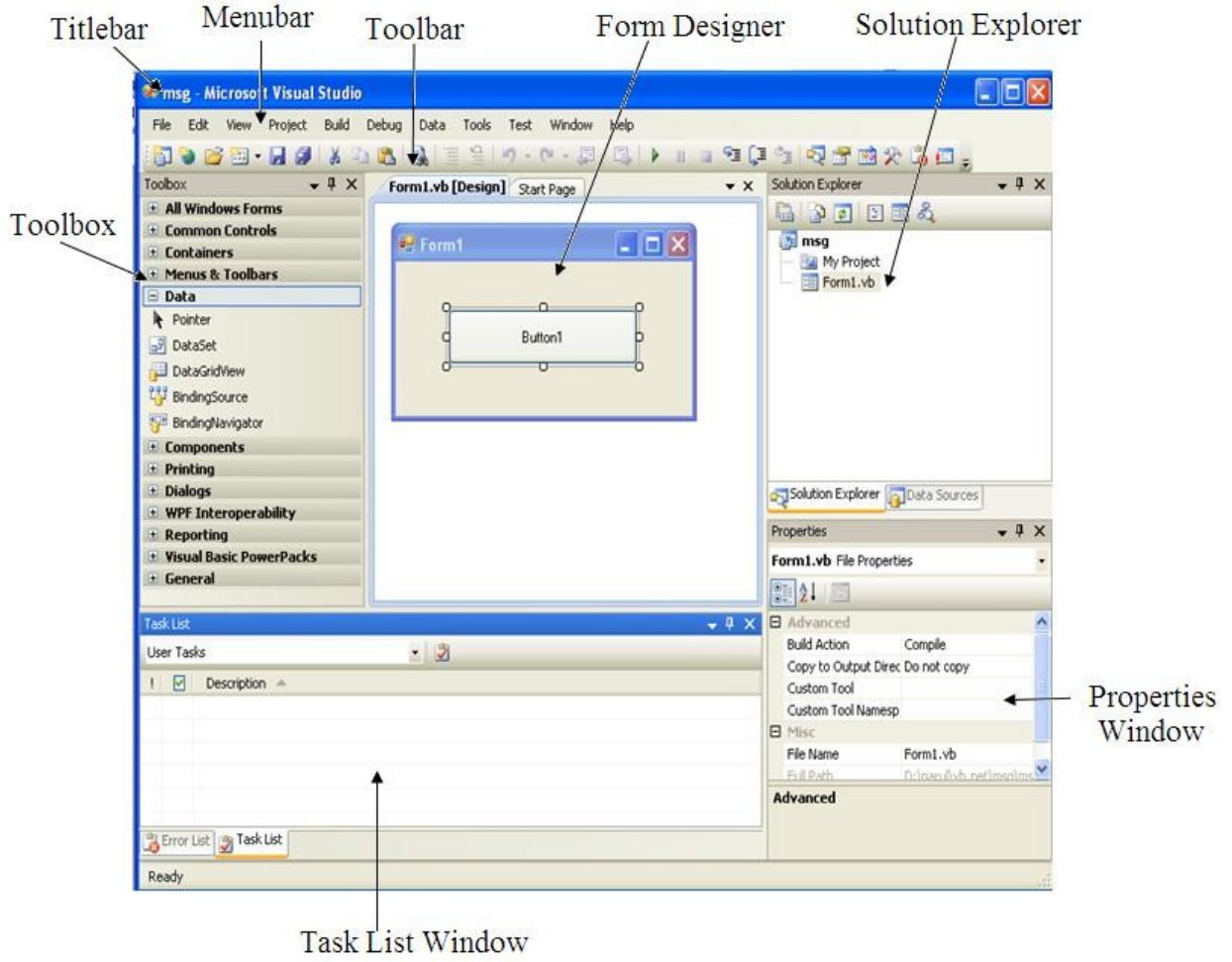

Figure 6: VB .NET IDE

The Menu

- Visual Studio .NET's menu is dynamic, meaning that items will be added or removed depending on what you are trying to do. The menu bar consist of the following options:

File: With this menu you can create a new project, open existing one, save the current project ,exit form the vb.net etc

Edit: The Edit menu provides access to the items you would expect: Undo, Redo, Cut, Copy, Paste, and Delete.

View: The View menu provides quick access to the windows that make up the IDE, such as the Solution Explorer, Properties window, Output window, Toolbox, etc.

Project: The Project menu allows you to add various extra files to your application.

Build: The Build menu becomes important when you have completed your application and want to be able to run it without the use of the Visual Basic .NET environment.

Debug: The Debug menu allows you to start and stop running your application within the Visual Basic .NET IDE. It also gives you access to the Visual Studio .NET **debugger**.

Data: The Data menu helps you use information that comes from a database. It only appears when you are working with the visual part of your application ,not when you are writing code.

Format: The Format menu also only appears when you are working with the visual part of your application. Items on the Format menu allow you to manipulate how the windows you create will appear to the users of your application.

Tools : The Tools menu has commands to configure the Visual Studio .NET IDE, as well as links to other external tools that may have been installed.

Window: The commands on this menu allow you to change the physical layout of the windows in the IDE.

Help: The Help menu provides access to the Visual Studio .NET documentation.

Code Designers

- You can use the tabs at the top center of the IDE to switch between graphical designer and code designer
- From view menu, you can also switch between by using code (F7) and Designer (Shift+F7) items.
- From solution Explorer, from the left side you can use top two buttons.
- At the top of code designer two drop down list boxes are available. The two drop-down list boxes at the top of the code designer; the one on the left lets you select what object's code you're working with, and the one on the right lets you select the part of the code that you want to work on, letting you select between the declarations area, functions, Sub procedures, and methods.

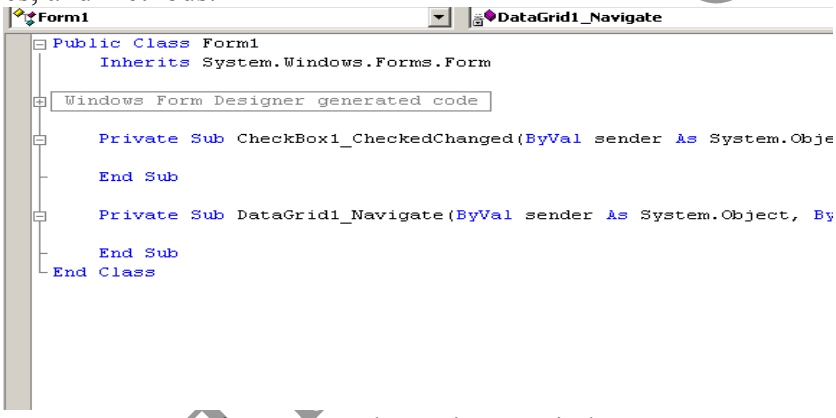

Figure 7: Code Designer Window

IntelliSense

- One useful feature of VB .NET code designers is Microsoft's *IntelliSense*. IntelliSense are those boxes that open as you write your code, listing all the possible options and even completing your typing for user.
- IntelliSense is made up of a number of options, including:
- *List Members*-Lists the members of an object.
- *Parameter Info*-Lists the arguments of procedure calls.
- *Quick Info*-Displays information in tool tips as the mouse rests on elements in your code.
- *Complete Word*-Completes typed words.
- *Automatic Brace Matching*-Adds parentheses or braces as needed.
- you can turn various parts of IntelliSense off if you want; just select the Tools → Options menu item, then select the Text Editor folder, then the Basic subfolder, and finally the General item in the Basic subfolder.

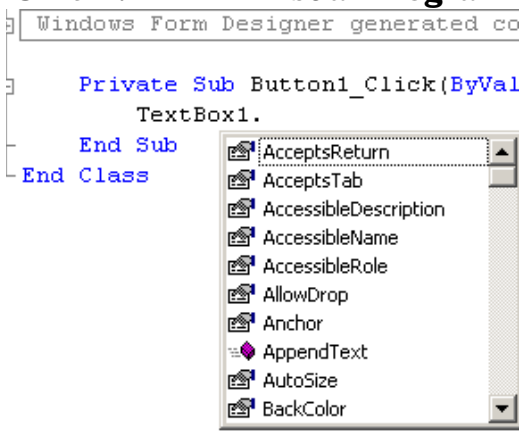

Figure 8 : Intellisense

The Toolbox

- It is available on left side of IDE.
- It uses tabs to divide its contents into categories like marked Data, Components, Windows Forms, and General.
- The Data, Components, Windows Forms, and General tabs appear when you're working with a Windows form in a Windows form designer, but when you switch to a code designer in the same project, all you'll see are General and Clipboard Ring in the toolbox. When you're working on a Web form, you'll see Data, Web Forms, Components, Components, HTML, Clipboard Ring, and General, and so on.
- The Data tab displays tools for creating datasets and making data connections.
- The Windows Forms tab displays tools for adding controls to Windows forms and so on.
- The General tab is empty by default, and is a place to store general components, controls, and fragments of code in.

The Solution Explorer

- It is available on right top corner of IDE
- This tool displays a hierarchy-with the solution at the top of the hierarchy, the projects one step down in the hierarchy, and the items in each project as the next step down.
- You can set the properties of various items in a project by selecting them in the Solution Explorer and then setting their properties in the properties window. And you can set properties of solutions and projects by right-clicking them and selecting the Properties item in the menu that appears, or you can select an item and click the properties button, which is the right-most button at the top of the Solutions Explorer.
- User can switch between graphical and code designers by using the buttons that appear at top left in the Solution Explorer
- You can right-click a solution and add a new project to it by selecting the Add→New Project menu item in the popup menu that appears. And you can specify which of multiple projects runs first-that is, is the startup project or projects-by right-clicking the project and selecting the Set As Startup Object item, or by right-clicking the solution and selecting the Set Startup Projects item.
- With this tool user can also add new item
- By clicking on see all button, solution explorer will shows all files available with current project.

- Refresh button is also available.

The Class View Window

- If you click the Class View tab under the Solution Explorer, you'll see the Class View window, This view presents solutions and projects in terms of the classes they contain, and the members of these classes.
- Using the Class View window gives you an easy way of jumping to a member of class that you want to access quickly-just find it in the Class View window, and double-click it to bring it up in a code designer.

The Properties Window

- The Properties window is divided into two columns of text, with the properties on the left, and their settings on the right.
- From drop-down list box at top of properties window, user can select any object which is available in current form.
- When you select a property, Visual Basic will give you an explanation of the property in the panel at the bottom of the Properties window. And you can display the properties alphabetically by clicking the second button from the left at the top of the Properties window, or in categories by clicking the left-most button.

The Dynamic Help Window

- The window that shares the Properties window's space, however, is quite new-the Dynamic Help window. Visual Basic .NET includes the usual Help menu with Contents, Index, and Search items, of course, but it also now supports dynamic help, which looks things up for you automatically. You can see the Dynamic Help window by clicking the Dynamic Help tab under the Properties window.
- VB .NET looks up all kinds of help topics on the element you've selected automatically; for example, I've selected a button on a Windows form, and dynamic help has responded by displaying all kinds of helpful links to information on buttons.

The Server Explorer

- It is used to explore what's going on in a server
- Using this tool you can drag and drop whole items onto windows forms from server explorer. Ex. Database.

The Output Window

- At the bottom of the IDE, two tabs are available one is Output and other is Breakpoints windows.
- From View menu → Other Window → Select Output Window.
- This window displays results of building and running programs.
- Using system Diagnostic.Debug.Write method user can send output to output window.
- Ex. System.Diagnostics.Debug.Write("Hello")

The Task List

- It is display from View → Show Tasks → All
- The Task List displays tasks that VB .NET assumes you still have to take care of, and when you click a task, the corresponding location in a code designer appears.

The Command Window

- It is display from View → Other Windows → Command Window
- It opens the Command window
- This window is a little like the Immediate window in VB6, because you can enter commands like **File.AddNewProject** here and VB .NET will display the Add New

Project dialog box. However, this window is not exactly like the Immediate window, because you can't enter Visual Basic code and have it executed.

Object Explorer Window

- The object explorer window allows us to view all the members of an object at once. It lists all the objects in our code and gives us access to them. The image below displays an object explorer window. You can view the object explorer window by selecting View->Other Windows-> Object Browser from the main menu.

Data Types:

- The following are the data type supported by VB.Net .
- Numeric Data Type(Short, Integer, Long, Single, Double, Decimal)
- Character Data Type (Char, String)
- Miscellaneous Data Type(Boolean , Byte, Date, Object)

Data Type	Sample value	Range of values
String	"hello there"	Alphanumeric characters, digits and other characters
Char	'a'C	Single characters the C stands for character
Integer	-5	Larger numbers ranging from -2147483648 to 2147483647
Long	18459038	Very Large whole numbers
Boolean	True	**True** or **False**
Byte	10	small numbers range 0 to 255
Short	1234	medium numbers ranging from -32,768 to 32,767
Single	5.678	Single-precision floating point numbers with six digits of accuracy.
Double	-2.4585E30	Double-precision floating point numbers with 14 digits of accuracy
Decimal	10.50	Decimal fractions, such as dollars and cents.
Date	#5/23/2002#	month, day, year, various date formats available
Object	frm as Form	Objects represent a group of many values

Following table shows storage size in memory for the data type.

Type	Storage Size
String	2 bytes
Char	2 bytes
Integer	4 bytes
Long	8 bytes
Boolean	2 bytes
Byte	1 byte
Short	2 bytes
Single	4 bytes
Double	8 bytes
Decimal	16 bytes
Date	8 bytes
Object	4 bytes

Variables:

- A variable is something that is used in a program to store data in memory.
- A variable has a name and a data type which determine the kind of data the variable can store.

Variable declaration: Dim statement is used to declare a variable.
Syntax: Dim variablename [([subscript])] [As [New] datatype

Variablename : It is required. It specifies the name of variable which user wants to create.

Subscript : It is optional. Subscript is used to specify the size of array when user declares an array.

New: New keyboard enables creation of new object. If you use new when declaring the object variable, a new instance of the object is created.

Type : The type specify the data type of the variables.

Ex: Dim a as integer, s1 as string

Variable name should follow the following rules:

- Begin with a letter or _.
- Must contain atleast one numeric digit or alphabetic character.
- Maximun 1023 characters are allowed.
- It must be unique in its scope.

- In VB.NET each variable contains default value depends on its data type.
- Default value for Numeric and Byte data type is 0(zero).
- Default value for Char data type is Binary 0(zero).
- Default value for all reference types like object, string, and arrays is Nothing.
- Default value for Boolean data type is False.
- Default value for Date data type is 12:00 AM of 1,1,0001.

Constant Declaration:

Declares and defines one or more constants.

Syntax: Const constantlist

Each *constant* has the following syntax and parts:

constantname[As *datatype*] = *initializer*

***constantlist* :** Required. List of constants being declared in this statement. *Constant* [, *constant* ...]

Constantname: It is required. It specifies the name of the constant.

Datatype: It specifies the type of constant.

Initialize: it is required. The value which is assigned to the constant. Once you initialize the constant variable with a value it can never change.

Ex: Const pi as double = 3.24

Operators

- Visual Basic comes with many built-in operators that allow us to manipulate data. An operator performs a function on one or more operands. For example, we add two variables with the "+" addition operator and store the result in a third variable with the "=" assignment operator like this: int x + int y = int z. The two variables (x ,y) are called operands. There are different types of operators in Visual Basic and they are described below in the order of their precedence.
- Operators may me Unary or Binary
- Unary used with a single operand for example Ans= -10
- Binary used with two operands for example Ans= 10 / 5

- Operators also categorized in following categories:

Arithmetic Operators : Arithmetic operators are used to perform arithmetic operations that involve calculation of numeric values. The table below summarizes them:

Operator	Use
^	Exponentiation
-	Negation (used to reverse the sign of the given value, exp -intValue)
*	Multiplication
/	Division for ex a=11/5 then answer is 5.5
\	Integer Division for ex a=11\5 then answer is 5
Mod	Modulus Arithmetic
+	Addition
-	Subtraction

Concatenation Operators : Concatenation operators join multiple strings into a single string. There are two concatenation operators, + and & as summarized below:

Operator	Use
+	**String Concatenation**
&	**String Concatenation**

Comparison Operators : A comparison operator compares operands and returns a logical value based on whether the comparison is true or not. The table below summarizes them:

Operator	Use
=	Equality
<>	Inequality
<	Less than
>	Greater than
>=	Greater than or equal to
<=	Less than or equal to

Logical / Bitwise Operators : The logical operators compare Boolean expressions and return a Boolean result. In short, logical operators are expressions which return a true or false result over a conditional expression. The table below summarizes them:

Operator	Use
Not	Negation

And	Conjunction
AndAlso	Conjunction
Or	Disjunction
OrElse	Disjunction
Xor	Disjunction

Type Conversion functions

- Type conversion is used for convert one data type to another
- There are two type of conversion : **Implicit and Explicit.**
- An **Implicit Conversion** does not require any special syntax in the source code.
- An **Explicit Conversion** requires function.
- For Example : Implicit Conversion

```
Dim a As Integer
Dim b As Double
a = 4499
b = a
```

- An explicit conversion requires function.

Function Name	Convert 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

- Following is common syntax for each type conversion function:

Syntax : Function_Name(argument)

For Example:

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

CTYPE function

- It uses to convert one type to another type.
- Instead of remember all conversion functions , we can use CTYPE function
- Execution is faster .

Syntax : Ctype(expression,Type name)

For Example :

```
Dim no1 As Integer
Dim no2 As Double
no2 = 66.77
no1 = Ctype(no2,Integer)
```

Boxing and Unboxing:

- Boxing and unboxing act like bridges between value type and reference types. When we convert
  value type to a reference type it's termed as boxing. Unboxing is just vice-versa.
- 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

Boxing conversions.

A boxing conversion permits any value-type to be implicitly converted to the type object or to any interface-type implemented by the value-type.
Boxing a value of a value-type consists of allocating an object instance and copying the value-type value into that instance.
For example any value-type G, the boxing class would be declared as follows:

```
Class vBox
Private value As G
Sub New(ByVal g As G)
value = g
End Sub 'New
End Class
```

BoxBoxing of a value v of type G now consists of executing the expression new G_Box(v), and returning the resulting instance as a value of type object.
Thus, the statements

```
Dim i As Integer = 12
Dim box As Object = i
```

conceptually correspond to

```
Dim i As Integer = 12
Dim box = New int_Box(i)
```

Boxing classes like G_Box and int_Box above don't actually exist and the dynamic type of a boxed value isn't actually a class type. Instead, a boxed value of type G has the dynamic type G, and a dynamic type check using these operator can simply reference type G. For example,

```
Dim i As Integer = 12
Dim box As Object = i
If TypeOf box Is Integer Then
    Console.Write("Box contains an int")
End If
```

will output the string "Box contains an integer" on the console.

Unboxing conversions.

An unboxing conversion permits an explicit conversion from type object to any value-type or from any interface-type to any value-type that implements the interface-type.
An unboxing operation consists of first checking that the object instance is a boxed value of the given value-type, and then copying the value out of the instance.
Unboxing conversion of an object box to a value-type G consists of executing the expression ((G_Box)box).value.
Thus, the statements

```
Dim box As Object = 12
Dim i As Integer = CInt(box)
```

conceptually correspond to

```
Dim box = New int_Box(12)
Dim i As Integer = CType(box, int_Box).value.
```

For an unboxing conversion to a given value-type to succeed at run-time, the value of the source argument must be a reference to an object that was previously created by boxing a value of that value-type. If the source argument is null or a reference to an incompatible object, an InvalidCastException is thrown.

Array:

- An ordered collection of same type of data having single variable name is known as array. Each element of the array can be referenced by a numerical subscript.
- In VB.Net two types of arrays are:
  a) Standard Array
  b) Dynamic Array

Declaration Of Standard Array:
Syntax : Dim varname [(subscripts)] [As type]

WithEvents: This keyword is valid only in class modules. This keyword specifies that varname is an object variable used to respond to events triggered by an ActiveX object.
VarName : The Varname specify the name of variable which you want to create.
Subscript : Subscript is used when you declare an array.
Type : The type specify the data type of the array variables. User can also include "To" keyword in array declaration.
Ex : Dim n(5) As Integer.
Dim s(3) As String.
Dim n(1 to 4) As Integer.

Dynamic Array:

- In any C or C++ programming language user can not modify the size of the array. Once we declared the size of array then it becomes fixed.
- In VB .NET we can increase the size of array.
- In some cases we may not know exactly the size of array at declaration time. We may need to change the size of the array at runtime. So we can resize the array at any time by using Redim statement.
- But with dynamic array we cannot change the dimension of the array.

Redim Statement: It reallocates storage space for array variables.
Syntax: ReDim [**Preserve**] *name***(**boundlist**) [,** *name***(**boundlist**) …]**
Preserve: It is optional. It is used to preserve the data in an existing array when user changes the size of the last dimension.

Name: The name of the array variable.
Boundlist: It is required. It is dimensions of an array variable.
Example:

```
Dim a() As Integer
Private Sub cmdInput_Click()
   ReDim a(5) As Integer
   For i = LBound(a) To UBound(a)
      a(i) = InputBox("Enter elements:")
   Next
```

```
End Sub
Private Sub cmdPrint_Click()
   ReDim Preserve a(3) As Integer
   'MsgBox LBound(a) & UBound(a)
    For i = LBound(a) To UBound(a)
      msgbox a(i)
   Next
End Sub
```

- **String Functions:**

1) Len: This function returns an integer containing the number of characters in a given string.

Syntax : Len(string)

String: Any valid string expression. If *string* contains Null, Null is returned.

Example:

```
S1=”hello”
Msgbox len(s1)
```

2) Mid: It returns a **Variant** (**String**) containing a specified number of characters from a string.

Syntax: **Mid**(***string***, ***start***[, ***length***])

The **Mid** function syntax has these named arguments:

Part	Description
String	Required. String expression from which characters are returned. If ***string*** contains Null, **Null** is returned.
Start	Required; Long. Character position in ***string*** at which the part to be taken begins. If ***start*** is greater than the number of characters in ***string***, **Mid** returns a zero-length string ("").
Length	Optional; **Variant** (**Long**). Number of characters to return. If omitted or if there are fewer than ***length*** characters in the text (including the character at ***start***), all characters from the ***start*** position to the end of the string are returned.

Examples:

```
Dim MyString, FirstWord, LastWord
MyString = "Mid Function Demo"   ' Create text string.
FirstWord = Mid(MyString, 1, 3)   ' Returns "Mid".
LastWord = Mid(MyString, 14, 4)   ' Returns "Demo".
```

3) Trim , Rtrim, Ltrim: It Returns a string that contains a copy of a specified string without leading spaces (**LTrim**), without trailing spaces (**RTrim**), or without leading or trailing spaces (**Trim**).

Syntax: **Trim / Rtrim / Ltrim (string)**

String: It is requires any valid **String** expression. If *string* equals **Nothing**, the function returns an empty string.

Example:

```
S1= “   This   is test   “
Msgbox Trim(s1)
Msgbox Rtrim(s1)
Msgbox  Ltrim(s1)
```

4) Instr: It returns a **Variant** (**Long**) specifying the position of the first occurrence of one string within another.

Syntax: **InStr**([*start*,]*string1*, *string2*[, *compare*])

The **InStr** function syntax has these arguments:

Part	Description
Start	Optional. Numeric expression that sets the starting position for each search. If omitted, search begins at the first character position. If ***start*** contains Null, an error occurs. The ***start*** argument is required if ***compare*** is specified.
string1	Required. String expression being searched.
string2	Required. String expression sought.
Compare	Optional. Specifies the type of string comparison. If ***compare*** is Null, an error occurs. If ***compare*** is omitted, the **Option Compare** setting determines the type of comparison.

Settings: The *Compare* argument settings are:

Constant	Value	Description
Binary	0	Performs a binary comparison
Text	1	Performs a text comparison

Return Value

If	InStr returns
String1 is zero length or **Nothing**	0
String2 is zero length or **Nothing**	*start*
String2 is not found	0
String2 is found within *String1*	Position where match begins

Examples:

```
Dim SearchString, SearchChar, MyPos
SearchString ="XXpXXpXXPXXP"   ' String to search in.
SearchChar = "P"   ' Search for "P".
' A textual comparison starting at position 4. Returns 6.
MyPos = Instr(4, SearchString, SearchChar, 1)
' A binary comparison starting at position 1. Returns 9.
```

MyPos = **Instr(**1**,** SearchString**,** SearchChar**,** 0**)**
' Comparison is binary by default (last argument is omitted).
MyPos = **Instr(**SearchString**,** SearchChar**)** ' Returns 9.

5) Lcase: It returns a String that has been converted to lowercase.

Syntax: **LCase**(*string*)

The required *string* argument is any valid string expression. If *string* contains Null, Null is returned. Only uppercase letters are converted to lowercase; all lowercase letters and nonletter characters remain unchanged.
Example:
Dim UpperCase, LowerCase
Uppercase = "Hello World 1234" ' String to convert.
Msgbox **Lcase(**UpperCase**)** ' Returns "hello world 1234"

6) Ucase: It returns a **Variant** (**String**) containing the specified string, converted to uppercase.

Syntax: **UCase**(*string*)

The required *string* argument is any valid string expression. If *string* contains Null, **Null** is returned.
Only lowercase letters are converted to uppercase; all uppercase letters and nonletter characters remain unchanged.
Example:
Dim LowerCase, UpperCase
LowerCase = "Hello World 1234" ' String to convert.
Msgbox **UCase(**LowerCase**)** ' Returns "HELLO WORLD 1234".

7) Asc: It returns an Integer representing the character code corresponding to the first letter in a string.

Syntax: **Asc(***string***)**

The required *string* argument is any valid string expression. If *String* is a **String** expression, only the first character of the string is used for input. If *String* is **Nothing** or contains no characters, an error occurs.
Example: msgbox **Asc("**A"**)**

8) Chr: It returns a String containing the character associated with the specified character code.

Syntax: **Chr(***charcode***)**
An **Integer** expression representing the *code point*, or character code, for the character. If *CharCode* is outside the valid range, an error occur. The valid range for **Chr** is 0 through 255.
Examples:
Dim MyChar
MyChar = **Chr(**65**)** ' Returns A.

9) Space: It returns a string consisting of the specified number of spaces.

Syntax: **Space**(number)

Number : It is required Integer expression. The number of spaces you want in the string.
Example: Msgbox " Hi" & space(5) & " How r u?"

10) Format: This function returns a string formatted according to instructions contained in a format String expression.

Syntax: **Format** (Expression, style)

Expression : it is any valid expression.
Style: it is applied on specified expression
Example:
D1= #02/14/1989#
Msgbox format(d1,"DD-MM-YY")

11) Strcomp: It returns -1, 0, or 1, based on the result of a string comparison.
Syntax: **Strcomp**(string1,string2[,compare])
String1 :Required. Any valid **String** expression.
String2 :Required. Any valid **String** expression.
Compare :Optional. Specifies the type of string comparison. If *Compare* is omitted, the **Option Compare** setting determines the type of comparison.

The *Compare* argument settings are:

Return Value:The StrComp function has the following return values.

If	StrComp returns
String1 sorts ahead of *String2*	-1
String1 is equal to *String2*	0
String1 sorts after *String2*	1

Example:

```
Dim TestStr1 As String = "ABCD"
Dim TestStr2 As String = "abcd"
Dim TestComp As Integer
' The two strings sort equally. Returns 0.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Text)
' TestStr1 sorts after TestStr2. Returns -1.
TestComp = StrComp(TestStr1, TestStr2, CompareMethod.Binary)
' TestStr2 sorts before TestStr1. Returns 1.
TestComp = StrComp(TestStr2, TestStr1)
```

12) Left: It returns a **Variant** (**String**) containing a specified number of characters from the left side of a string.

Syntax: **Left**(***str***, ***length***)

str :it is required **String** expression from which the rightmost characters are returned.
Length*:* It is required Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *str*, the entire string is returned.

Examples:

```
Dim AnyString, MyStr
AnyString = "Hello World"   ' Define string.
MyStr = Microsoft.VisualBasic.Left(AnyString, 1)
Msgbox Mystr      ' Returns "H".
```

13) **Right:** It returns a string containing a specified number of characters from the right side of a string.

Syntax: **Right**(str,length)

str :it is required **String** expression from which the rightmost characters are returned.
Length: It is required Numeric expression indicating how many characters to return. If 0, a zero-length string ("") is returned. If greater than or equal to the number of characters in *str*, the entire string is returned.

Example:

```
S1=”this is test”
Msgbox Micosoft.VisualBasic.Right(s1,3)
```

14) **Replace:** It returns a string in which a specified substring has been replaced with another substring a specified number of times.

Syntax: **Replace (Expression, Find, Replacement)**

Expression :Required. String expression containing substring to replace.
Find :Required. Substring being searched for.
Replacement :Required. Replacement substring.

Example: S1= “Shopping List”

Msgbox Replace(s1, “o”,”I”)

Tostring with its Methods:

1) **ToString.Concat:** This method is used Concatenates three specified instances of String.

Syntax : **System.String.Concat(str1,str2)**

str1 : Parameter String

str2 : Parameter String

Example:

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
```

```
        ByVal e As System.EventArgs) Handles Button1.Click
    Dim str1 As String
    Dim str2 As String

    str1 = "Concat() "
    str2 = "Test"
    MsgBox(String.Concat(str1, str2))
  End Sub
End Class
```

2) **String.copy :** This method creates a new instance of String with the same value as a specified String.

Syntac: System.String.Copy(str)

str : The argument String for Copy method

Example:

```
Public Class Form1
  Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    Dim str1 As String
    Dim str2 As String

    str1 = "VB.NET Copy() test"
    str2 = String.Copy(str1)
    MsgBox(str2)
  End Sub
End Class
```

3) **String.Indexof:** It returns the index of the first occurrence of the specified substring.

Syntax: System.String.IndexOf(str)

str - The parameter string to check its occurrences

If the parameter String occurred as a substring in the specified String then it returns position of the first character of the substring. If it does not occur as a substring, -1 is returned.

Example:

```
Public Class Form1
  Private Sub Button1_Click(ByVal sender As System.Object, _
  ByVal e As System.EventArgs) Handles Button1.Click
    Dim str As String
    str = "VB.NET TOP 10 BOOKS"
    MsgBox(str.IndexOf("BOOKS"))
  End Sub
End Class
```

4) **String.substring:** It returns a new string that is a substring of this string. The substring begins at the specified given index and extended up to the given length.

Syntax: Substring(startIndex,length)

startIndex: The index of the start of the substring.

length: The number of characters in the substring.

Example:

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click

      Dim str As String
      Dim retString As String
      str = "This is substring test"
      retString = str.Substring(8, 9)
      MsgBox(retString)

   End Sub
End Class
```

5) **String.format:** VB.NET **String Format** method replace the argument Object into a text equivalent System.Striing.

Syntax: System.Format(format, arg0)

String format : The format String

The format String Syntax is like {indexNumber:formatCharacter}

Object arg0 : The object to be formatted.

Examples:

Currency : ***String.Format("{0:c}", 10)*** will return $10.00

The currency symbol ($) displayed depends on the global locale settings.

Date : ***String.Format("Today's date is {0:D}", DateTime.Now)***

You will get Today's date like : 01 January 2005

Time : ***String.Format("The current time is {0:T}", DateTime.Now)***

You will get Current Time Like : 10:10:12

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, _
   ByVal e As System.EventArgs) Handles Button1.Click
      Dim dNum As Double
```

```
    dNum = 32.123456789
    MsgBox("Formated String " & String.Format("{0:n4}", dNum))
  End Sub
End Class
```

6) **String.ToUpper:** This method uses the casing rules of the current culture to convert each character in the current instance to its uppercase equivalent. If a character does not have an uppercase equivalent, it is included unchanged in the returned string.

The ToUpper method is often used to convert a string to uppercase so that it can be used in a case-insensitive comparison.

Syntax: **string.ToUpper()**

Example:
S1= “This is Test”
Msgbox s1.Toupper()

7) **String.ToLower**: This method does not modify the value of the current instance. Instead, it returns a new string in which all characters in the current instance are converted to lowercase.

Syntax: **string.tolower()**

Example:
S1= “This is Test”
Msgbox s1.tolower()

8) **String.Remove:** Deletes all the characters from this string beginning at a specified position and continuing through the last position.

Syntax: **String.Remove(startIndex)**

startIndex : The position to begin deleting characters.

Example:

S1=”abc---def”
Msgbox s1.remove(3)