

SUBJECT: C# & .NET FRAMEWORK
RGM-R15 Regulations
III B Tech IISEM
DEPARTMENT OF CSE
RGMCET,NANDYAL

PREPARED
BY
Mr . G . RAJASEKHAR REDDY

**(A0510155) C# & .NET FRAMEWORK
(common to CSE & IT)**

AIM:

- To cover the fundamental concepts of the C# language and the .NET framework.

OBJECTIVE:

- The student will gain knowledge in the concepts of the .NET framework as a whole and the technologies that constitute the Framework.
- The student will gain programming skills in C# both in basic and advanced levels.
- By building sample applications, the student will get experience and be ready for large-scale projects.

OUTCOMES:

- To get knowledge of windows programming.
- To get knowledge on server side programming.
- To gain knowledge on web services and dynamic link libraries.

UNIT I

INTRODUCTION to C#: Introducing C#, Understanding .NET, Overview of C#, Literals, Variables, Data Types, Operators, Expressions, Branching, Looping, Methods, Arrays, Strings, Structures, Enumerations.

UNIT II

OBJECT ORIENTED ASPECTS OF C#: Classes, Objects, Inheritance, Polymorphism, Interfaces, Operator Overloading, Delegates, Errors and Exceptions.

UNIT III

WINDOWS APPLICATIONS: Drawbacks of Console Applications, Container Controls, Non Container Controls, Developing Windows Application from Notepad and Visual Studio, Events, Types of Events – Mouse, Focus, Drag, Key and Other Related Events, Building Windows Applications.

ADO.NET: Problems with File Handling, Data Source Communication, Drivers and Providers, Introduction of ADO.NET, ADO.NET Namespaces, ADO.NET Objects, Accessing Data with ADO.NET.

UNIT IV

BUILDING ASP.NET WEB PAGES: HTML form Development, Client side Scripting, Server side Scripting, Web applications and Web servers, HTTP, Advantages Using ASP.NET, ASP.NET Application, ASP.NET Namespaces, ASP.NET Web Page Structure, Creating Sample C# Web Applications, ASP.NET Web Page Execution Architecture, Debugging and Tracing of ASP.NET.

UNIT V

ASP.NET WEB CONTROLS: Web Form Structures, Introduction to Web Form controls, Server Side Controls, Web Server Controls, GET and POST, Page Submission, Web Page Creation Techniques, Redirection between Web Pages, Validation Controls.

UNIT VI

WEB SERVICES: Web Services, Web Service Architecture, WSDL, Building WSDL Web Service.

CONFIGURING .NET ASSEMBLIES: Private Assemblies, Shared Assemblies, and Versioning.

MULTITHREADED PROGRAMMING: Thread Class, Life Cycle of a Thread, Steps for Creating a Thread, Thread Synchronization.

UNIT I

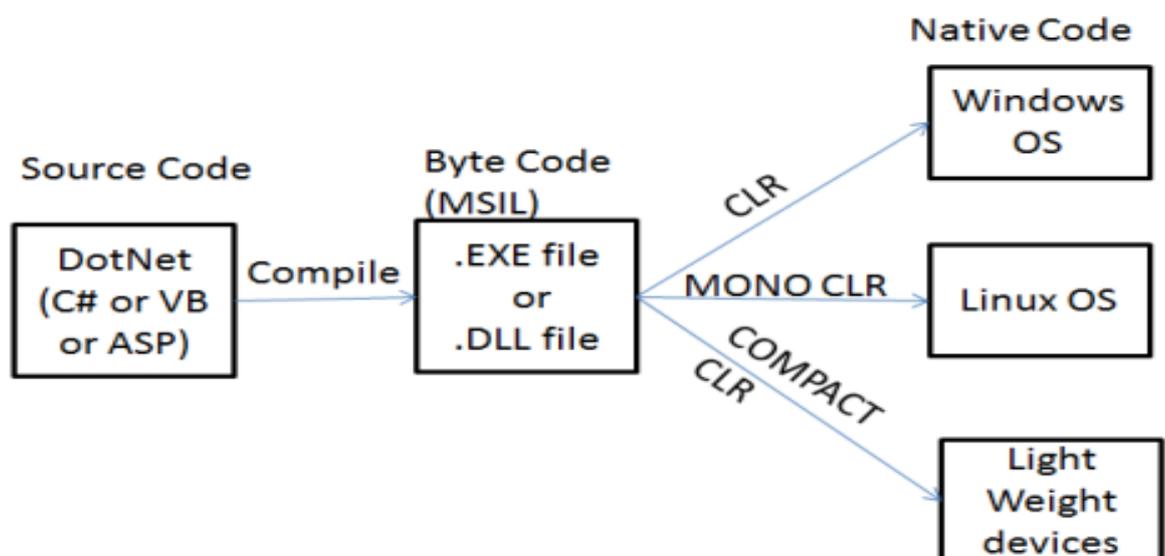
.NET:

- .NET stands for Network Enable Technology which is developed by Microsoft Corporation.
- .NET is a collection of languages like VC#.NET, VC++.NET, VB.NET and so on.
- .NET is a Framework which provides a common platform to Execute or Run the applications developed in various programming languages.

.NET Objectives: The .NET Framework is designed to fulfil the following objectives

- **Platform independent:** As .exe and .dll files work in any operating system with the help of CLR, hence .Net is called as platform independent.
- (.exe) is executable file, it consists of executable code, and (.dll) is dynamic link library file it consist of reusable code. .exe and .dll files contains the code in the format of byte code is also called as **MSIL (Microsoft intermediate language) code**.
- Machine language is also called as native code. CLR is common language runtime; CLR software converts byte code into native code. Dot Net is **platform independent** but CLR software is **platform dependent**.

One question arises if we go in detail, that is either Dot Net is pure platform independent or not? Answer is Dot Net is partially platform dependent, as of now CLR software's are not available for DOS operating system and Windows 95.



- **Language independent:** As Dot Net programming logic can be developed in any Dot Net framework compatible languages; hence Dot Net is called as language independent.

Microsoft is introducing approximately 40 languages into Dot Net framework, out of which as of now approximately 24 languages and one specification are released. Ex: VC#.Net,VB.Net,VC++,VJ#,VF#,PHP,COBOL,PERL,PHYTHON,SMALLTALK,JSCRI PT...etc One specification is ASP.Net.

- **Language interoperability:** It is a concept of developing an application with the help of more than one .NET programming language. After an application is compiled, the source code will be converted into byte code. Byte code follows a standard instruction set provided by cls(common language specification).Cls provides common rules and syntaxes for all the languages and cls also provides common data types for all the languages and these common data types are called as CTS(common type system).

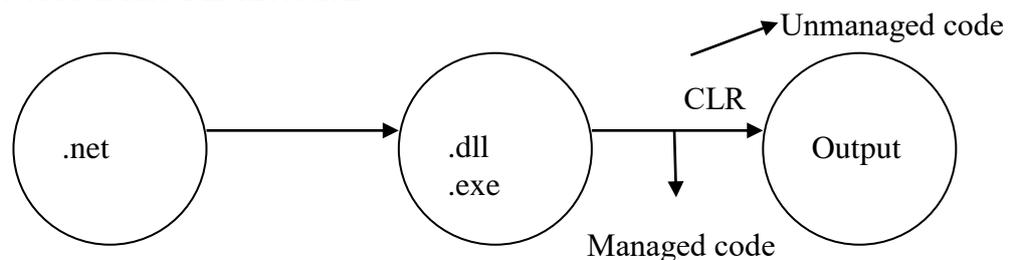
C++	C#	VB .NET
int(2)	short(2)	short(2)
long(4)	int(4)	integer(2)
	long(8)	long(8)
	SYSTEM.INT16	
	SYSTEM.INT32	
	SYSTEM.INT64	

- It supports object-oriented programming environment.
- It supports to work with database programming with the help of ADO.NET.
- It supports to work with WPF (windows presentation foundation) for developing animations.
- It Provides environment for developing various types of applications, such as Windows-based applications and Web-based applications
- It Supports to develop 3-tier architecture with the help of Dot Net remoting
- Supports to develop game programming with the help of multi-threading
- Supports to work with link programming

Managed Code: The code is developed and running under the control of the CLR is often termed *managed code*

Unmanaged code: The code which takes OS help while execution is called as unmanaged code

Note: Managed code is faster in execution.



.NET Framework:

- The .Net framework is a revolutionary platform that helps you to write the following types of applications:
Windows applications

Web applications

Web services

- The .NET framework applications are multi-platform applications.
- The .NET framework has been designed in such a way that it can be used by .NET framework compatible languages (C#, C++, Visual Basic, Jscript, COBOL, etc). All .NET framework compatible languages can access the framework as well as communicate with each other.
- The .NET frame work contain three major things that is
 - **Languages**—VB,C#,PASCAL,COBOL
 - **Technology**—ASP.NET,ADO.NET
 - **Servers**—SQL SERVER,SHARE POINT SERVER
- In platform independent languages, the code gets executed under the control of special software called **framework**.

Example: java-->JVM

.NET-->CLR

- **Framework:** A framework is a software which masks the functionalities of a OS and makes the code to execute under its control .Frame work provides the following basic features like
 - Platform independency
 - Security
 - Memory management
- Microsoft has started the development of .NET framework in late 90's originally under the name of NGWS (next generation windows services).
- To develop the framework first the Microsoft has prepared a set of specifications known as CLI (common language infrastructure).
- The .NET specifications are open to all (everyone can develop) which is standardized under ISO (International Standards organization) AND ECMA(European computer manufacture association).
- CLI specification talks about four important aspects that is
 - CLS (common language specification)
 - CTS (common Type system)
 - BCL (Base class libraries)
 - VES (Virtual execution system) or CLR (common language runtime)
- .NET framework versions

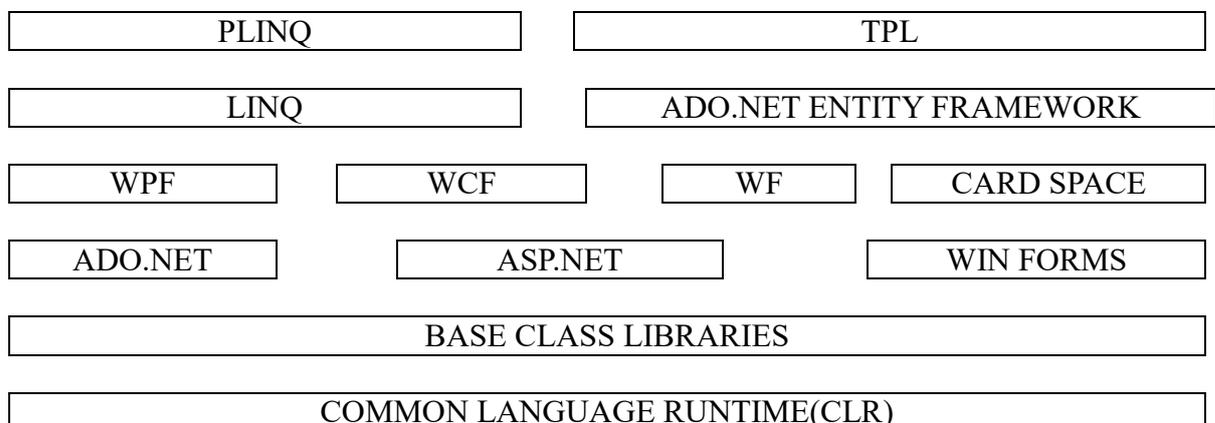
Versions

Release Date

1.0	2002
1.1	2003
2.0	2005
3.0	2006
3.5	2007
4.0	2009

- The .Net framework consists of an enormous library of codes used by the client languages such as C#. Following are some of the components of the .Net framework:
 - Common Language Runtime (CLR)
 - The .Net Framework Class Library
 - Common Language Specification
 - Common Type System
 - Metadata and Assemblies
 - Windows Forms
 - ASP.Net and ASP.Net AJAX
 - ADO.Net
 - Windows Workflow Foundation (WF)
 - Windows Presentation Foundation
 - Windows Communication Foundation (WCF)
 - LINQ

.NET framework architectural diagram: The .NET framework is one of the tool provided by the .NET infrastructure and tools component of the .NET platform. The .NET framework provides an environment for building, deploying and running web services and .NET applications.



Common language runtime (CLR):

- CLR is the heart of the .NET framework. It is the execution engine of .NET framework, where all .net applications are running under the supervision of CLR. The main function of CLR is to convert the managed code into native code and then execute the program. CLR acts

as a layer between OS and the applications written in .NET languages. CLR provides various features to applications like

- Security
 - Platform independency
 - Automatic memory management
 - Runtime error handling
 - Code verification
 - Compilation
 - Thread management
- The components of CLR are

Class Loader: It is used to load all the classes at runtime for the execution of an application.

JIT Compiler: It is responsible for the conversion of MSIL (byte code) into machine code (native code).

Code Manager: Is responsible for managing code at runtime.

Garbage Collector: The .NET garbage collector is responsible for automatic memory management, where memory management is a process of allocation and de-allocation of memory that is required for a program execution. Memory management is of two types

1. Manual/explicit memory management: In this the programmers are responsible for allocation and deallocation of memory that is required for a program execution.

2. Automatic/implicit memory management: In this case the garbage collector will take care of allocation and deallocation process of memory.

The garbage collector will allocate the memory for an object when and where they are required and also deallocates the memory of those objects once they become unused under the program. The unused object of a program is called garbage and deallocate immediately.

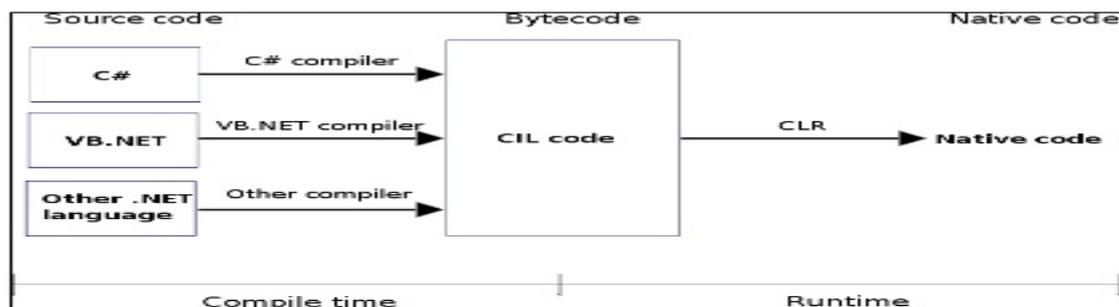
Security Engine (manager): It is responsible for taking care of the security of the application that is security engine will not allow the application to interact directly with the operating system (or) OS to interact with the application.

Type Checker: It enforces strictness in type checking that is type checker will verify the types used in the application with CTS (or) CLS standards supported by the CLR.

Common type system: Common Type System (CTS) describes a set of types that can be used in different .Net languages in common. That is, the Common Type System (CTS) ensure that objects written in different .Net languages can interact with each other.

Microsoft Intermediate Language (MSIL): MSIL stands for Microsoft Intermediate Language. We can call it as Intermediate Language (IL) or Common Intermediate Language (CIL). During the compile time, the compiler convert the source code into Microsoft Intermediate Language (MSIL) .Microsoft Intermediate Language (MSIL) is a CPU-independent set of instructions that can be efficiently converted to the native code. During the runtime the Common Language Runtime (CLR)'s Just In Time (JIT) compiler converts the Microsoft Intermediate Language (MSIL) code into native code to the Operating System.

Code Conversion



Thread Support: It allows multithreading support to our application.

Debug engine: It allows proper debugging of an application.

Base class library: It provides all the types that an application need at runtime.

Exception manager: Handles all the exception for an application during runtime.

COM Marshaller: It provides interoperability to our applications.

➤ **Base class libraries:**

- A library is a set of reusable functionalities where each and every programming language has built in library like header files in c/c++ ,packages in java similarly .NET languages have built in libraries known as base class libraries .the speciality of these library is that they can be consumed under any .NET language(BCL are the best example of language interoperability because those library are developed in C# language and can be consumed in any .NET framework compatible languages.
- The very popular namespace in base class library is **system**
- We can use the base class library in system namespace for many different tasks including.
 - Input/output operations
 - String handling
 - Managing arrays, lists and maps
 - Accessing files and file systems
 - Security

- Windowing
 - Windows messages
 - Drawing
 - Database management
 - Managing errors and exceptions
- **Win form technology** is used to develop windows based applications.
 - **Asp.net** is used to build rich internet based web applications
 - **Ado.net** is used to create Data Access Layer to query and manipulate data from underlying data source like SQL Server, Oracle etc.
 - **WPF (windows presentation foundation)** is used to create applications with a rich user experience. It includes application User Interface, 2D graphics, 3D graphics and multimedia. It takes advantage of hardware acceleration of modern graphic cards. WPF makes the User Interface faster, scalable and resolution independent.
 - **WCF (windows communication foundation)** is used for building and developing services based on Web Services standards.
 - **WF (WWF) stands windows workflow foundation**, which is used to build process oriented business workflow and rules engine.
 - **LINQ (language integrated query)** allows you to query(communicate) the data from the various data sources (like SQL databases, XML documents, Ado.Net Datasets, Various Web services and any other objects such as Collections, Generics etc.) using a SQL Query like syntax with .Net framework languages like C# and VB.
 - **ADO.NET Entity Framework** provides added features under the traditional ADO.NET. This is used to query and store data into to the relational databases (like SQL Server, Oracle, DB2 etc.) in ORM (Object-relational mapping) fashion.
 - **TPL (Task Parallel Library):** The purpose of TPL is to make the developers more productive by simplifying the process of adding parallelism and concurrency to applications.
 - **PLINQ (Parallel Language Integrated Query):** It is used to maximize processor utilization with increased throughput in a multicore architecture.

The Origins of .NET Technology: The Origins of the .NET Technology are

- **OLE Technology (Object Linking and Embedding):** Object linking and embedding (OLE) was developed by Microsoft in the early 1990's to enable inter process communication. OLE provides support to achieve the following.
 - To embed documents from one application into another application.

- To enable one application to manipulate objects located in other application.

OLE Technology enabled uses to develop applications which required interoperability between various modules such as MS-Word and MS-Excel

- **COM Technology (Component Object Model):** Till the advent of com technology, the monolithic approach had been used for developing software. But when the programs became too large and complex, the monolithic approach leads to a number of problems in terms of maintainability and testing of software. To overcome these problems Microsoft introduced a component based model for developing software. In this approach a program is divided into number of independent components where each one offers a particular task. Each Component can be developed and tested independently and then integrated into the main system. This technology is called component object model (COM).

Benefits:

1. It reduces the complexity of software.
 2. It enhances software maintainability
 3. It enables distributed development among multiple departments (or) organizations.
- **.NET Technology (Network Enable Technology):**
 1. It is a third-generation component model which provides a new level of interoperability Compared to COM technology.
 2. It Enables Application level communication by Microsoft intermediate language(MSIL or CIL) mechanism.
 3. .NET Technology allows true cross-language integration with MSIL.
 4. .NET Technology contains MSIL and also includes other technologies and tools that enable users to develop and implement applications easily.

.NET Languages: The .NET framework is a language neutral. So we can use the number of languages for developing .NET applications. They include

Native to .NET:

- C#
- Visual basic
- C++
- Jscript

Third-Party languages:

- COBOL
- Eiffel
- Perl
- Python
- Small Talk

- Mercury
- Scheme

All .NET languages are not created equal. Some can use the components of other languages, some can use the classes produced in other languages to create objects, and some languages can extend the classes of other languages using the inheritance features of .NET.

Benefits of the .NET Framework

- Simple and faster system development
- Enhanced built in functionality.
- It supports rich oop concepts
- Integration of different applications into one platform.
- Ease of deployment and execution
- Interoperability with existing applications.
- Wide range of scalability
- Simple and easy to build sophisticated development tools.
- Fewer bugs
- Potentially better performance.

Features of .NET:

- Language independency
- Base class library's
- Portability
- COM interoperability
- CLR
- Memory management
- Simplified development

Integrated Development Environment (IDE) for C#: An IDE is a tool that helps you write your programs Microsoft provides the following development tools for C# programming:

- Visual Studio 2010 (VS)
- Visual C# 2010 Express (VCE)
- Visual Web Developer

The last two are freely available from Microsoft official website. Using these tools, you can write all kinds of C# programs from simple command-line applications to more complex applications. You can also write C# source code files using a basic text editor like Notepad, and compile the code into assemblies using the command-line compiler, which is again a part of the .NET Framework.

Visual C# Express and Visual Web Developer Express edition are trimmed down versions

of Visual Studio and has the same appearance. They retain most features of Visual Studio.

INTRODUCING C# (PRONOUNCED "C SHARP"): C# is a simple, modern, general-purpose, object-oriented and type-safe programming language developed by Microsoft, USA and approved by European Computer Manufacturers Association (ECMA) and International Standards Organization (ISO).

C# was developed by **Anders Hejlsberg** and his team during the development of .Net Framework. C# is designed for Common Language Infrastructure (CLI), which consists of the executable code and runtime environment that allows use of various high-level languages on different computer platforms and architectures.

C# can be used to create various types of applications, such as web, windows, console applications or other types of applications using Visual studio.

HISTORY OF C# LANGUAGE: During the development of .NET, the class libraries were originally written in a language called simple managed code (SMC) and later the language had been renamed as C#.

The C# principal designer and lead architect Anders Hejlsberg has previously involved with the design of visual j++, Borland Delphi, Turbo Pascal languages.

The following **reasons make C# a widely used** professional language:

- It is a modern, general-purpose programming language
- It is object oriented.
- It is component oriented.
- It is easy to learn.
- It is a structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- It is a part of .Net Framework.

OVERVIEW OF C#: C# can be used to develop two categories of programs, that is

- **Executable application programs:** These programs are written to carry out certain tasks and require the main method in one of the classes.
- **Component libraries:** These do not require a main declaration because they are not stand-alone application programs. These are written for use by other applications.

CHARACTERISTICS OF C#:

SIMPLE:

1. C# is simple by eliminating some operators such as "::<" or "->" and pointers.

2. C# treats integer and Boolean data types as entirely two different data types. Boolean values are pure true or false values in C# so no more errors of "="operator and "=="operator."==" is used for comparison operation and "=" is used for assignment operation.

MODERN: C# is a modern language because of the following features.

1. It provides Automatic garbage collection.
2. It provides Rich intrinsic model for error handling.
3. It provides Modern approach to debugging
4. It provides robust security model.
5. It provides Decimal data type for financial applications.

OBJECT ORIENTED:

1. C# truly object oriented. It supports Data Encapsulation, inheritance, polymorphism, interfaces.
2. In C#, everything is an object .there are no more global functions, variables and constants.

TYPE SAFE:

1. In C# we cannot perform unsafe casts like convert double to a Boolean.
2. All dynamically allocated objects and arrays are initialized to zero.
3. Usage of any uninitialized variable produces an error message by the compiler.
4. Access to Arrays is range checked and warned if it goes out of boundary.
5. C# checks the Overflow in arithmetic operations.
6. C# supports automatic garbage collection

INTEROPERABILITY: C# provides support for using COM objects, No matter what language was used to develop them. C# also supports a special feature that enables a program to call out any native API.

CONSISTANT: C# supports an unified system, which eliminates the problem of varying integer types. All types are treated as objects and developers can extend the type system simply and easily.

VERSIONABLE: Making new versions of software modules work with the existing applications is known as versioning with the help of new and override keywords, With this support, a programmer can guarantee that his new class library will maintain binary compatibility with the existing client application.

COMPATIBLE: C# enforces the .NET CLS (common language specification) and therefore allows interoperation with other .NET language. C# provides support for transparent access to COM and OLE Applications.

Applications of C#:

- Console applications
- Windows applications
- Developing windows controls
- Developing ASP.NET projects
- Creating web controls
- Providing web services
- Developing .NET component library

NOTES:

- C# is case sensitive.
- All statements and expression must end with a semicolon (;).
- The program execution starts at the Main method.
- Unlike Java, program file name could be different from the class name.

CREATING A SIMPLE C# PROGRAM: A simple code that prints the words "Hello World":

```
using System;
namespace HelloWorldApplication
{
    class HelloWorld
    {
        public static void Main(string[] args)
        {
            Console.WriteLine("Hello World");
            Console.ReadKey();
        }
    }
}
```

Output: Hello World

using System:

using System:

- using is the keyword, which is used to specify namespace.
- The above line tells the compiler to look into the System namespace library for unresolved class names.
- Because of the above line we have not used system prefix to the console class in the output line.
- When the compiler parses the **Console.WriteLine** method, it will understand that the method is undefined, However it will search through the namespaces specified in using directives and upon finding the method in the system namespace ,will compile the code without any

complaint.

Namespace declaration: A namespace is a collection of classes. The HelloWorldApplication namespace contains the class Hello World.

Class Declaration: The next line has a class declaration, the class HelloWorld contains the data and method definitions that your program uses. Classes generally contain multiple methods. Methods define the behaviour of the class. However, the HelloWorld class has only one method Main. The line

class Helloworld:

- The above line Declares a class, C# is true object oriented language and therefore everything must be placed inside a class.
- Class is keyword, which declares new class definition.
- Helloworld is the name of the class. It should be a valid identifier.

Braces({ }): C# is a block structured language and the code always enclosed by { and }.therefore every class definition in c# begins with opening brace { and ends with a closing brace }.

Public static void Main(): The line

public static void Main()

- The above line defines a method named Main. Every C# executable program must include the Main() method in one of the classes. This is the starting point for executing the program. A C# application can have any number of classes but only one class can have the Main() method to initiate the execution.
- The above line contains public, static and void keywords
 - public:** It tells the C# compiler that Main() method is accessible by anyone.
 - static:** The keyword static declares the Main() method is a global one and can be called without creating an instance of the class.
 - void:** The keyword void is a type modifier that tells that the main method does not return any value.
- In contrast to other languages main has capital, not lower case M.

Output line: **System.Console.WriteLine("hi");**

- The WriteLine is the static method of the Console class, Which is located in system namespace.
- . is represented as the member access operator.
- ; is called line terminator
- WriteLine always append a newline character to the end of the string. This means any

subsequent output will start on a newline

COMMENTS: Comments play very important role in the maintenance of programs. They are used to enhance readability and understanding of code. C# supports two types of comments

- **Single line comments:** Single line comments begins with a double backslash (//) symbol and terminate at the end of the line.

Example: //c# program

- **Multiline comments:** If we want to use multiple lines for a comment, This comment starts with /* and terminates with */.

Example: /* this is Example of c#

 Csharp program */

Console.ReadKey (); The line **Console.ReadKey ()**, Makes the program wait for a key press and it prevents the screen from running and closing quickly when the program is launched from Visual Studio .NET.

COMPILING AND EXECUTING THE PROGRAM: If you are using Visual Studio.Net for compiling and executing C# programs, take the following steps:

- Start Visual Studio.
- On the menu bar, choose File -> New -> Project.
- Choose Visual C# from templates, and then choose Windows.
- Choose Console Application.
- Specify a name for your project and click OK button. This creates a new project in Solution Explorer.
- Write code in the Code Editor.
- Click the Run button or press F5 key to execute the project. A Command Prompt window appears that contains the line Hello World.

YOU CAN COMPILE A C# PROGRAM BY USING THE COMMAND-LINE INSTEAD OF THE VISUAL STUDIO IDE:

- Open a text editor and add the above-mentioned code.
- Save the file as helloworld.cs
- Open the command prompt tool and go to the directory where you saved the file.
- Type csc helloworld.cs and press enter to compile your code.
- If there are no errors in your code, the command prompt takes you to the next line and generates **helloworld.exe** executable file.
- Type **helloworld** to execute your program.

- You can see the output Hello World printed on the screen.

Example: Implementation of a Rectangle class and discuss C# basic syntax:

```
using System;
namespace RectangleApplication
{
    class Rectangle
    {
        // member variables
        double length;
        double width;
        public void Acceptdetails()
        {
            length = 4.5;
            width = 3.5;
        }
        public double GetArea()
        {
            return length * width;
        }
        public void Display()
        {
            Console.WriteLine("Length: {0}", length);
            Console.WriteLine("Width: {0}", width);
            Console.WriteLine("Area: {0}", GetArea());
        }
    }
    class ExecuteRectangle
    {
        static void Main(string[] args)
        {
            Rectangle r = new Rectangle();
            r.Acceptdetails();
            r.Display();
            Console.ReadLine();
        }
    }
}
```

Output: Length: 4.5 Width: 3.5 Area: 15.75

USING ALIASES FOR NAMESPACE CLASSES: we have seen how we can avoid the prefix **System** to the **Console** class by implementing the **using System;** statement. can use this approach to avoid prefixing of **System.Console** to the method **WriteLine()** ? The answer is no.

System is a namespace and Console is a class. The using directive can be applied only to the namespaces and cannot be applied to classes. Therefore the statement

using System.Console;

The above statement is illegal. However, we can overcome this problem by using aliases for

namespace classes. This takes the following form.

```
using aliasname=classname;
```

Example:

```
using A= System.Console; //A is alias for System.Console
class sample
{
    public static void Main()
    {
        A.WriteLine("Hello");
    }
}
```

PASSING STRING OBJECTS TO WRITELINE METHOD: So far we have seen constant string output to the Console. we can store string values in string objects and use these objects as parameters to the **WriteLine** method. We can use string data type to create string variable and assign string constant to it.

```
string s="abc";
```

The content of s may be printed out using the WriteLine method.

```
System.Console.WriteLine(s);
```

Example:

```
using System;
class sample
{
    class static void Main()
    {
        string name="c sharp";
        Console.WriteLine(name);
    }
}
```

COMMAND LINE ARGUMENTS: command line arguments are parameters supplied to the Main method at the time of invoking it for execution.

Example:

```
using System;
class sample
{
    public static void Main(string[] args)
    {
        Console.Write("welcome to");
        Console.Write(" "+arg[0]);
        Console.WriteLine(" "+args[1]);
    }
}
```

In this program the Main is declared with a parameter args. The parameter args is declared as an array of strings (known as string objects). Any arguments provided in the command line (at the time of execution) are passed to the array args as its elements. We can access the array elements by using a subscript like args[0], args[1] and so on.

For example, consider the command line => **Sample c sharp**

This command line contains two arguments which are assigned to the array args as follows

```
C      →  args[0]
Sharp →  args[1]
```

If we execute the program with the above command line, we will get the following output

Output: **welcome to c sharp**

WriteLine(): Outputs one (or) more values to the screen but adds a newline character at the end of the output.

Example: Console.WriteLine("hello");
Console.WriteLine("c sharp");

Output: hello
C sharp

Write(): Outputs one (or) more values to the screen without a newline character.

Example: Console.Write("hello");
Console.Write("\n");
Console.Write(" c sharp");

Output: hello
C sharp

MAIN WITH A CLASS:

Example:

```
class testclass
{
    public void fun()
    {
        System.Console.WriteLine("C# is simple");
    }
}
class sample
{
    public static void Main()
    {
        testclass test= new testclass(); // creating test object
        test.fun(); // calling fun() function
    }
}
```

- The above program has two class declarations, one for the testclass and another for the Main method. testclass contains only one method to print a string "C# is simple". The Main method in sample class creates an object of testclass and uses it to invoke the method fun() contained in testclass.
- The object test is used to invoke the method fun() of testclass with the help of the dot operator.

PROVIDING INTERACTIVE INPUT: So far we have seen two approaches for giving values to string objects.

- **Using an assignment statement.**
- **Through command line arguments.**

It is possible to give values to string variables interactively through the keyboard at the time of execution.

Example:

```
using System;
class sample
{
    public static void Main()
    {
        Console.WriteLine("enter your name");
        string name=Console.ReadLine();
        Console.WriteLine("Hello"+name);
    }
}
```

The method “**Console.WriteLine(“enter your name”);**” outputs the message “enter your name” and the method “**Console.ReadLine();**” causes the the execution to wait for the user to enter his name. The moment the user types his name and presses the enter key, the input is read into the string variable name. The second output line “**Console.WriteLine(“Hello”+name);**” concatenates the name-string with hello and presents the resultant string to the user.

Output: enter your name raja
Hello raja

Example1:

```
using System;
class sample
{
    public static void Main()
    {
        Console.WriteLine("what is your name");
        string s=Console.ReadLine();
        Console.WriteLine("Hello"+s);
        Console.WriteLine("what is your age");
        string s=Console.ReadLine();
        Console.WriteLine("you look nice at"+s);
    }
}
```

TOKENS: The smallest, non-reducible, textual elements in a program are referred to as tokens.

The compiler recognizes them for building up expressions and statements. A C# program is a collection of **tokens, comments and whitespaces**. The c# tokens are

- Keywords
- Identifiers
- Literals
- Operators
- Punctuators

KEYWORDS: Keywords are reserved words predefined to the C# compiler. They have fixed

meanings and these meanings cannot be changed. Keywords serve as basic building blocks for program statements. All keywords must be written in lowercase. Keywords cannot be used as identifiers except when they are prefaced by the @ character.

Reserved Keywords						
abstract	as	base	bool	break	byte	case
catch	char	checked	class	const	continue	decimal
default	delegate	do	double	else	enum	event
explicit	extern	false	finally	fixed	float	for
foreach	goto	if	implicit	In	in (generic modifier)	int
interface	internal	is	lock	long	namespace	new
null	object	operator	out	out (generic modifier)	override	params
private	protected	public	readonly	ref	return	sbyte
sealed	short	sizeof	stackalloc	static	string	struct
switch	this	throw	true	try	typeof	uint
ulong	unchecked	unsafe	ushort	using	virtual	void
volatile	while					

In C#, some identifiers have special meaning in context of code, such as get and set, these are called contextual keywords.

Contextual Keywords						
add	alias	ascending	descending	dynamic	from	get
global	group	into	join	let	orderby	partial (type)
partial (method)	remove	select	set			

IDENTIFIERS: An identifier is a name, which is used to identify a class, variable, function, interface, namespace and label. The basic rules for framing identifiers in C# are as follows:

- They can have alphabets, digits (0 - 9), or underscore.
- They must not begin with a digit.
- Upper case and lower case letters are distinct.
- It should not be a C# keyword.

LITERALS: Literals are value constants assigned to variables in a program. C# supports several types of literals.

Numeric literals:

Integer literals: An integer literal refers to a sequence of digits. There are two types of integers, namely **decimal** and **hexadecimal** integers.

Decimal integers: Decimal integers contains a set of digits from **0 to 9** with an optional + or – sign

Example:	Valid examples	Invalid examples
	1. 123	1. 153 43
	2. -76565	2. 20,000
	3. 0	3. \$667
	4. +736	4. 9-898

Hexadecimal integers: A sequence of digits preceded by 0x (or) 0X is considered as a hexadecimal integer. It may also include alphabets A through F (or) a through f. A letter A through F represents the numbers from 10 through 15.

Examples:	Valid examples	Invalid examples
	1. 0x2	1. 0xnmjk
	2. 0x87665	2. 0x1286m
	3. 0xabc	3. 08972
	4. 0x53647	4. 0khjdi

Real literals: The numbers containing fractional part is called real (or) floating-point literals. We have two Notations for representing real literals.

- Decimal notation
- Scientific notation (or) exponential notation

Decimal notation: It represents numbers having a whole number followed by decimal point and fractional part.

Example: 215.
.95
-.75
+.5

Scientific notation: A real number may also be expressed in exponential (or) scientific notation.

Example: The number 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by 10^2 .

Syntax: mantissa e exponent

Mantissa: mantissa is either a **real number** expressed in **decimal notation**.

Exponent: It is an **integer number** with an **optional sign**.

E (or) e: It separates the **mantissa** and the **exponent** can be written in either **uppercase (or) lower case**.

Example: 0.65e4
12e-2
1.5e+5
3.18e3

Exponential notation is useful for representing the numbers that are either very large (or) very small in magnitude.

Example: 7500000000 is equals to 7.5e9 is also equals to 75e8
-0.000000368 is equals to -3.68e-7

Boolean literals: There are two Boolean literal values

true
false

Character literals:

Single Character Constants: A single character constant contains a single character enclosed by single quote marks is called single character constants.

Examples: 'a'
'#'
'2'
, ,

String Constants: A string constant is a sequence of characters enclosed by double quotes. The character may be any letter, digits, special characters and blank space .

Example: "Welcome to c programming"
"2001"
"Well done"
"34+23"
"d"

BACKSLASH CHARACTER CONSTANTS: C# supports some special backslash character constants that are used in output methods. Each backslash character constant contains **two characters** to represent a single character. These combinations are called **escape sequences**.

1. \a – alert
2. \b – Backspace
3. \f – Form feed
4. \n – New line
5. \r – Carriage return
6. \t – Horizontal tab
7. \v – Vertical tab
8. \\ – Backslash
9. \' – Single quote
10. \" – Double quote
11. \0 - Null

DATA TYPES: Every variable in C# is associated with a data type. Data types specify the and size of values that a variable can be stored. The data types in C# are primarily divided into two types.:

- Value types
- Reference types

Value types and reference types differ in two characteristics

- Where they are stored in the memory
- How they behave in the context of assignment statements.

Value Type: Value types (which are of fixed length) are stored on the stack(stack is a place where the data stores in fixed length like float,int etc.), when a value of a variable is assigned to a variable the value is actually copied. This means that two identical copies of the value are available in memory.

The value types of C# can be grouped into two categories, namely,

- **User defined types (or) complex types**
- **Predefined types (or) simple types**

User defined types (or) complex types: Here we can define our own complex types known as User defined value types, which includes

- structures
- enumerations

Predefined types: These are also known as **simple types (or) primitive types**. Predefined types are divided into

- Numeric types
- Boolean types
- Character types

Numeric types: Numeric types include integral types, floating point types and decimal types.

Integral types: Integral types can hold whole numbers such as 123, -96 and 5639. The size of the values that can be stored depends on the integral data type we choose. C# supports the concept of unsigned types and therefore it supports eight types of integers .

Signed integers: signed integer types can hold both positive and negative numbers.

Type	Size	Minimum value	Maximum value
sbyte	1 byte	-128	+127
short	2 byte	-32768	+32767
int	4 byte	2147483648	+2147483647
long	8 byte	-9223372036854775808	+9223372036854775807

Unsigned integers: unsigned integer types can hold only positive numbers

Type	Size	Minimum value	Maximum value
byte	1 byte	0	255
ushort	2 byte	0	65535
uint	4 byte	0	4294967295
ulong	8 byte	0	18446744073709551615

All integers are by default int type. In order to specify other integer types, we must append the characters U,L or UL

- 123U(for uint type)
- 123L(for long type)
- 123UL(for ulong type)

Floating point types: Floating point types hold numbers containing fractional parts such as 27.59 and -1.375. There are two kinds of Floating point storage in C#

The Floating point values are single-precision numbers with a precision of seven digits.

The double types represent double-precision numbers with a precision of 15/16 digits

Type	Size	Minimum value	Maximum value
float	4 byte	1.5×10^{-45}	3.4×10^{38}
double	8 byte	5.0×10^{-324}	1.7×10^{308}

Floating point numbers by default as double-precision quantities. to force them to be in single precision mode, we must append f (or) F to the numbers.

Example: 1.23f
7.56923f

Double-precision types are used when we need greater precision in storage of floating-point numbers.

Floating-point data types support a special value known as Not-a-Number (NaN). NaN is used to represent the result of operations such as dividing zero by zero, where an actual number is not produced. Most operations NaN as an operand will produce NaN as result.

Decimal type: The decimal type is a high precision, 128-bit data type that is designed for use in financial monetary calculations. It can store values in the range 1.0×10^{-28} to 7.9×10^{28} with 28 significant digits.

To specify a number to be decimal type, we must append the character M or m to the value.

Example: 123.45M (if we omit M, the value will be treated as double).

Character type: In order to store single characters in memory, C# provides a character data type called **char**. The **char** type assumes a size of two bytes but, in fact it can hold only a single character. char data type has been designed to hold a 16-bit Unicode character, in which 8-bit ASCII code is a subset.

Boolean type: Boolean condition can be used when we want to test a particular condition during the execution of the program. There are two values that a Boolean type can take true (or) false. Boolean type can be denoted by the keyword bool and uses only one bit of storage.

Reference types: reference types (which are of variable length) are stored on the heap, and when an assignment between two reference variables occurs, only the reference is copied. The actual values remains in the same memory location. This means there are two references to a single value. The reference types can be divided into two groups

- User defined types (or) complex types
- Predefined types (or) simple types

User defined types: user defined reference types refer to those types which we define using predefined type. They include

- Classes
- Delegates

- Interfaces
- Arrays

Predefined types: Predefined types include two types

- Object type
- String type

VARIABLES: A variable is an identifier that denotes a storage location used to store a data value. Unlike constants that remain unchanged during the execution of the program, a variable may take different values at different times during the execution of the program. Every variable has a type that determines what values can be stored in the variable.

Rules for forming a variable:

- They must not begin with a digit.
- Uppercase and lower case is distinct. This means that the variable TOTAL is not same as total or Total.
- It should not be a keyword.
- White space is not allowed.
- Variable names can be of any length.

Variable declaration: After designing suitable variable names, we must declare the variable before it is used in the program. Declaration does three things

- It tells the compiler what the variable name is.
- It specifies what type of data the variable will hold.
- The place of declaration in the program decides the scope of the variable.

Syntax: **data_type variable1, variable2,..... variableN;**

Here, data_type must be a valid C# data type including char, int, float, double, or any user-defined data type, and variables are separated by commas. Some valid variable definitions are shown here:

```
int i, j, count;
float f, salary;
double pi;
byte b;
char c1,c2,c3;
decimal d1,d2;
uint m;
ulong n;
```

Initializing Variables: The process of giving initial values to the variables is called initialization. Once the declaration has been done then initializes the variables with the assignment operator. The general form of initialization is:

Syntax: **variable_name = value;**

It is also possible to assign a value at the time of its declaration.

Syntax: <data_type> <variable_name> = value;

Example:

```
int d = 3, f = 5;
byte z = 22;
double pi = 3.14159;
char x = 'x';
```

Example:

```
float x,y,z;           //declares three variables
int m=5,n=10;         //declares and initializes two int variables
int m, n=10;         //declares m and n and initializes n.
```

Example: The following example uses various types of variables:

```
using System;
namespace VariableDefinition
{
    class Program
    {
        static void Main(string[] args)
        {
            short a; int b ; double c;
            /* actual initialization */
            a = 10; b = 20; c = a + b;
            Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c);
            Console.ReadLine();
        }
    }
}
```

Output: a = 10, b = 20, c = 30

Scope of variables: The scope of the variable determines over what part(s) of the program a variable is actually available for use (active). This depends on the type of the variable and place of its declaration. C# defines several categories of variables.

- Static variables
- Instance variables
- Array elements
- Value parameters
- Reference parameters
- Output parameters
- Local variables

Example:

```
class ABC
{
    static int m;
    int n;
    void fun (int x , ref int y, out int z, int[] a)
    {
        int j = 10;
    }
}
```

- ✓ m = static variable,
- ✓ n = instance variable,
- ✓ x = value parameter,
- ✓ y = reference parameter,
- ✓ j = local variable,
- ✓ z =output parameter,
- ✓ a[0] = array element

TYPE CONVERSION: The process of converting one data type into another data type is known as type casting or type conversion. Type conversions are divided into two types.

1. **Implicit (or) up casting (or) widening (or) automatic type conversion.**
2. **Explicit (or) down casting (or) narrowing type conversion.**

Implicit type conversion (Automatic type conversion): Converting lower data type into higher data type is called widening. In this one data type is automatically converted into another type as per the rules described in c# language, which means the lower level data type is converted automatically into higher level data type before the operation proceeds. The result of the data type having higher level data type.

Example: byte x = 12;
 int y = x;

In the above statement, the conversion of data from byte to int is done implicitly, in other words programmers don't need to specify any type operators. Widening is safe because there will not be any loss of data. This is the reason even though the programmer does not use the cast operator the compiler does not complaint because of lower data type is converting into higher data type. Here higher data type having the much more space to store the lower data type.

Example: using System;
 namespace explicit_cast_conversion
 {
 class Program
 {
 static void Main(string[] args)
 {
 int num1=10;
 long num2=num1;
 Console.WriteLine("num2 value is : " +num2);
 Console.ReadLine();
 }
 }
 }

Explicit type conversion: Converting higher data type into lower data type is called narrowing.

Here we can place intended data type in front of the variable to be cast.

Syntax: data-type variablename1=(cast-type)variablename2;

Example: double d=12.67853;

```
int n = (int) d;
```

Here we are converting higher level data type into lower level data type that means double type is converted into int type, the fractional part of the number is lost and only 12 is stored in n. Here we are losing some digits this is the reason the compiler forces the programmer to use the cast operator when going for explicit casting.

Example: The following example shows an explicit type conversion:

```
using System;
namespace TypeConversionApplication
{
    class ExplicitConversion
    {
        static void Main(string[] args)
        {
            double d = 5673.74;
            int i;
            i = (int)d;
            Console.WriteLine(i);
            Console.ReadKey();
        }
    }
}
```

Output: 5673

Example:

```
using System;
namespace explicit_cast_conversion
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1, num2;
            float avg;
            num1 = 10;
            num2 = 21;
            avg = (float)(num1 + num2) / 2;
            Console.WriteLine("average is : " + avg);
            Console.ReadLine();
        }
    }
}
```

Microsoft .NET provides three ways of type conversion:

- **Parsing**
- **Convert Class**
- **Boxing and unboxing**

parsing: Parsing is used to convert string type data to primitive value type. For this we use parse methods with value types.

Example:

```

using System;
class Program
{
    static void Main()
    {
        string text = "500";// Convert string to number.
        int num = int.Parse(text);
        Console.WriteLine(num);
        int num2 = int.Parse(Console.ReadLine());
        Console.WriteLine(num2);
    }
}

```

Example:

```

using System;
namespace parsing
{
    class Program
    {
        static void Main(string[] args)
        {
            int number;
            float weight;
            Console.Write("Enter any number : ");
            number = int.Parse(Console.ReadLine());
            Console.Write("Enter your weight : ");
            weight = float.Parse(Console.ReadLine());
            Console.WriteLine("You have entered : " + number);
            Console.WriteLine("You weight is : " + weight);
            Console.ReadLine();
        }
    }
}

```

Convert class: It is used to convert **one primitive type to another primitive type**. Convert is the predefined class. C# provides the following built-in type conversion methods as described:

S.No	Method	Description
1	ToBoolean	Converts a type to a Boolean value, where possible
2	ToByte	Converts a type to byte.
3	ToChar	Converts a type to single Unicode character, where possible.
4	ToDateTime	Converts a type to date time structures.
5	ToDecimal	Converts a floating point or integer type to decimal type.
6	ToDouble	Converts a type to double type.
7	ToInt16	Converts a type to a 16-bit integer.
8	ToInt32	Converts a type to a 32-bit integer.
9	ToInt64	Converts a type to a 64-bit integer.
10	ToSbyte	Converts a type to a signed byte type.
11	ToSingle	Converts a type to a small floating point number.
12	ToString	Converts a type to a string.
13	ToType	Converts a type to a specified type.
14	ToUInt16	Converts a type to unsigned int type.

15	ToUInt32	Converts a type to unsigned long type.
16	ToUInt64	Converts a type to unsigned big integer.

Example: The following example converts various value types to string type:

```
using System;
namespace TypeConversionApplication
{
    class StringConversion
    {
        static void Main(string[] args)
        {
            int i = 75;
            float f = 53.005f;
            double d = 2345.7652;
            bool b = true;
            Console.WriteLine(i.ToString());
            Console.WriteLine(f.ToString());
            Console.WriteLine(d.ToString());
            Console.WriteLine(b.ToString());
            Console.ReadKey();
        }
    }
}
```

Output:
75
53.005
2345.7652
True

Example:

```
using System;
namespace Arithmetic_Operators
{
    class Program
    {
        static void Main(string[] args)
        {
            int num1, num2;
            int add, sub, mul;
            Console.Write("Enter first number\t\t");
            num1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("\n\nEnter second number\t\t");
            num2 = Convert.ToInt32(Console.ReadLine());
            add = num1 + num2;
            sub = num1 - num2;
            mul = num1 * num2;
            Console.WriteLine("\n\n=====");
            Console.WriteLine("Addition\t\t{0}", add);
            Console.WriteLine("Subtraction\t\t{0}", sub);
            Console.WriteLine("Multiplication\t\t{0}", mul);
            Console.WriteLine("\n\n=====");
            Console.ReadLine();
        }
    }
}
```



```
Console.WriteLine(om); //om=10
```

When a code changes the value of m, the value of om not affected.

Unboxing: Unboxing is the process of converting a reference type to value type. Remember that we can only unbox a variable that has previously been boxed. Unboxing is an explicit type conversion.

When unboxing a value, we have to ensure that the value type is large enough to hold the value of the object (reference type value). Otherwise, the operation may result in runtime error.

Example:

```
int m=10;
object om=m;
byte n=(byte)om;
```

The above code will produce a runtime error.

Notice that when unboxing, we need to use explicit cast. This is because in case of unboxing, an object could be cast to any type. Therefore, the cast operator is necessary for the compiler to verify that it is valid as per the specified value type.

Example:

```
int m = 10;
Object om = m; //Boxing
int i = (int)om; //Unboxing
```

The first two lines show how to Box a Value Type. The next line (int i = (int) om) shows extracts the Value Type from the Object (reference type) . That is converting a value of a Reference Type into a value of a Value Type. This operation is called Unboxing.

A c-style cast for unboxing

Example:

```
int m=10;
object om=m;
int n=(int)om;
using System;
namespace boxing
{
    class Program
    {
        static void Main(string[] args)
        {
            int i = 10;
            object a = i; // boxing
            int j = (int)a; // unboxing
            Console.WriteLine("value of o object : " + a);
            Console.WriteLine("Value of j : " + j);
            Console.ReadLine();
        }
    }
}
```

OPERATORS: An operator is a symbol that tells the computer to perform specific mathematical or logical manipulations. Operators are used in programs to manipulate data and variables. C# has rich set of operators. C# operators can be classified into the following types:

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators
- Increment and decrement operators
- Conditional operators
- Bitwise Operators
- Special Operators

Arithmetic Operators: The operators that are used to perform **Arithmetic** operations such as addition, subtraction, multiplication, ----etc. are called **Arithmetic** operators. Assume variable A holds 10 and variable B holds 20 then:

Operator	Meaning	Description	Example
+	Addition operator	Adds two operands	a+b=30
-	Subtraction operator	Subtracts second operand from the first	a-b=-10
*	Multiplication operator	Multiplies both operands	a*b=200
/	Division operator	Divides numerator by de-numerator	b/a=2
%	Modulus division operator	It gives Remainder after an integer division	b%a=0

Integer Arithmetic: When both the operands in a single arithmetic expression such as a+b are integers, the expression is called as integer expression and the operation is called integer arithmetic

The a=4, b=2, then

$$a+b = 6$$

$$a-b = 2$$

$$a*b = 8$$

$$a/b = 2$$

$$a\%b = 0$$

Real Arithmetic: An arithmetic operation involving only real operands is called real arithmetic. A real operand may assume values either in decimal or exponential notation.

Example: a=20.5F,b=6.4F

$$a+b = 26.9$$

$$a-b = 14.1$$

$$a*b = 131.2$$

$$a/b = 3.203125$$

$$a\%b = 1.3$$

Example:

Mixed-mode Arithmetic: When one of the operands is real and the other is integer, the expression is called a mixed-mode arithmetic expression. If either operand is real type, then the other operand is converted to real and real arithmetic is performed. The result will be real.

Example: 15/10.0=1.5
 15.0/10=1

Example: The following example demonstrates all the arithmetic operators available in C#:

```

using System;
namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 21;
            int b = 10;
            int c;
            c = a + b;
            Console.WriteLine("Line 1 - Value of c is {0}", c);
            c = a - b;
            Console.WriteLine("Line 2 - Value of c is {0}", c);
            c = a * b;
            Console.WriteLine("Line 3 - Value of c is {0}", c);
            c = a / b;
            Console.WriteLine("Line 4 - Value of c is {0}", c);
            c = a % b;
            Console.WriteLine("Line 5 - Value of c is {0}", c);
            c = a++;
            Console.WriteLine("Line 6 - Value of c is {0}", c);
            c = a--;
            Console.WriteLine("Line 7 - Value of c is {0}", c);
            Console.ReadLine();
        }
    }
}

```

Output: Line 1 - Value of c is 31
 Line 2 - Value of c is 11
 Line 3 - Value of c is 210
 Line 4 - Value of c is 2
 Line 5 - Value of c is 1
 Line 6 - Value of c is 21
 Line 7 - Value of c is 22

Relational Operators: These operators are used to compare the value of two variables. Relational operator's checks relationship between two operands. If the relation is true, it returns value 1 and if the relation is false, it returns value 0. Assume variable A holds 10 and variable B holds 20, then:

Operator	Meaning	Description	Example
>	Greater than	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
<	Less than	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
<=	Less than or equals to	Checks if the value of left operand is less than or	a<=b is

		equal to the value of right operand, if yes then condition becomes true.	true
>=	Greater than or equals to	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
!=	Not equals to	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	a!=b is true
==	Equals to	Checks if the values of two operands are equal or not, if yes then condition becomes true.	a==b is not true

Example: The following example demonstrates all the relational operators available in C#:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 21;int b = 10;
        if (a == b)
            Console.WriteLine("Line 1 - a is equal to b");
        else
            Console.WriteLine("Line 1 - a is not equal to b");
        if (a < b)
            Console.WriteLine("Line 2 - a is less than b");
        else
            Console.WriteLine("Line 2 - a is not less than b");
        if (a > b)
            Console.WriteLine("Line 3 - a is greater than b");
        else
            Console.WriteLine("Line 3 - a is not greater than b");
        a = 5;
        b = 20;
        if (a <= b)
            Console.WriteLine("Line 4 - a<=b");
        if (b >= a)
            Console.WriteLine("Line 5-b >=b");
    }
}
```

Output: Line 1 - a is not equal to b
Line 2 - a is not less than b
Line 3 - a is greater than b
Line 4 - a is either less than or equal to b
Line 5 - b is either greater than or equal to b

Logical Operators: These operators are used to perform logical operations on the given two variables. Logical operators are used to combine expressions containing relation operators. Assume variable A holds Boolean value true and variable B holds Boolean value false, then:

Operator	Meaning	Description	Example
&&	Logical AND	If both operands are non-zero then condition	(a&&b) is false

		becomes true	
	Logical OR	If any of the two operands is non-zero then condition becomes true	(a b) is true
!	Logical NOT	Used to reverse the logical state of its operand. If a condition is true then logical NOT operator will make false.	!(a&&b) is true

Example: The following example demonstrates all the logical operators available in C#:

```
using System;
namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            bool a = true;
            bool b = true;
            if (a && b)
                Console.WriteLine("Line 1 - Condition is true");
            if (a || b)
                Console.WriteLine("Line 2 - Condition is true");
            /* lets change the value of a and b */
            a = false;
            b = true;
            if (a && b)
                Console.WriteLine("Line3-Condition is true");
            else
                Console.WriteLine("Line3-not true");
            if (!(a && b))
                Console.WriteLine("Line4-Condition is true");
            Console.ReadLine();
        }
    }
}
```

Output: Line 1 - Condition is true
 Line 2 - Condition is true
 Line 3 - Condition is not true
 Line 4 - Condition is true

Assignment Operators: These are used to assign the values for the variables in C# programs. The most common assignment operator is =. The syntax is shown below:

data type Variable_name = expression;

Operator	Meaning	Example	Same as
=	Assignment	a=b	a=b
+=	Addition assignment	a+=b	a=a+b
-=	Subtraction assignment	a-=b	a=a-b

Operator	Meaning	Example	Same as
*	Multiplication assignment	a*=b	a=a*b
/	Division assignment	a/=b	a=a/b
%	Modula division assignment	a%=b	a=a%b

Example: The following example demonstrates all the assignment operators available in C#:

```
using System;
namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 21;
            int c;
            c = a;
            Console.WriteLine("Line 1 - =Value of c = {0}", c);
            c += a;
            Console.WriteLine("Line 2 - += Value of c = {0}", c);
            c -= a;
            Console.WriteLine("Line 3 - -=Value of c = {0}", c);
            c *= a;
            Console.WriteLine("Line 4 - *=Value of c = {0}", c);
            c /= a;
            Console.WriteLine("Line 5 - /=Value of c = {0}", c);
            c = 200;
            c %= a;
            Console.WriteLine("Line 6 - %=Value of c = {0}", c);
            Console.ReadLine();
        }
    }
}
```

Output:

```
Line 1 - =Value of c = 21
Line 2 - += Value of c = 42
Line 3 - -= Value of c = 21
Line 4 - *= Value of c = 441
Line 5 - /= Value of c = 21
Line 6 - %= Value of c = 11
```

Increment/Decrement Operators: Increment operators are used to increase the value of the variable by one and decrement operators are used to decrease the value of the variable by one in C programs.

Decrement operator: --var_name; (or) var_name --;

Increment Operators:

1. ++ is an increment operator. This is an unary operator.
2. It increments the value of the operand by one.

Syntax: Increment operator: ++var_name; (or) var_name++;

Post Increment: If the increment operator ++ is placed after the operand, then operator is called Post Increment. As the name indicates Post indicates means increment after the operand value is used.

Pre Increment: If the increment operator ++ is placed before the operand. Then operator is called Pre Increment. As the name indicates Pre indicates, increment before the operand value is used.

Decrement Operators:

- is a decrement operator. This is an unary operator.
- It decrements the value of the operand by one.

The decrement operator is classified into two categories:

Post Decrement(Ex: a--): If the decrement operator -- is placed after the operand. Then the operator is called post decrement. So the operand value is used first and then the operand value is decremented by 1.

Pre Decrement(Ex:--a): If the decrement operator -- is placed before the operand then the operator is called Pre decrement. So the operand value is decrement by 1 first and this decremented value is used.

Example:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int i = 4;
        i++;
        Console.WriteLine(i);
        ++i;
        Console.WriteLine(i);
        i--;
        Console.WriteLine(i);
        --i;
        Console.WriteLine(i);
    }
}
```

Example:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int i = 4;int y = 0;
        y = i++;
        Console.WriteLine("i = " + i);
        Console.WriteLine("y = " + y);
        Console.WriteLine();
        ++i;
    }
}
```

```

        Console.WriteLine("i = " + i);
        Console.WriteLine("y = " + y);
        Console.WriteLine();
        y = i--;
        Console.WriteLine("i = " + i);
        Console.WriteLine("y = " + y);
        Console.WriteLine();
        y = --i;
        Console.WriteLine("i = " + i);
        Console.WriteLine("y = " + y);
    }
}

```

Conditional Operators: The conditional operator is also called a ternary operator. As the name indicates an operator that operates on three operands is called ternary operator. The ternary operators consisting two symbols i.e ? and :

Syntax: (Condition? true_value: false_value);

Where condition is an expression evaluated True or False. If condition is evaluated to True, the true_value is executed. If condition is evaluated to False, the false_value is executed.

Bitwise Operator: A bitwise operator works on each bit of data. Bitwise operators are used in bit level programming

x	y	x y	x & y	x ^ y	Operator	Description
0	0	0	0	0	&	Bitwise AND
0	1	1	0	1		Bitwise OR
1	0	1	0	1	~	One's compliment
1	1	1	1	0	^	XOR
					<<	Left Shift
					>>	Right Shift

Example: The following example demonstrates all the bitwise operators available in C#:

```

using System;
namespace OperatorsAppl
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 60; /* 60 = 0011 1100 */
            int b = 13; /* 13 = 0000 1101 */
            int c = 0;
            c = a & b;          /* 12 = 0000 1100 */
            Console.WriteLine("Line 1 - Value of c is {0}", c);
            c = a | b;         /* 61 = 0011 1101 */
            Console.WriteLine("Line 2 - Value of c is {0}", c);
            c = a ^ b;        /* 49 = 0011 0001 */
        }
    }
}

```



```

        Console.WriteLine("The size of int is {0}", sizeof(int));
        Console.WriteLine("The size of short is {0}", sizeof(short));
        Console.WriteLine("The size of double is {0}", sizeof(double));
        /* example of ternary operator */
        int a, b;
        a = 10;
        b = (a == 1) ? 20 : 30;
        Console.WriteLine("Value of b is {0}", b);
        b = (a == 10) ? 20 : 30;
        Console.WriteLine("Value of b is {0}", b);
        Console.ReadLine();
    }
}

```

Output:

```

The size of int is 4
The size of short is 2
The size of double is 8
Value of b is 30
Value of b is 20

```

CONDITIONAL STATEMENT: A block of code that is executed based on a condition is called conditional statement. These are divided into two types.

- **Conditional branching**
- **Conditional looping**

When a program breaks the sequential flow and jumps to another part of the code, it is called branching. Branching statements are used to execute a statement or a group of statements

Conditional branching: when branching takes place based on certain condition is called (or) conditional branching. The conditional branching statements are.

- if→simple if, if-else, if-else if-else, nested-if
- switch

Unconditional branching: when branching takes place without any condition is called unconditional branching. The unconditional branching statements are

- break
- continue
- goto
- return

Simple-if: A simple-if statement is used to execute a block of code only if condition is true.

Syntax:

```

if( boolean_expression (Or) condition)
{
    True-block-statement(s);
}

```

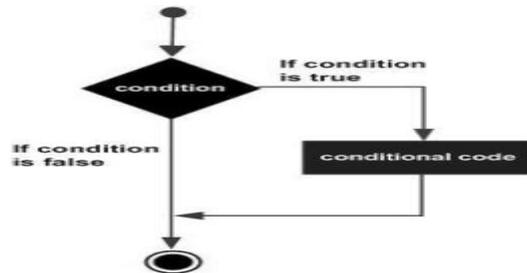
```

}

```

If the Boolean expression evaluates to true, then the block of code inside the simple-if statement is executed. If Boolean expression evaluates to false, the execution will jump to the statements after if statement.

Flow Diagram



Example: using System;
namespace DecisionMaking
{
class Program
{
static void Main(string[] args)
{
int age = 10;
if (a >=18)
Console.WriteLine("you are eligible for voting");
Console.WriteLine("Thank you");
Console.ReadLine();
}
}
}

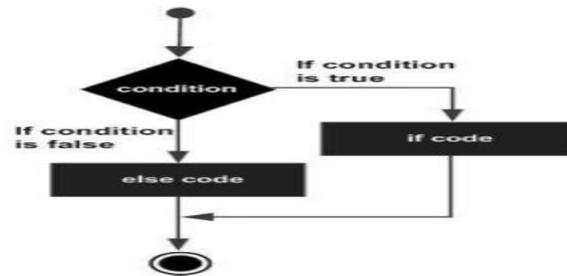
Output: Thank you

if...else Statement: The if-else statement is the extension of simple-if statement. The if else statement is used to execute true-block-statements if condition is true, otherwise it will execute the else false-block-statements.

Syntax: if(boolean_expression(or) condition)
{
True-block-statement(s);
}
else
{
False-block-statement(s) will execute if the Boolean expression is false */
}

If the Boolean expression evaluates to true, then the true block of code is executed, otherwise false-block-statements is executed.

Flow Diagram



Example: using System;

```
namespace DecisionMaking
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            int a = 100;
```

```
            if (a < 20)
```

```
                Console.WriteLine("a is less than 20");
```

```
            else
```

```
                Console.WriteLine("a is not less than 20");
```

```
            Console.WriteLine("value of a is : {0}", a);
```

```
            Console.ReadLine();
```

```
        }
```

```
    }
```

```
}
```

Output: a is not less than 20;
value of a is : 100

if...else if...else Statement: It is also called as multi-way decision statement .In this the conditions are evaluated from top to downwards, when the true condition is found, the statements associated with it is executed. When all the n conditions become false, then the final contains default statement will be executed.

Syntax: if(boolean_expression 1)
 Statement-1;
else if(boolean_expression 2)
 Statement-2;
else if(boolean_expression 3)
 Statement-n;
else
 Default-Statement;

Example: using System;

```
namespace DecisionMaking
```

```
{
```

```
    class Program
```

```
    {
```

```
        static void Main(string[] args)
```

```
        {
```

```
            int a = 100;
```

```
            if (a == 10)
```

```

        Console.WriteLine("Value of a is 10");
    else if (a == 20)
        Console.WriteLine("Value of a is 20");
    else if (a == 30)
        Console.WriteLine("Value of a is 30");
    else
        Console.WriteLine("None of the values is matching");
    Console.WriteLine("Exact value of a is: {0}", a);
    Console.ReadLine();
}
}
}

```

Output: None of the values is matching
Exact value of a is: 100

Example:

```

using System;
namespace if_else_construct
{
    class Program
    {
        static void Main(string[] args)
        {
            int opt, num1, num2;
            float result;
            label:
            Console.WriteLine("\n\tMenu");
            Console.WriteLine("\nPress 1 for add");
            Console.WriteLine("Press 2 for subtraction");
            Console.WriteLine("Press 3 for multiplication");
            Console.WriteLine("Press 4 for Division");
            Console.Write("\n\nEnter first number:\t");
            num1 = Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter second number:\t");
            num2 = Convert.ToInt32(Console.ReadLine());
            Console.Write("\nEnter your option:\t");
            opt = Convert.ToInt32(Console.ReadLine());
            if (opt == 1)
            {
                result = num1 + num2;
                Console.WriteLine("\n {0} + {1} = {2}", num1,
                    num2, result);
            }
            else if (opt == 2)
            {
                result = num1 - num2;
                Console.Write("\n {0} - {1} = {2}", num1, num2, result);
            }
            else if (opt == 3)
            {
                result = num1 * num2;
            }
        }
    }
}

```

```

        Console.WriteLine("\n{0}x{1}={2}", num1, num2, result);
    }
    else if (opt == 4)
    {
        result = (float)(num1 / num2);
        Console.WriteLine("\n{0} / {1}={2}", num1, num2, result);
    }
    else
    {
        Console.WriteLine("Invalid option. Try again");
        goto label;
    }
    Console.ReadLine();
}
}
}

```

Example:

```

using System;
class BiggestNumber
{
    static void Main()
    {
        int a = int.Parse(Console.ReadLine());
        int b = int.Parse(Console.ReadLine());
        int c = int.Parse(Console.ReadLine());

        if (a > b && a > c)
        {
            Console.WriteLine("The biggest number is: {0}", a);
        }
        else if (b > a && b > c)
        {
            Console.WriteLine("The biggest number is: {0}", b);
        }
        else
        {
            Console.WriteLine("The biggest number is: {0}", c);
        }
    }
}

```

Example: The program finds greatest of three numbers and then prints the number which is the greatest. If all 3 input numbers are same then it prints "Entered Numbers are not Distinct."

```

using System;
namespace DotNetMirror
{
    class GreatestOfThreeNumbers
    {
        static void Main()
        {

```

```

int number1, number2, number3;
Console.WriteLine("Enter three numbers (followed by Enter key): ");
number1 = Convert.ToInt16(Console.ReadLine());
number2 = Convert.ToInt16(Console.ReadLine());
number3 = Convert.ToInt16(Console.ReadLine());
if (number1 > number2 && number1 > number3)
{
    Console.WriteLine("Number {0} is largest.", number1);
}
else if (number2 > number1 && number2 > number3)
{
    Console.WriteLine("Number {0} is largest.", number2);
}
else if (number3 > number1 && number3 > number2)
{
    Console.WriteLine("Number {0} is largest.", number3);
}
else
{
    Console.WriteLine("Entered Numbers are not Distinct.");
}
Console.ReadLine();
}
}
}

```

Nested if Statements: The statements within the if statement can contain another if statement and which in turn may contain another if and so on is called nested if-else statement. The nested if statement is used when multiple conditions needs to be tested. The inner statement will execute only when outer if statement is true otherwise control won't even reach inner if statement

Syntax: if(boolean_expression1)

```

{
    if(boolean_expression2)
    {
        Statement-1;
    }
    else
    {
        Statement-2;
    }
}
else
{
    Statement-3;
}

```

Example: using System;
namespace DecisionMaking

```

{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 100;
            int b = 200;
            if (a == 100)
            {
                if (b == 200)
                    Console.WriteLine("Value of a is 100 and b is 200");
            }
            Console.WriteLine("Exact value of a is : {0}", a);
            Console.WriteLine("Exact value of b is : {0}", b);
            Console.ReadLine();
        }
    }
}

```

Output: Value of a is 100 and b is 200
Exact value of a is : 100
Exact value of b is : 200

Example: using System;

namespace DecisionMaking

```

{
    class Program
    {
        public static void Main()
        {
            double n1, n2, n3;
            Console.WriteLine("Enter three numbers: ");
            n1=Console.ReadLine();
            n2=Console.ReadLine();
            n3=Console.ReadLine();
            if (n1>=n2)
            {
                if(n1>=n3)
                    Console.WriteLine ("%%.2lf is the largest number.", n1);
                else
                    Console.WriteLine ("%%.2lf is the largest number.", n3);
            }
            else
            {
                if(n2>=n3)
                    Console.WriteLine ("%%.2lf is the largest number.", n2);
                else
                    Console.WriteLine ("%%.2lf is the largest number.",n3);
            }
        }
    }
}

```

```

}
Example:    using System;
              namespace NestingIfStatementsDemo
              {
                  public class Program
                  {
                      public static void Main()
                      {
                          int age;
                          string gender;
                          Console.Write("Enter your age: ");
                          age = Convert.ToInt32(Console.ReadLine());
                          Console.Write("Enter your gender (male/female): ");
                          gender = Console.ReadLine();
                          if (age > 12)
                          {
                              if (age < 20)
                              {
                                  if (gender == "male")
                                      Console.WriteLine("teenage boy.");
                                  else
                                      Console.WriteLine("teenage girl.");
                              }
                              else
                              {
                                  Console.WriteLine("already an adult.");
                              }
                          }
                          else
                          {
                              Console.WriteLine("You are still too young.");
                          }
                      }
                  }
              }

```

Switch Statement: A switch statement allows a variable or value of an expression to be tested for equality against a list of possible case values and when match is found, the block of code associated with that case is executed. Otherwise default case is executed.

Syntax:

```

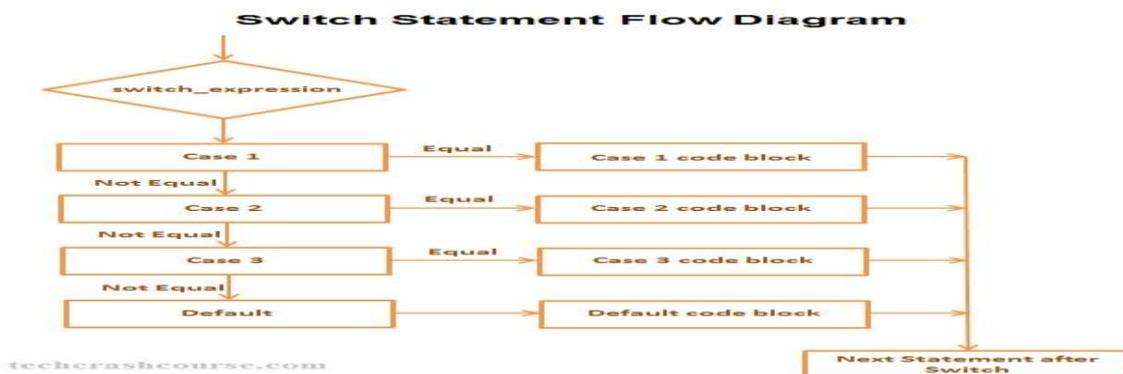
switch (expression)
{
    case constant1: statements_block1;
        break;
    case constant2: Statements_lock2;
        break;
    default: default_block
}

```

- The expression is evaluated first.

- The value of the expression is compared against the constants, constant1, constant2...If a case is found whose value matches the value of the expression, then the block of statements that follows the case are executed.
- The break statement at the end of each block signals the end of a particular case and causes an exit from the switch statement, transferring the control to the statements following the switch.
- The default is an optional case. when present, it will be executed if the value of the expression does not match any of the case values. If not present, no action takes place when all matches fail and the control goes to the statements following the switch

Flow Diagram



Rules for switch statement:-

1. The switch expression must be an integral Type (**integer, character**)
2. Case labels must be consonants or consonant expression.
3. Case label must be of integral Type (Integer, Character).
4. Case labels must be unique. No two labels can have the same values.
5. Case Labels must end with Colon.
6. The break statements transfer the control out of switch statement.
7. The break statement is optional. i.e; two or more case labels may belong to the same statements.
8. The default label is optional if present it will be executed when the expression does not find a matching case label.
9. There can be at most one default label.
10. The default may be placed anywhere but usually placed at the end.
11. Nesting (switch within switch) is allowed

Example: using System;
 namespace DecisionMaking
 {
 class Program

```

    {
        static void Main(string[] args)
        {
            char grade = 'B';
            switch (grade)
            {
                case 'A':Console.WriteLine("Excellent!");
                    break;
                case 'B':
                case 'C':Console.WriteLine("Well done");
                    break;
                case 'D':Console.WriteLine("You passed");
                    break;
                case 'F':Console.WriteLine("Better try again");
                    break;
                default:Console.WriteLine("Invalid grade");
                    break;
            }
            Console.WriteLine("Your grade is {0}", grade);
            Console.ReadLine();
        }
    }
}

```

Output: Well done
Your grade is B

Example:

```

using System;
namespace DecisionMaking
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 100;int b = 200;
            switch (a)
            {
                case 100:Console.WriteLine("This is part of outer switch ");
                    switch (b)
                    {
                        case 200:Console.WriteLine("This is part of inner switch ");
                            break;
                    }
                    break;
            }
            Console.WriteLine("Exact value of a is : {0}", a);
            Console.WriteLine("Exact value of b is : {0}", b);
            Console.ReadLine();
        }
    }
}

```

Output: This is part of outer switch
 This is part of inner switch
 Exact value of a is : 100
 Exact value of b is : 200

CONDITIONAL LOOPING: Loop control statements are used to execute a block of code several times until the given condition is true

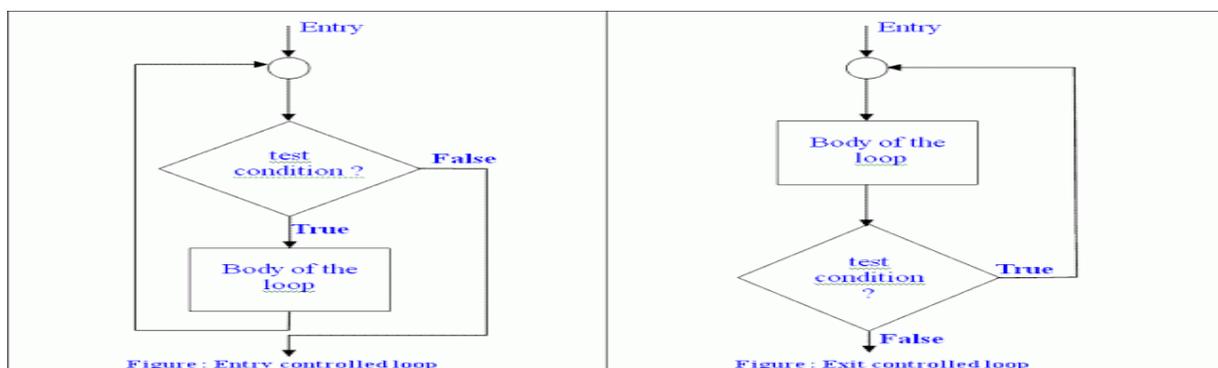
Or

The looping control statements that enable the programmer to execute a set of statements repeatedly till the required activity is completed are called looping control statements

TYPES OF LOOPING CONTROL STATEMENTS: The looping control statements are divided into two types. They are

Entry controlled loop: In such type of loop, the test condition is checked first before the loop is executed. Examples of Entry controlled loop are: [for](#) and [while](#).

Exit controlled loop: In such type of loop, the loop is executed first, then condition is checked, the loop executed at least one time even the condition if false. Examples of exit controlled loop is [do-while](#).



For Loop: Repeats a block of code multiple times until a given condition is true. Initialization, looping condition and update expression (increment/decrement) is part of for loop.

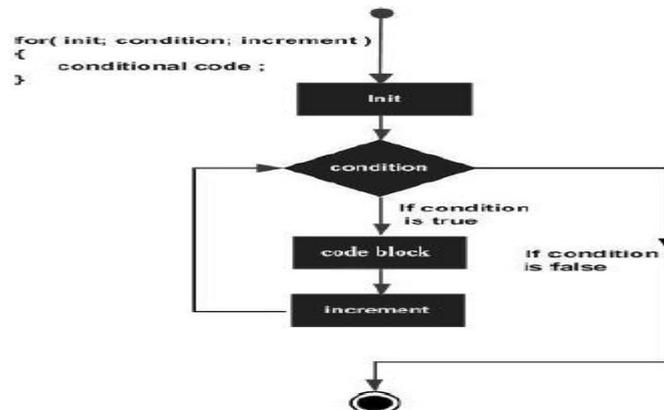
Syntax: for (initialization; condition; increment)
 {
 statement(s);
 }

Here is the flow of control in a for loop:

- The initialization is usually an assignment statement that sets the loop control variable. For Example: $i=1$ and $count=0$ Here i , $count$ are loop control variables
- Next, the condition is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop does not execute and flow of control jumps to the next statement just after the for loop.
- After the body of the for loop executes, the flow of control jumps back up to the increment

statement. This statement allows you to update any loop control variables. This statement can be left blank, as long as a semicolon appears after the condition.

- The condition is now evaluated again. If it is true, the loop executes and the process repeats itself (body of loop, then increment step, and then again testing for a condition). After the condition becomes false, the for loop terminates.



Example: using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

for (int a = 10; a < 20; a = a + 1)

Console.WriteLine("value of a: {0}", a);

Console.ReadLine();

}

}

}

Output:

value of a: 10

value of a: 11

value of a: 12

value of a: 13

value of a: 14

value of a: 15

value of a: 16

value of a: 17

value of a: 18

value of a: 19

Example: Write a program to print following output using for loop.

1

22

333

4444

55555

using System;

```

namespace Example2
{
    class Program
    {
        static void Main(string[] args)
        {
            int i,j;
            i=0;
            j=0;
            for (i = 1; i <= 5; i++)
            {
                for (j = 1; j <= i; j++)
                {
                    Console.Write(i);
                }
                Console.Write("\n");
            }
            Console.ReadLine();
        }
    }
}

```

Example:

```

using System;
namespace nested_loop
{
    class Program
    {
        static void Main(string[] args)
        {
            int i,j;
            for (i = 1; i <= 5; i++)
            {
                for (j = 1; j <= i; j++) //Nested for loop
                {
                    Console.Write(j);
                }
                Console.Write("\n");
            }
            Console.ReadLine();
        }
    }
}

```

While Loop: A while loop repeats a statement or group of statements until a given condition is true. It tests the condition before executing the loop body.

Syntax: while (condition)

```

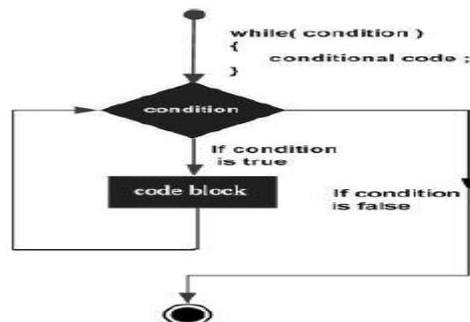
{
    statement(s);
}

```

Here is the flow of control in a while loop:

- while is keyword
- **condition:** The Test condition is evaluated and if the condition is true, then the body of the loop is executed. After execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test condition is true. The test condition must be enclosed with in parentheses.
- **Body of the loop:** The body of the loop may have one or more statements. The braces are needed only if the body contains two or more statements.

Flow Diagram



Example: using System;

```
namespace Loops
```

```
{
```

```
class Program
```

```
{
```

```
static void Main(string[] args)
```

```
{
```

```
int a = 10;
```

```
while (a < 20)
```

```
{
```

```
Console.WriteLine("value of a: {0}", a);
```

```
a++;
```

```
}
```

```
Console.ReadLine();
```

```
}
```

```
}
```

```
}
```

Output:

```
value of a: 10
```

```
value of a: 11
```

```
value of a: 12
```

```
value of a: 13
```

```
value of a: 14
```

```
value of a: 15
```

```
value of a: 16
```

```
value of a: 17
```

```
value of a: 18
```

```
value of a: 19
```

Example: C# Program to Find Magnitude of Integer

```

using System;
class Program
{
    public static void Main()
    {
        int num, mag=0;
        Console.WriteLine("Enter the Number : ");
        num = int.Parse(Console.ReadLine());
        Console.WriteLine("Number: " + num);
        while (num > 0)
        {
            mag++;
            num = num / 10;
        }
        Console.WriteLine("Magnitude: " + mag);
        Console.Read();
    }
}

```

Do...While Loop: Similar to while loop, but it tests the condition at the end of the loop body. The block of code inside do while loop will execute at least once even though the condition is false.

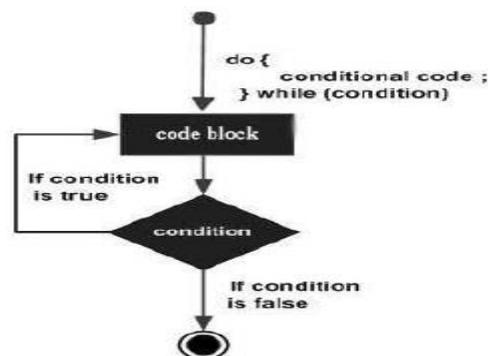
Syntax: do

```

{
    statement(s);
} while( condition );

```

- Where **do** and **while** are two keywords.
- Here the body of the loop is executed first. At the end of the loop, the condition in the while statement is evaluated. If the condition is true, the program continues to execute the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop will be terminated and the control goes to the statements that appears immediately after the while statement. Remember the body of the loop is always executed at least once.
- There is semicolon at the end of while(condition); in do-while loop

Flow Diagram

Example: using System;

```
namespace Loops
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 10;
            do
            {
                Console.WriteLine("value of a: {0}", a);
                a = a + 1;
            } while (a < 20);
            Console.ReadLine();
        }
    }
}
```

Output: value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19

Nested Loops: C# allows to use one loop inside another loop. Following section shows few examples to illustrate the concept.

Syntax: for (init; condition; increment)

```
{
    for ( init; condition; increment )
    {
        statement(s);
    }
    statement(s);
}
```

The syntax for a nested while loop statement in C# is as follows:

```
while(condition)
{
    while(condition)
    {
        statement(s);
    }
    statement(s);
}
```

The syntax for a nested do...while loop statement in C# is as follows:

```
do
```

```

{
    statement(s);
do
{
    statement(s);
}while( condition );
}while( condition );

```

A final note on loop nesting is that you can put any type of loop inside of any other type of loop. For example a for loop can be inside a while loop or vice versa.

Example: Write a program uses a nested for loop to find the prime numbers from 2 to 50

```

using System;
namespace Loops
{
    class Program
    {
        static void Main(string[] args)
        {
            int i, j;
            for (i = 2; i < 20; i++)
            {
                for (j = 2; j <= (i / j); j++)
                    if ((i % j) == 0) break;
                if (j > (i / j))
                    Console.WriteLine("{0} is prime", i);
            }
            Console.ReadLine();
        }
    }
}

```

Output:

```

2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime

```

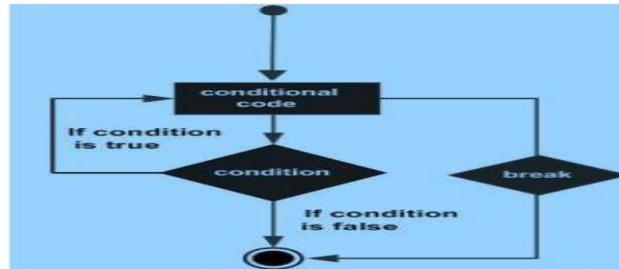
LOOP CONTROL STATEMENTS: A Loop control statement alters the normal execution path of a program. Loop control statements are used when we want to skip some statements inside loop or terminate the loop immediately when some condition becomes true.

Break Statement: When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop. It can be used to terminate a case in the switch statement. If you are using break statement in nested loops (i.e., one loop inside another loop), the break statement will stop the execution of the innermost loop and start

executing the next line of code after the block.

Syntax: **break;**

Flow Diagram:



Example: using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

int a = 10;

while (a < 20)

{

Console.WriteLine("value of a: {0}", a);

a++;

if (a > 14)

break;

}

Console.ReadLine();

}

}

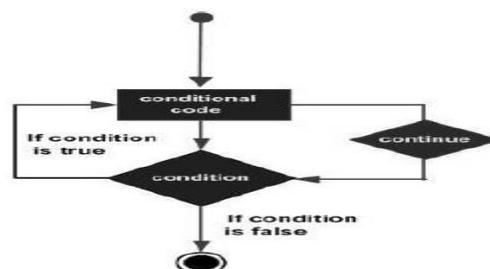
}

Output: value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14

Continue Statement: The continue statement is used to skip some part(s) of loop's body. It means We can use continue statement inside any loop (for, while and do-while). It skips the remaining statements of loop's body and starts next iteration.

Syntax: **continue;**

Flow Diagram:



Example: using System;

```

namespace Loops
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 10;
            do
            {
                if (a == 15)
                {
                    a = a + 1;
                    continue;
                }
                Console.WriteLine("value of a: {0}", a);
                a++;
            } while (a < 20);
            Console.ReadLine();
        }
    }
}

```

Output:

```

value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
value of a: 16
value of a: 17
value of a: 18
value of a: 19

```

goto:

1. C# supports the “goto” statement to jump unconditionally from one point to another in the program.
2. The goto requires a label in order to identify the place where the jump is to be made.
3. A label is any valid variable name and must be followed by a colon(:).
4. The label is placed immediately before the statement where the control is to be transferred.
5. The label can be anywhere in the program either before or after the goto label statement.
6. During running of a program, when a statement like “goto begin;”is met, the flow of control will jump to the statement immediately following the label “begin:” this happens unconditionally.
7. goto” breaks the normal sequential execution of the program.
8. If the “label:” is before the statement “goto label;” a loop will be formed and some statements will be executed repeatedly. Such a jump is known as a „backward jump“.

9. If the “label:” is placed after the “goto label;” some statements will be skipped and the jump is known as a “forward jump”.



Example: using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

int i;

double number, average, sum=0.0;

for(i=1; i<=5; ++i)

{

Console.WriteLine("Enter a number:");

number=double.Parse(Console.ReadLine());

if(number < 0.0)

goto jump;

sum += number; // sum = sum+number;

}

jump:

average=sum/(i-1);

Console.WriteLine ("Sum value is" +sum);

Console.WriteLine ("Average is" +average);

}

}

}

Infinite Loop: A loop becomes infinite loop if a condition never becomes false. We can make an endless (or) infinite loop by leaving the conditional expression empty.

Example: using System;

namespace Loops

{

class Program

{

static void Main(string[] args)

{

for (; ;)

Console.WriteLine ("Hey! I am Trapped");

}

}

}

ARRAYS: An array is a set of similar type values that are stored in sequential order.

Or

Array is a group of related data items that shares a common name.

Or

Array is collection of homogeneous data elements

In C# the array index starts at zero. That means the first item of an array starts at 0th position. The position of the last item of an array will totalnumber of items-1.so if an array has 10 items, the last 10th item is at 9th position.In C#, arrays can be declared as **fixed length** or **dynamic**.

Fixed length arrays: A fixed length array can store a predefined number of items.

Dynamic arrays: A dynamic array does not have a predefined size. The size of a dynamic array increases as you add new items to the array.

- To access an element we can use array name and index value of the concern (or) particular element.
- In c# arrays are reference types. Hence it stores default values based on data type.

Types of Arrays: C# .NET supports three types of arrays.

One-dimensional array: Arranging collection of elements in a single row can be called **One-dimensional array**.

Or

A list of items can be given under single variable name using only one subscript is called single subscripted variable (or) One-dimensional array.

All items in a single dimension array are stored contiguously starting from 0 to the size of the array-1.

Declaration: Like other variables arrays must be declared before they are used. To declare an array in C#, you can use the following syntax:

Syntax: `datatype[] arrayName=new data type[size];`

or

`data type[] arrayname;`
`arrayname=new datatype[size];`

or

`int[] arrayname={list of values};`

Where,

- data type- It is used to specify the type of elements in the array.
- []- specifies subscriptal operator.
- Size-specifies the size of the array.
- array name-specifies the name of the array.
- new-memory allocation operator.

Example: `int[] a=new int[10];`
 Or
`double[] balance;`
`balance=new double[5];`

Initializing an Array: Once an array is declared, the next step is to initialize an array. The initialization process of an array includes adding actual data to the array. Array is a reference type, so you need to use the new keyword to create an instance of the array.

Example: You can assign values to the array at the time of declaration as shown:

```
double[] balance = { 2340.0, 4523.69, 3421.0};
```

Example: You can also create and initialize an array as shown:

```
int [] marks = new int[5]{ 99,98, 92, 97, 95};
```

Example: You may also omit the size of the array as shown:

```
int [] marks = new int[] { 99,98, 92, 97, 95};
```

You can copy an array variable into another target array variable. In such case, both the target and source point to the same memory location (Both the array will have same values).

```
int [] marks = new int[] { 99,98, 92, 97, 95};
int[] score = marks;
```

When you create an array, C# compiler implicitly initializes each array element to a default value depending on the array type. For example, for an int array all elements are initialized to 0.

Example: using System;

```
namespace ArrayApplication
{
    class MyArray
    {
        static void Main(string[] args)
        {
            int [] n = new int[10];
            int i,j;
            for ( i = 0; i < 10; i++ )
                n[ i ] = i + 100;
            for (j = 0; j < 10; j++ )
                Console.WriteLine("Element[{0}] = {1}", j, n[j]);
            Console.ReadKey();
        }
    }
}
```

Output: 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

```

Example: using System;

```

namespace One_Dimensional_Array
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] Books = new string[5];
            Books[0] = "C#";
            Books[1] = "Java";
            Books[2] = "VB.NET";
            Books[3] = "C++";
            Books[4] = "C";
            Console.WriteLine("Element of Books array is:\n\n");
            int i = 0;
            Console.Write("\t1\t2\t3\t4\t5\n\n\t");
            for (i = 0; i < 5; i++)
                Console.Write("{0}\t", Books[i]);
            Console.ReadLine();
        }
    }
}

```

foreach Loop: It is specially designed for accessing the value of an array (or) collection(list, hash-table, stack, queue and linked list) where for each iteration of the loop one value of the array is assigned to the loop variable and the return to us.

Syntax: foreach(type variable_name in expression)

```

{
    Body of the loop;
}

```

Where,

- **type**-specifies the data type
- **variable_name**-specifies the array variable name.
- **in**- it is a keyword.
- **expression** - specifies array type or collection type.

Example: using System;
namespace ArrayApplication

```

{
    class MyArray
    {
        static void Main(string[] args)
        {
            int [] n = new int[10];
            for ( int i = 0; i < 10; i++ )
                n[i] = i + 100;
        }
    }
}

```

```

        foreach (int j in n )
        {
            int i = j-100;
            Console.WriteLine("Element[{0}] = {1}", i, j);
            i++;
        }
        Console.ReadKey();
    }
}

```

Output:

```

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

```

Example: using System;

```

namespace store_value_in_array

```

```

{
    class Program
    {
        static void Main(string[] args)
        {
            int i;
            int[] arr = new int[5]; // 5 size array
            for (i = 0; i < 5; i++)
            {
                Console.Write("\nEnter your number:\t");
                arr[i] = Convert.ToInt32(Console.ReadLine());
            }
            Console.WriteLine("\n\n");
            for (i = 0; i < 5; i++)
            {
                Console.WriteLine("you entered {0}", arr[i]);
            }
            Console.ReadLine();
        }
    }
}

```

Example: using System;

```

namespace accessing_array_value

```

```

{
    class Program
    {
        static void Main(string[] args)

```

```

    {
        int[] age=new int[6];
        string[] name = new string[6];
        int i,j=0;
        string find;
        for (i = 0; i < 6; i++)
        {
            Console.WriteLine("\n\nEnter your name:\t");
            name[i] = Console.ReadLine();
            Console.WriteLine("Enter your age:\t\t");
            age[i] = Convert.ToInt32(Console.ReadLine());
        }
        Console.WriteLine("\n\nEnter your name to find age:\t");
        find = Console.ReadLine();
        for (i = 0; i < 6; i++)
        {
            if (name[i] == find)
            {
                Console.WriteLine("\n\nName\t: {0} ", name[i]);
                Console.WriteLine("Age\t: {0} ", age[i]);
                j++;
            }
        }
        if (j == 0)
        {
            Console.WriteLine("Not Found!!!");
        }
        Console.ReadLine();
    }
}
}

```

Example: using System;
namespace forgetCode
{

```

    class Program
    {
        public static void Main()
        {
            int n;
            float large, small;
            int[] a = new int[50];
            Console.WriteLine("Enter the size of Array");
            string s = Console.ReadLine();
            n = Int32.Parse(s);
            Console.WriteLine("Enter the array elements");
            for (int i = 0; i < n; i++)
            {
                string s1 = Console.ReadLine();
                a[i] = Int32.Parse(s1);
            }
        }
    }
}

```


You can also omit the new operator as we did in single dimension arrays. You can assign these values directly without using the new operator. For example:

```
int[,] numbers = { { 1, 2 }, { 3, 4 }, { 5, 6 } };
string[,] names = { { "Rosy", "Amy" }, { "Peter", "Albert" } };
```

We can also initialize the array items one item at a time. The following code is an example of initializing array items one at a time.

```
int[,] numbers = new int[3, 2];
    numbers[0, 0] = 1;
    numbers[1, 0] = 2;
    numbers[2, 0] = 3;
    numbers[0, 1] = 4;
    numbers[1, 1] = 5;
    numbers[2, 1] = 6;
```

Example: using System;

```
namespace ArrayApplication
{
    class MyArray
    {
        static void Main(string[] args)
        {
            int[,] a = new int[5, 2] {{0,0}, {1,2}, {2,4}, {3,6}, {4,8} };
            int i, j;
            for (i = 0; i < 5; i++)
            {
                for (j = 0; j < 2; j++)
                    Console.WriteLine("a[{0},{1}] = {2}", i, j, a[i,j]);
            }
            Console.ReadKey();
        }
    }
}
```

Output:

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

Jagged Array (or) arrays of arrays (or) dynamic arrays: Jagged array is collection rows which contain distinct number of elements in each row that means all the rows may not contain same number of elements.

Or

Jagged array is collection of one-dimensional arrays of varying size.

Declaration: Declaration of a jagged array involves two brackets.

```
type[][] array_name = new type[3][];
```

or

```
type[][] array_name={list of values};
```

Initialization: Before a jagged array can be used, its items must be initialized. The following code snippet initializes a jagged array; the first item with an array of integers that has two integers, second item with an array of integers that has 4 integers, and a third item with an array of integers that has 6 integers.

```
type[][] array_name = new type[3][];
intJaggedArray[0] = new int[2]{2, 12};
intJaggedArray[1] = new int[4]{4, 14, 24, 34};
intJaggedArray[2] = new int[6]{6, 16, 26, 36, 46, 56};
```

Example: example illustrates using a jagged array:

```
using System;
namespace ArrayApplication
{
    class MyArray
    {
        static void Main(string[] args)
        {
            /* a jagged array of 5 array of integers*/
            int[][] a = new int[][]{new int[]{0,0},new int[]{1,2},
            new int[]{2,4},new int[]{ 3, 6 }, new int[]{ 4, 8 } };
            int i, j;
            /* output each array element's value */
            for (i = 0; i < 5; i++)
            {
                for (j = 0; j <= 2; j++)
                {
                    Console.WriteLine("a[{0}][{1}] = {2}",i, j, a[i][j]);
                }
            }
            Console.ReadKey();
        }
    }
}
```

Output:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

Passing Arrays as Function Arguments: You can pass an array as a function argument in C#. For this we follow the rules.

1. The function must be called by passing only the name of the array and the size of the array.

Syntax: **max(a,n);**

2. In the function definition, the formal parameter must be an array type; the size of the array does not need to be specified.

3. The function header might look like:

Syntax: **float max(float[] array , int size)**

```
using System;
namespace ArrayApplication
{
    class MyArray
    {
        double getAverage(int[] arr, int size)
        {
            int i;double avg;int sum = 0;
            for (i = 0; i < size; ++i)
                sum += arr[i];
            avg = (double)sum / size;
            return avg;
        }
        static void Main(string[] args)
        {
            MyArray app = new MyArray();
            int [] balance = new int[]{1000, 2, 3, 17, 50};
            double avg;
            /* pass pointer to the array as an argument */
            avg = app.getAverage(balance, 5 );
            /* output the returned value */
            Console.WriteLine( "Average value is: {0} ", avg );
            Console.ReadKey();
        }
    }
}
```

Output: Average value is: 214.4

Variable argument list (or) Param Arrays: At times, while declaring a method, you are not sure of the number of arguments passed as a parameter. C# param arrays (or parameter arrays) come into help at such times. In C# we can define methods that can handle variable number of arguments is called parameter arrays. Parameter arrays are declared using the keyword **params**.

Example: using System;

```
namespace ArrayApplication
{
    class ParamArray
    {
        public int AddElements(params int[] arr)
        {
            int sum = 0;
```

```

        foreach (int i in arr)
            sum += i;
        return sum;
    }
}
class TestClass
{
    static void Main(string[] args)
    {
        ParamArray app = new ParamArray();
        int sum = app.AddElements(512, 720, 250, 567, 889);
        Console.WriteLine("The sum is: {0}", sum);
        Console.ReadKey();
    }
}
}

```

Output: The sum is: 2938

Array Class: The Array class is the base class for all the arrays in C#. Array Class is defined in the System namespace. In C# every array is automatically derived from the System.Array class. The Array class provides various properties and methods to work with arrays.

Properties of the Array Class: The following table describes some of the most commonly used properties of the Array class:

S.No	Property name	Description
1	Length	Gets a 32-bit integer that represents the total number of elements in all the dimensions of the array.
2	Rank	Gets the rank (number of dimensions) of the array.
3	IsFixedSize	Gets a value indicating whether the array has fixed size.
4	IsReadOnly	Gets a value indicating whether the array is read only
5	LongLength	Gets a 64-bit integer that represents the total number of elements in all the dimensions of the array.

Methods of the Array Class: The following table describes some of the most commonly used methods of the Array

S.No	Property name	Description
1	Clear	Sets the range of elements in the array to zero, to false,(or) to null, depending on the element type.
2	Copy()	Copies the range of elements from an array starting at the first element and pastes them into another array
3	CopyTo()	Copies all the elements of the current one dimensional array to the specified one dimensional array starting at the specified destination array index.
4	GetLenth	Gets a 32-bit integer that represents the number of elements in the specified dimension of the array.
5	GetLongLength	Gets a 64-bit integer that represents the number of elements in the specified dimension of the array.
6	GetLowerBound	Get the lower bound of the specified dimension in the array

7	GetUpperBound	Get the Upper bound of the specified dimension in the array
8	GetType	Gets type of the current instance.
9	GetValue()	Gets the value at the specified position in the one dimensional array
10	IndexOf()	Searches for the specified object and returns the index of the first occurrence within the entire one dimensional array.
11	Reverse()	Reverses the sequence of the elements in the entire one dimensional array.
12	SetValue()	Sets a value to the element at the specified position in the one dimensional array.
13	Sort()	Sorts the elements in an entire one dimensional array.
14	ToStringK()	Returns a string that represents the current object

Example: The following program demonstrates use of some of the methods of the Array class:

```
using System;
namespace ArrayApplication
{
    class MyArray
    {
        static void Main(string[] args)
        {
            int[] list = { 34, 72, 13, 44, 25, 30, 10 };
            int[] temp = list;
            Console.WriteLine("Original Array: ");
            foreach (int i in list)
                Console.WriteLine(i + " ");
            Console.WriteLine();
            // reverse the array
            Array.Reverse(temp);
            Console.WriteLine("Reversed Array: ");
            foreach (int i in temp)
                Console.WriteLine(i + " ");
            Console.WriteLine();
            //sort the array
            Array.Sort(list);
            Console.WriteLine("Sorted Array: ");
            foreach (int i in list)
                Console.WriteLine(i + " ");
            Console.WriteLine();
            Console.ReadKey();
        }
    }
}
```

Output: Original Array: 34 72 13 44 25 30 10
 Reversed Array: 10 30 25 44 13 72 34
 Sorted Array: 10 13 25 30 34 44 72

Example: Programming Examples of array class
 using System;

```

namespace Array_Class
{
    class Program
    {
        static void printarray(int[] arr)
        {
            Console.WriteLine("\nElements of array is:\n");
            foreach (int i in arr)
            {
                Console.Write("\t{0}", i);
            }
            Console.WriteLine("\n");
        }
        static void Main(string[] args)
        {
            int[] arr1=new int[5]{43,25,33,14,5};
            int[] arr2 = new int[5];
            int len,rank;
            bool fixedsize, read_only;
            len = arr1.Length;
            Console.WriteLine("Length:\t{0}", len);
            rank = arr1.Rank;
            Console.WriteLine("Rank:\t{0}", rank);
            fixedsize = arr1.IsFixedSize;
            Console.WriteLine("Fixed Size:\t{0}", fixedsize);
            read_only = arr1.IsReadOnly;
            Console.WriteLine("Read Only:\t{0}", read_only);
            Array.Sort(arr1);
            printarray(arr1);
            Console.WriteLine("Get Length:\t{0}",arr1.GetLength(0));
            Console.WriteLine("Get Value:\t{0}", arr1.GetValue(2));
            Console.WriteLine("Get Index:\t{0}",Array.IndexOf(arr1, 33));
            Array.Copy(arr1, arr2,5);
            printarray(arr2);
            Array.Clear(arr1, 0, 5);
            printarray(arr1);
            Console.ReadLine();
        }
    }
}

```

Example:

```

using System;
namespace Array_Class
{
    class Program
    {
        static void printarray(int[] arr)
        {
            Console.WriteLine("Single Dimension Array Sample");

```

```

string[] strArray = new string[] { "Mahesh Chand", "Mike
Gold", "Raj Beniwal", "Praveen Kumar", "Dinesh Beniwal" };
foreach (string str in strArray)
{
    Console.WriteLine(str);
}
Console.WriteLine("-----");
Console.WriteLine("Multi-Dimension Array Sample");
string[,] string2DArray = new string[2, 2]
{{ "Rosy", "Amy" }, { "Peter", "Albert" } };
foreach (string str in string2DArray)
{
    Console.WriteLine(str);
}
Console.WriteLine("-----");
Console.WriteLine("Jagged Array Sample");
int[][] intJaggedArray3 = { new int[] { 2, 12 }, new int[] { 14, 14,
24, 34 }, new int[] { 6, 16, 26, 36, 46, 56 } };
for (int i = 0; i < intJaggedArray3.Length; i++)
{
    Console.WriteLine("Element({0}): ", i);
    for (int j = 0; j < intJaggedArray3[i].Length; j++)
    {
        Console.WriteLine("{0}{1}", intJaggedArray3[i][j],
            j==(intJaggedArray3[i].Length-1)?"":" ");
    }
    Console.WriteLine();
}
Console.WriteLine("-----");
}
}
}

```

Example: Write a program of sorting an array. Declare single dimensional array and accept 5 integer values from the user. Then sort the input in ascending order and display output.

```

using System;
namespace Example1
{
    class Program
    {
        static void printarray(int[] arr)
        {
            Console.WriteLine("\n\nElements of array are:\n");
            foreach (int i in arr)
            {
                Console.WriteLine("\t{0}", i);
            }
        }
        static void Main(string[] args)
        {

```

```

        int[] arr = new int[5];
        int i;
        for (i = 0; i < 5; i++)
        {
            Console.WriteLine("Enter number:\t");
            arr[i] = Convert.ToInt32(Console.ReadLine());
        }
        Program.printarray(arr);
        Array.Sort(arr); //use array's sort function
        Program.printarray(arr);
        Console.ReadLine();
    }
}
}

```

Differences between for and foreach loop:

S.No	for	foreach
1	Loop variable refers to the index of an array.	Loop variable refers to the values of an array
2	Loop variable is integer type	The type of the loop variable is similar to the type of values inside an array
3	It is used for both accessing and assigning values to an array	It is used only for accessing.

STRINGS: String represents sequence of characters. C# supports two types of strings.

- **Mutable strings(dynamic strings)**
- **Immutable strings**

Immutable strings: String objects are immutable, meaning that we cannot modify the characters contained in them. String is alias for the predefined **System.String** class in the CLR, there are many built in operations available that work with strings.

C# also supports regular expressions that can be used for complex string manipulations and pattern matching.

Creating a String Object: C# supports a predefined reference type known as string. We can use string to declare string type objects. You can create string object using one of the following methods:

- By assigning a string literal to a String variable
- Copying from one object to another.
- Concatenating two objects.
- Reading from the keyword.
- Using **ToString** method.

By assigning a string literal to a String variable: The most common way to create a string is to assign a quoted string of characters known as string.

Example: string S1;

```
S1="abc"
(or)
string S1="abc";
```

COPYING STRINGS: we can also create new copies of existing strings. This can be done in two ways.

- Using the overloaded operator (=)
- Using the static copy method.

Example: `string s2=s1;`
`string s2=string.Copy(s1);`

both these statements would accomplish the same thing, namely, copying the contents of s1 into s2.

Concatenating strings: We may also create new strings by Concatenating existing strings. This can be done in two ways.

- Using the overloaded operator (+)
- Using the static Concat method

Example: `string s3=s1+s2;`
`string s3=string.Concat(s1,s2);`

If s1="abc" and s2="xyz", then both the statements will store the string "abcxyz" in s3.

Reading from the keyword: It is possible to read a string value interactively from the keyboard and assign it to a string object.

```
string s=Console.ReadLine();
```

On reaching this statement, the computer will wait for a string of characters to be entered from the keyboard. When the return key is pressed, the string will be read and assigned to the string object s.

Using ToString method: Another way of creating string is to call the ToString method on an object and assign the result to a string variable.

```
int number=123;
string numStr=number.ToString();
```

The above statement convert the number123 to a string "123" and then assigns the string value to the string variable numStr.

Verbatim Strings: String can also be created using verbatim strings. Verbatim strings are those that start with the @ symbol. This symbol tells the compiler that the string should be used verbatim even if it includes escape characters.

```
string s1= @"\\EBG\Csharp\string.cs";
```

In order to obtain the same output without using the symbol @, The input string should be written as follows.

```
string s1= "\\EBG\\Csharp\\string.cs";
```

we may also use escape characters such as \n and \t in @-quoted strings. They will not be processed during output.

Note: If ordinary string contains an embedded \n, the string that follows \n will be displayed on the next line when the string is processed for output.

Methods for immutable strings:

S.No	Method	Operation
1	Compare()	Compares two strings
2	CompareTo()	Compared the current instance with another instance
3	ConCat()	Concatenates two or more strings
4	Copy()	Creates a new string by copying another
5	CopyTo()	Copies a specified number of characters to an array of Unicode characters
6	EndsWith()	Determines whether the substring exists at the end of the string
7	Equals()	Determines if two strings are equal
8	IndexOf()	Returns position of first occurrence of a substring
9	Insert()	Inserts a string at a specified position.
10	Join()	Joins an array of strings together.
11	LastIndexOf()	Returns the position of the last occurrence of a substring.
12	PadLeft()	Left-aligns the string in a field.
13	PadRight()	Right-aligns the string in a field.
14	Remove()	Deletes characters from the string.
15	Replace()	Replaces all instances of a character with a new character.
16	Split()	Creates an array of strings by splitting the string at any occurrence of one.
17	StartsWith()	Determines whether the substring exists at the beginning of the string
18	SubString()	Extracts a substring.
19	ToLower()	Returns a lower case version of the string
20	ToUpper()	Returns a upper case version of the string
21	Trim()	Removes white space from the string.
22	TrimEnd()	Removes a string of characters from the end of the string.
23	TrimStart()	Removes a string of characters from the beginning of the string

Inserting Strings: We can insert the string at specified position using the Insert() method, which is available in **System.String** class.

```

Example:    using System;
                class Stringmethod
                {
                    public static void Main()
                    {
                        string s1="lean";
                        string s2=s1.Insert(3,"r");
                        string s3=s2.Insert(5,"er");
                        for(int i=0;i<s3.Length;i++)
                        {
                            Console.Write(s3[i]);
                        }
                    }
                }

```

In the above program ,when the statement

```
string s2=s1.Insert(3,"r");
```

is executed, the string variable s2 contains the string "Learn". The string "r" is inserted in s1 after 3 characters. Similarly, the string "er" is inserted at the end of the string. Finally the variable s3 contains the value "Learner".

Example: Take two inputs as string from the users and copies the input of string2 to string5 and checks for the string3 ends with IDE or not. The program shows true if string3 ends with IDE. Searches char 'a' from the string1. Insert hello in string6 at position 6 and shows the atring6.

```
using System;
namespace SearchString
{
    class Program
    {
        public void Display()
        {
            string str1="";
            Console.Write("Enter a string");
            str1=Console.ReadLine();
            string str2="";
            Console.Write("Enter another string");
            str2=Console.ReadLine();
            string str3="C# 2005 is developed in Visual Studio 2005IDE";
            Console.Write("string str3 is {0}",str3);
            string str5=string.Copy(str2);
            Console.Write(" string str5 is copied from str2: {0}",str5);
            Console.Write(" string str5 is {0} characters long:",str5.Length);
            Console.Write(" the 10th character of string str3 is : {0}",str3[9]);
            Console.Write(" string str3 {0} \n ends with
            IDE?: {1}\n",str3,str3.EndsWith("IDE"));
            Console.Write("Ends with studio?: {0}",str3.EndsWith("studio"));
            Console.Write("\n the first time character 'a' occurred in string str1 at
            position: {0}",str1.IndexOf("a")+1);
            String str6=str2.Insert(6,"hello");
            Console.Write(" 'hello' is inserted in string str6.string str6 is
            noe: {0}",str6);
        }
        static void Main()
        {
            Program prg=new Program();
            Prg.Display();
        }
    }
}
```

Comparing Strings: String class supports overloaded methods and operators to compare whether two strings are equal or not. They are

- Overloaded Compare() Method
- Overloaded Equals() Method
- Overloaded == operator

Compare() method: There are two versions of overloaded static Compare() method. The first one

takes two strings as parameters and compares them.

Example: `int n=string.Compare(s1,s2);`

This performs case-sensitive comparison and returns different integer values for different conditions as under:

Zero , if s1 is equal to s2.
 A positive integer(1), if s1>s2.
 A negative integer(-1), if s1<s2.

Example: `s1="abc"`
`s2="ABC";`
`int n=string.Compare(s1,s2);`

Output: n value is -1.

The second version of Compare () method takes an additional bool type parameter to decide whether case should be ignored or not. If the bool parameter is true, case is ignored.

Example: `int n = string.Compare(s1,s2,true);`

Example: `s1="abc"`
`s2="ABC";`
`int n=string.Compare(s1,s2,true);`

output: n value is 0.

Example: `using System;`
`namespace StringApplication`
`{`
`class StringProg`
`{`
`static void Main(string[] args)`
`{`
`string str1 = "This is test";`
`string str2 = "This is text";`
`if (String.Compare(str1, str2) == 0)`
`Console.WriteLine(str1 + " and " + str2 + " are equal.");`
`else`
`Console.WriteLine(str1+"and"+str2 + " are not equal.");`
`Console.ReadKey() ;`
`}`
`}`
`}`

Output: This is test and this is text are not equal.

Equals() method: The string class supports an overloaded Equals method for testing the equality of strings. There are again two versions of Equals method.

If, `s1="abc";`
`s2="ABC";`
`bool b1=s2.Equals(s1);`
`bool b1=string.Equals(s2,s1);`

These methods return a Boolean value true if s1 and s2 are equal, otherwise false.

The == operator: A simple and natural way of testing the equality of strings is by using overloaded

== operator.

Example: bool b3= (s1==s2); //b3 is true if they are equal.
 Or
 If(s1==s2)
 Console.Write(“equal”);

Finding substrings: It is possible to extract substrings from a given string using the overloaded Substring method available in String class. There are two versions of Substring:

s.Substring(n): It extract substrings from the nth position to the last character of the string contained in s.

s.Substring(n1,n2):It extracts a substring from s beginning at n1 position and ending at n2 position.

Examples: string s1=“NEW YORK”;
 string s2=s1.Substring(5);
 string s3=s1.Substring(0,3);
 string s4=s1.Substring(5,8);

When the above statements are executed, the string variables will contain the following substrings

s2:YORK
s3:NEW
s4:YORK

Mutable strings: string objects are mutable, meaning strings are modifiable. Mutable strings are created using **StringBuilder** class.

Example: **StringBuilder str1=new StringBuilder(“abc”);//with initial size of three characters**

StringBuilder str1=new StringBuilder(); //empty string

The string object str1 is created with an initial size of three characters and str2 is created as an empty string. They can grow dynamically as more characters are added to them.

The **StringBuilder** class supports many methods that are useful for manipulating dynamic strings.

StringBuilder mehods:

S.No	Method	Operation
1	Append()	Appends a string
2	AppendFormat()	Appends string using specific format
3	EnsureCapacity()	Ensure sufficient size
4	Insert()	Insert a string at a specified position
5	Remove()	Removes the specified characters
6	Replace()	Replaces all instances of a character with a specified one.

C# also supports some special functions known as properties.

StringBuilder properties:

S.No	Method	Operation
1	Capacity	To retrieve or set the number of characters the object can hold
2	Length	To retrieve or set the length
3	MaxCapacity	To retrieve the maximum capacity of the object
4	[]	To get or set a character at a specified position.

The System.Text namespace contained the StringBuilder class and therefore we must include the

using System.Text directive for creating and manipulating mutable strings.

Example:

```
using System.Text; using System;
class StringBuilderMethod
{
    public static void Main()
    {
        StringBuilder s=new StringBuilder("object");
        Console.WriteLine("original string:"+s);
        Console.WriteLine("length:"+s.Length);
        s.Append("language");
        Console.WriteLine("String now:"+s);
        s.Insert(7,"oriented");
        Console.WriteLine("modified string:"+s);
        int n=s.Length;
        s[n-1]='!';
        Console.WriteLine("final string:"+s);
    }
}
```

Output: original string: object
 Length : 7
 String now: object language
 Modified string: object oriented language
 Final string: object oriented language!

Example: program to accept two string inputs from users and appends the first string to a predefined value. Using StringBuilder class, a string value is inserted in the string. Then, all spaces in first are replaced with * and the program calculates the length of two strings after appending.

Arrays of strings: The statement

```
string[] itemarray=new string[3];
```

the above will create itemarray of size 3 to hold three strings. we can assign the strings to the itemarray element by element using 3 different statements (or) using for loop.

```
string[] itemarray={"a","b","c"};
```

Example:

```
using System;
class Strings
{
    public static void Main()
    {
        string[] countries={"india","Germany","America","France"};
        int n=countries.Length;
        Array.Sort(countries);
        for(int i=0;i<n;i++)
            Console.WriteLine(countries[i]);
        Array.Reverse(countries);
        for(int i=0;i<n;i++)
            Console.WriteLine(countries[i]);
    }
}
```

Regular expressions: Regular expressions provide a powerful tool for searching and manipulating a

large text. A regular expression may be applied to a text to accomplish tasks such as.

- To locate substrings and return them.
- To modify one (or) more substrings and return them.
- To identify substrings that begins with (or) ends with a pattern of characters.
- To find all words that begins with a group of characters and end with some other characters.
- To find all occurrences of a substring.

A regular expression is a string containing two types of characters.

Literals: These are characters that we wish to search and match in the text.

Metacharacters: These are special characters that give commands to the regular expression parser.

Examples:

S.No	Expression	Meaning
1	"\bm"	Any word beginning with m
2	"er\b"	Any word ending with er
3	"\BX\B"	Any X in the middle of the word
4	"\bm\S*er\b"	Any word beginning with m and ending with er.
5	" ,""	Any word separated by a space or a comma

In the above table ,\b,\B,\S* and | are metacharacters and m, er, X and comma are literals.

The .NET framework provides support for regular expression matching and replacement. The namespace System.Text.RegularExpressions support a number of classes that can be used for searching, matching and modifying a text document. The important classes are

- Regex
- MatchCollection
- Match

Example:

```
using System;
using System.Text;
using System.Text.RegularExpressions;
class RegexTest
{
    public static void Main()
    {
        string str;
        str="amar,akbar,antony are friends";
        Regex reg=new Regex(",");
        stringBuilder sb=new StringBuilder();
        int count=1;
        foreach(string sub in reg.Split(str))
            sb.AppendFormat("{0}:{1}\n",count++,sub);
        Console.WriteLine(sb);
    }
}
```

Output:
1:amar
2:akbar

3:antony
4:are
5:friends

STRUCTURES: Structure is collection of data items of different type.

Or

A structure is a collection of heterogeneous data elements

- Structures are similar to classes in C#.
- In C# structure is a value type data type.

Defining a Structure: To define a structure, you must use the struct statement.

Syntax: struct struct_name
 {
 Data member1;
 Data member2;
 Data member3;
 }
 Or
 struct struct_name
 {
 Data member1;
 Data member2;
 Data member3;
 };

Example: struct Books
 {
 public string title;
 public string author;
 public string subject;
 }

In the above code the keyword struct declares Books as a new data type that can hold three variables of different data types. These variables are known as members or elements.

Declaring structure variables: we can declare structure variables using the structure-tag in the program. A structure variable declaration is similar to the declaration of variables of any other data types. It includes the following elements.

- The structure tag name
- List of variable names separated by commas
- A terminating semicolon

Syntax: structure_tag structure_var1, structure_var2,.....structure_varN;

Example: Books book1,book2,book3;

Each one of the variables has three members as specified by the template.

Assigning values to structure members: Structure members can be accessed using the simple dot notation.

Syntax: `structurevariablename.structuremember= value;`

Example: `book1.title="c#";`
`book1.author="raja";`
`book1.subject="c sharp";`

Copying Structure Variables Two variables of the same structure type can be copied the same way as ordinary variables. If person1 and person2 belongs to the same structure, then the following assignment operations are valid:

`person1 = person2;` ----- assigns person1 to person2

`person2 = person1;` ----- assigns person2 to person1

Note:

1. we can also use the operator **new** to create stricture variables

Syntax: `struct-tag structure-variable=new struct-tag();`

Example: `books b1=new books();`

2. Structure members are by default private and therefore cannot be accessed outside the structure definition.

Example: The following program shows the use of the structure:

```
using System;
struct Books
{
    public string title;
    public string author;
    public string subject;
    public int book_id;
};
public class testStructure
{
    public static void Main(string[] args)
    {
        Books Book1;
        Books Book2;
        Book1.title = "C Programming";
        Book1.author = "Nuha Ali";
        Book1.subject = "C Programming Tutorial";
        Book1.book_id = 6495407;
        Book2.title = "Telecom Billing";
        Book2.author = "Zara Ali";
        Book2.subject ="Telecom Billing Tutorial";
        Book2.book_id = 6495700;
        Console.WriteLine("Book 1 title : {0}", Book1.title);
        Console.WriteLine("Book 1 author : {0}", Book1.author);
        Console.WriteLine("Book 1 subject : {0}", Book1.subject);
        Console.WriteLine("Book 1 book_id :{0}", Book1.book_id);
        Console.WriteLine("Book 2 title : {0}", Book2.title);
        Console.WriteLine("Book 2 author : {0}", Book2.author);
        Console.WriteLine("Book 2 subject : {0}", Book2.subject);
        Console.WriteLine("Book 2 book_id : {0}", Book2.book_id);
        Console.ReadKey();
    }
}
```

```

    }
}

```

Output:

```

Book 1 title : C Programming
Book 1 author : Nuha Ali
Book 1 subject : C Programming Tutorial
Book 1 book_id : 6495407
Book 2 title : Telecom Billing
Book 2 author : Zara Ali
Book 2 subject : Telecom Billing Tutorial
Book 2 book_id : 6495700

```

Example: The following program shows the use of the structure:

```

using System;
namespace Structure
{
    class Program
    {
        struct book
        {
            public string bookname;
            public int price;
            public string category;
        }
        static void Main(string[] args)
        {
            book language, database;
            Console.Write("Enter book name:\t");
            language.bookname = Console.ReadLine();
            Console.Write("Enter book price:\t");
            language.price=Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter book category:\t");
            language.category = Console.ReadLine();
            Console.Write("\n\nEnter book name:\t");
            database.bookname = Console.ReadLine();
            Console.Write("Enter book price:\t");
            database.price=Convert.ToInt32(Console.ReadLine());
            Console.Write("Enter book category:\t");
            database.category = Console.ReadLine();
            Console.Write("\n\n=====");
            Console.Write("\n\t\tLanguage\n");
            Console.Write("=====\n\n");
            Console.Write("BookName:\t{0}",language.bookname);
            Console.Write("\nBook Price:\t{0}", language.price);
            Console.Write("\nBook Category:\t{0}", language.category);
            Console.Write("\n\n=====");
            Console.Write("\t\tDatabase\n");
            Console.Write("=====\n\n");
            Console.Write("BookName:\t{0}",database.bookname);
            Console.Write("\nBookPrice:\t{0}",database.price);
            Console.Write("\nBookCategory:\t{0}",database.category);

```

```

        Console.ReadLine();
    }
}

```

Output:

```

Enter book name      :    C Sharp
Enter book price     :    34
Enter book category  :    Object Oriented Programming
Enter book name      :    SQL Server
Enter book price     :    23
Enter book category:  Database Programming
=====

```

Language

=====

```

Enter book name      :    C Sharp
Enter book price     :    34
Enter book category:  Object Oriented Programming
=====

```

Database

=====

```

Enter book name :    SQL Server
Enter book price :    23
Enter book category :  Database Programming

```

Structures with methods: we have seen that values may be assigned to the data members using structure_variable and the dot operator. We can also assign values to the data members using what are known as constructors.

A constructor is a method which is used to set values of data members at the time of declaration.

Example:

```

struct Number
{
    int number; // data member
    public Number(int value) // constructor
    {
        number=value;
    }
}

```

The constructor method has the same name as structure-tag and declared as public.the constructor is invoked as follows.

Number n1=new Number(100);

The above statement creates a structure object n1 and assigns the value of 100 to its only data member number. C# does not support default constructor.

Example:

```

using System;
struct Rectangle
{
    int a,b;
}

```

```

        public Rectangle(int x, int y);
        {
            a=x;
            b=y;
        }
        public int Area()
        {
            return (a*b);
        }
        public void display()
        {
            Console.WriteLine("Area="+Area());
        }
    }
class TestRectangle
{
    public static void Main()
    {
        Rectangle rect=new Rectangle(10,20);
        rect.display();
    }
}

```

Nested structures: Structure inside the structure is called nested structure.

Syntax:

```

struct employee
{
    public string name;
    public int code;
    public struct salary
    {
        public double basic;
        public double allowance;
    }
}

```

We can also use Structure variables as members of another Structure.

```

struct M
{
    public int x;
}
struct N
{
    public int y;
    public M m;
};

```

Class versus Structure: Classes and Structures have the following basic differences:

S.No	class	Structure
1	classes are reference types	struts are value types
2	classes are stored on heap	struts are stored on stack
3	Classes support inheritance	Structures do not support inheritance
4	Permit initialization of instance fields	Do not Permit initialization of instance fields

5	Classes supports default constructor	structures do not support default constructor
6	Classes supports destructors	Structures do not support destructors
7	Default value is NULL	Default value is zero
8	It copies the reference	It copies the value.

Example: using System;

```

struct Books
{
    private string title;
    private string author;
    private string subject;
    private int book_id;
    public void getValues(string t, string a, string s, int id)
    {
        title = t;
        author = a;
        subject = s;
        book_id = id;
    }
    public void display()
    {
        Console.WriteLine("Title : {0}", title);
        Console.WriteLine("Author : {0}", author);
        Console.WriteLine("Subject : {0}", subject);
        Console.WriteLine("Book_id :{0}", book_id);
    }
};
public class testStructure
{
    public static void Main(string[] args)
    {
        Books Book1 = new Books(); /* Declare Book1 of type Book */
        Books Book2 = new Books(); /* Declare Book2 of type Book */
        /* book 1 specification */
        Book1.getValues("C Programming", "Nuha Ali","C Programming
            Tutorial",6495407);
        /* book 2 specification */
        Book2.getValues("Telecom Billing", "Zara Ali", "Telecom Billing
            Tutorial", 6495700);
        /* print Book1 info */
        Book1.display();
        /* print Book2 info */
        Book2.display();
        Console.ReadKey();
    }
}

```

Output: Title : C Programming
 Author : Nuha Ali
 Subject : C Programming Tutorial

Book_id : 6495407
 Title : Telecom Billing
 Author : Zara Ali
 Subject : Telecom Billing Tutorial
 Book_id : 6495700

Features of C# Structures: The C# structures have the following features:

- Structures can have methods, fields, indexers, properties, operator methods, and events.
- Structures can have defined constructors, but not destructors. However, you cannot define a default constructor for a structure.
- Structures cannot inherit other structures or classes. Structures cannot be used as a base for other structures or classes.
- A structure can implement one or more interfaces.
- Structure members cannot be specified as abstract, virtual, or protected.
- When you create a struct object using the new operator, it gets created and the appropriate constructor is called. structs can be instantiated without using the New operator.
- If the New operator is not used, the fields remain unassigned and the object cannot be used until all the fields are initialized.

Enumerations: Enumerated Types allow us to create our own symbolic names for a list of related ideas. It is defined as

Syntax: `enum Identifier{Value 1,value 2,.....value n};`

Example:

```
enum shape
{
    Circle;
    Square;
    Triangle;
}
Or
enum shape
{
    Circle;
    Square;
    Triangle;
};
```

enum: It is the keyword which allows the user to create our own symbolic names for a list of ideas.

Identifier: It is the name of enumerated data type

Example: `enum day{mon,tue,wed....sun};`

The compiler automatically assign integer digits beginning with 0 to all enumeration constants, that is value1=0,value2=1.....however the programmer can change the default values.

```
enum day{ mon=0,tue=10,wed=20...sun=15};
```

Example: The following example demonstrates use of enum variable:

```
using System;
namespace EnumApplication
{
    class EnumProgram
    {
        enum Days { Sun, Mon, tue, Wed, thu, Fri, Sat };
        static void Main(string[] args)
        {
            int WeekdayStart = (int)Days.Mon;
            int WeekdayEnd = (int)Days.Fri;
            Console.WriteLine("Monday: {0}", WeekdayStart);
            Console.WriteLine("Friday: {0}", WeekdayEnd);
            Console.ReadKey();
        }
    }
}
```

Output: Monday: 1
Friday: 5

Enumerator initialization: As mentioned earlier, by default, the value of the first enum number is set to 0, and that of each subsequent member is incremented by one. However we may assign specific values for different members.

Example:

```
enum Days
{
    Sun=1;
    Mon=3;
    Sat=5;
}
```

We can also have expressions.

Example:

```
enum Days
{
    Sun=1;
    Mon=Mon+3;
    Sat=Sun+Mon+5;
}
```

If the declaration of an enum member has no initializer, then its value is set implicitly as follows.

Example:

```
enum Alphabet
{
    A,
    B=5,
    C,
    D=20,
    E
}
```

Output: A=0
B=5
C=6

D=20

E=21

Enumerator base types: By default, The type of an enum is int .However, we can declare explicitly a base type for each enum.The valid base types are: byte, sbyte, short, ushort, int, uint, long and ulong.

Example:

```
enum Position:byte
{
    Off;
    On;
}
```

The values assigned to the members must be within the range of values that can be represented by the base type.

Enumerator type conversion: enum types can be converted to their base types and back again with an explicit conversion using a cast .

Example:

```
enum values
{
    Value0;
    Value1;
    Value2;
}
```

```
Values u1=(Values) 1;
int a=(int) u1;
```

Example:

```
using system
class Enumtype
{
    enum Direction
    {
        North,
        East==10,
        West,
        South
    }
    public static void Main()
    {
        Direction d1=0,
        Direction d2= Direction.East;
        Direction d3= Direction.West;
        Direction d4= (Direction)12;
        Console.WriteLine("d1="+d1);
        Console.WriteLine("d2="+(int)d2);
        Console.WriteLine("d3="+d3);
        Console.WriteLine("d4="+d4);
    }
}
```

Output: d1=North
d2=10
d3=West
d4=South

Example: write a c# program which stores values in two enumerations, Staff and Company. It uses two functions to display the data contained in Staff and Company enumerations.

```
using system;
enum Staff { Directors, Managers, Executives }
enum Company
{
    Newsoftltd,
    TechnologiesInc,
    Hillrockltd
}
class program
{
    public static void Show(staff st)
    {
        switch(st)
        {
            case staff.Directors: Console.WriteLine("you are a director");
                                break;
            case staff.Managers: Console.WriteLine("you are a manager");
                                break;
            case staff.Executives: Console.WriteLine("you are a executive");
                                break;
            default: break;
        }
    }
    public static void CompDisplay(Company com)
    {
        switch(st)
        {
            case Company. Newsoftltd: Console.WriteLine("Newsoftltd");
                                break;
            case Company. TechnologiesInc: Console.WriteLine("TechnologiesInc");
                                break;
            case Company. Hillrockltd: Console.WriteLine("Hillrockltd");
                                break;
            default: break;
        }
    }
    static void Main()
    {
        Staff st;
        st=Staff.Directors;
        Console.WriteLine("this is an example of enumeration");
        Show(st);
        Company com;
```

```
        com=Company.Newsoftltd;  
        Console.WriteLine("Director belongs to Company:");  
        CompDisplay(com);  
    }  
}
```

UNIT II

C# is true object oriented programming language. All object oriented languages employ three core principles, namely

- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**

These are often referred as pillars (or) building blocks of oop.

Encapsulation (or) data hiding (or) information hiding: Encapsulation is the process of hiding irrelevant data from the user. The outside users may not be able to change the state of an object directly. However, the state of an object may be altered indirectly using the access modifier keywords public, private and protected. To understand encapsulation, consider an example of mobile phone. Whenever you buy a mobile, you don't see how circuit board works. You are also not interested to know how digital signal converts into analog signal and vice versa. These are the irrelevant information for the mobile user, that's why it is encapsulated inside a cabinet.

Abstraction: Abstraction is just opposite of Encapsulation. Abstraction is mechanism to show only relevant data to the user. Consider the same mobile example again. Whenever you buy a mobile phone, you see their different types of functionalities as camera, mp3 player, calling function, recording function, multimedia etc. It is abstraction, because you are seeing only relevant information instead of their internal engineering.

Inheritance: It is the concept we use to build new classes using the existing class definitions. The original class is known as base (or) parent class and the modified one is known as derived class (or) subclass (or) child class. Inheritance provides the reusability of existing code and thus improves integrity of programs and productivity of programmers.

Polymorphism: It is the ability to take more than one form. For example, an operation may exhibit different behaviour in different situations. The behaviour depends upon the types of data used in the operation. for example, an addition operation involves two numeric values will produce a sum and

same addition operation will produce a string if the operands are string values instead of numbers.

CLASSES: It is a collection of objects.

Or

A class is a blueprint of an object that contains variables for storing data and functions to perform operations on the data.

Or

A class is a collection of member variables and member functions.

A class will not occupy any memory space and hence it is only a logical representation of data.

Defining a Class: A class definition starts with the keyword class followed by the class name; and the class body enclosed by a pair of curly braces. Following is the general form of a class definition:

```
class class_name
{
    Variable declaration;
    Methods declaration;
}
```

- Class is keyword
- Class name is any C# valid identifier.
- Everything inside the square brackets is called body of the class .It is optional.

Example: class class_name
 {
 }
 }

In the above class body is empty, this class does not contain any properties and therefore cannot do anything.

Adding variables: data is encapsulated in a class by placing the data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. We can declare the instance variables exactly the same way as we declare local variables.

Example: class Rectangle
 {
 int length;
 int width;
 }

The class Rectangle contains two integer type instance variables. Remember these variables are only declared and therefore no storage space has been created in the memory. Instance variables are also known as member variables.

Adding methods: methods are declared inside the body of the class, usually after the declaration of instance variables.

Syntax: type methodname(parameter_list)
 {

Method-body;

}

The method body actually describes the operations to be performed on the data.

Example:

```
class Rectangle
{
    int length;
    int width;
    public void GetData(int x,int y)
    {
        length=x;
        width=y;
    }
}
```

Note that the method has a return type void because it does not return any value. We pass two integer values to the method, which are then assigned to the instance variables length and width. The GetData() method is basically added to provide values to the instance variables. Now we are able to use directly length and width inside the method.

Example:

```
class Rectangle
{
    int length;
    int width;
    public void GetData(int x,int y)
    {
        length=x;
        width=y;
    }
    public int RectArea()
    {
        int area=length*width;
        return area;
    }
}
```

The method RectArea() computes the area of the rectangle and returns the result.

Member access modifiers: the goal of oop is data hiding. That is a class may be designed to hide its members from outside accessibility. C# provides a set of access modifiers that can be used with the members of a class to control their visibility to outside users.

Modifier	Accessibility control
private	Private members of a class are completely restricted and are accessible only within the class in which they are declared.
public	Member is accessible from anywhere outside the class as well. It is accessible in derived classes.
protected	Member is visible only its own class and its derived classes
internal	Member is available with in the assembly or component that is being created but not to the clients of that component.
Protected	Available in the containing program or assembly and in the derived classes.

internal	
----------	--

In C# all members have private access by default. If we want a member to have any other visibility range, then we must specify a suitable access modifier to it individually.

Example:

```
class Visibility
{
    public int x;
    internal int y;
    protected double d;
    float p;
}
```

Local variables: The variables which are declared within the method is called local variables.

Example:

```
public void print()
{
    int a;           // local variable
    string s;       // local variable
}
```

Instance variable: The variables which are created within the class are called instance variables.

Example:

```
class abc
{
    private int a;           // Instance variable
    private string s;       // Instance variable
}
```

Note: when the local variable name and instance variable name is same then default the importance is given to local variables.

Object: An object is an instance of a class

Creating objects: Object in c# is created using the new operator .the new operator creates an object of the specified class and returns a reference to that object.

Syntax: classname object=new classname();

Example:

```
Rectangle rect1 = new Rectangle();
Rectangle rect2 = new Rectangle();
```

Accessing class members: Now we have created objects, each containing its own set of variables, we should assign values to these variables in order to use them in our program. Since we are outside the class, we cannot access the instance variables and the methods directly. To do this, we must use the concerned object and dot operator.

Syntax: objectname.variablename;
objectname.methodname(parameter-list);

Where,

- **Objectname:** It is the name of the object.
- **Variablename:** It is the name of the instance variable inside the object that we wish to access.
- **Methodname:** It is the name of the method we wish to call.
- **parameter-list:** It is a comma separated list actual values that must match in type and

number with the parameter-list of the method name declared in the class.

The instance variables of the rectangle class may be accessed and assigned value as follows.

```
rect1.length=15;
rect1.width=10;
rect2.length=20;
rect2.width=12;
```

Another way of assigning values to the instance variables is to use a method that is declared inside a class. Here the method GetData can be used to do this work. We can call the GetData method on any Rectangle object to set the values of both length and width.

Example: Rectangle rect1 = new Rectangle();
 rect1.GetData(15,10);

Example: using System;
 class Rectangle
 {
 int length;
 int width;
 public void GetData(int x,int y)
 {
 length=x;
 width=y;
 }
 public int RectArea()
 {
 int area=length*width;
 return area;
 }
 }
 Class RectangleArea
 {
 public static void Main()
 {
 int area1,area2;
 Rectangle rect1 = new Rectangle();
 Rectangle rect2 = new Rectangle();
 rect1.length=15;
 rect1.width=10;
 area1= rect1.length* rect1.width;
 rect2.GetData(20,12);
 area2=rect2.RectArea();
 Console.WriteLine(Area1="+area1);
 Console.WriteLine(Area2="+area2);
 }
 }
 }
 using System;
 namespace BoxApplication
 {
 class Box

```

    {
        public double length; // Length of a box
        public double breadth; // Breadth of a box
        public double height; // Height of a box
    }
    class Boxtester
    {
        static void Main(string[] args)
        {
            Box Box1 = new Box(); // Declare Box1 of type Box
            Box Box2 = new Box(); // Declare Box2 of type Box
            double volume = 0.0; // Store the volume of a box here
            Box1.height = 5.0;
            Box1.length = 6.0;
            Box1.breadth = 7.0;
            Box2.height = 10.0;
            Box2.length = 12.0;
            Box2.breadth = 13.0;
            volume = Box1.height * Box1.length * Box1.breadth;
            Console.WriteLine("Volume of Box1 : {0}", volume);
            volume = Box2.height * Box2.length * Box2.breadth;
            Console.WriteLine("Volume of Box2 : {0}", volume);
            Console.ReadKey();
        }
    }
}

```

Output: Volume of Box1 : 210
Volume of Box2 : 1560

Main(): as mentioned earlier, C# programs start execution at a Main() method. This method must be static method of a class and must have either int or void return type.

```

public static int Main()
public static void Main()

```

The Main() method can also have parameters which may receive values from the command line at the time execution.

```

public static int Main(string[] args)
public static void Main(string[] args)

```

METHOD: A method is a self-contained block of code that performs a particular task. Every C# program has at least one class with a method named Main.

Defining Methods (or) declaring methods: methods are declared inside the body of the class, normally after the declaration of data fields.

Syntax:

```

< Access modifier > <Return Type> <Method Name> (Parameter List)
{
    Method Body
}

```

Where,

Method name: Method name is a valid c# identifier and it is case sensitive. It cannot be same as any other identifier declared in the class.

Parameter list: Enclosed between parentheses, the parameters are used to pass and receive data from a method. The parameter list refers to the type, order, and number of the parameters of a method. Parameters are optional; that is, a method may contain no parameters.

Method body: This contains the set of instructions needed to complete the required activity and method body can be enclosed in curly braces.

Return type: Return type specifies the type of value the method will return. If the method is not returning any values, then the return type is void.

Access modifier: This determines the visibility of a variable or a method from another class.

List of Method Modifiers:

Modifier	Description
new	The method hides an inherited method with the same signature.
public	The method can be accessed from anywhere, including outside the class
protected	The method can be accessed from within the class to which it belongs or a type derived from that class
internal	The method can be accessed from within the same program
private	The method can only be accessed from inside the class to which it belongs.
static	The method does not operate on a specific instance of the class
virtual	The method can be overridden by a derived class
abstract	A virtual method which defines the signature of the method, but does not provide an implementation.
override	The method overrides an inherited virtual or abstract method
sealed	The method overrides an inherited virtual method, but cannot be overridden by any classes which inherit from this class. Must be used in conjunction with override.
extern	This method is implemented externally, in a different language.

Invoking or Calling Methods: once methods have been defined, they must be activated for operations. The process of activating a method is known as invoking (or) calling. You can call a method using dot operator.

Syntax: **objectname.methodname(actual-parameterlist);**

Here, object name is the name of the object on which we are calling the method.

The actual parameter list is a comma separated list of actual values that must match in type, order and number with formal parameter list method declared in class

Example: using System;

```
namespace CalculatorApplication
{
    class NumberManipulator
    {
        public int FindMax(int num1, int num2)
        {
            int result;
            if (num1 > num2)
                result = num1;
            else
```

```

        result = num2;
        return result;
    }
    static void Main(string[] args)
    {
        int a = 100;
        int b = 200;
        int ret;
        NumberManipulator n = new NumberManipulator();
        ret = n.FindMax(a, b);
        Console.WriteLine("Max value is : {0}", ret );
        Console.ReadLine();
    }
}

```

Output: Max value is : 200

Example:

```

using System;
namespace CalculatorApplication
{
    class Method
    {
        int Cube(int x)
        {
            return (x*x*x);
        }
    }
    class MethodTest
    {
        static void Main(string[] args)
        {
            Method M = new Method();
            int y=M.Cube(5);
            Console.WriteLine("y" );
        }
    }
}

```

Output: Max value is: 125

Nesting of methods: we have earlier that a method can be called only by an object of that class if it is called outside the class. We can also called a method without using any object or dot operator. That is, a method can be called only its name by another method of the same class. This is known as nesting of methods.

Example:

```

using System;
class nesting
{
    void largest(int m,int n)
    {

```

```

        int large=Max(m,n);
        Console.WriteLine(large);
    }
    int Max(int a, int b)
    {
        int x=(a>b)?a:b;
        return x;
    }
}
class nesttesting
{
    public static void Main()
    {
        nesting next=new nesting();
        next.largest(100,200);
    }
}

```

Method parameters:

- **Input parameters:** Input parameters are used for bringing a value into the method for execution.
- **Value parameters:** Value parameters are used for passing parameters into methods by value.
- **Reference parameters:** Reference parameters are used to pass parameters into methods by reference.
- **Output parameters:** Output parameters are used to pass results back from a method.
- **Parameter arrays:** Parameter arrays are used in a method definition to enable it to receive variable number of arguments when called.

Parameters Passing Methods: There are three ways that parameters can be passed to a method.

Passing Parameters by Value: In this type value of actual arguments are passed to the formal arguments of the called function. Any changes made in the formal arguments does not effect the actual arguments in the calling function.

Or

- ✚ In call by value method, the value of the variable is passed to the function as parameter.
- ✚ The value of the actual parameter cannot be modified by formal parameter.
- ✚ Different Memory is allocated for both actual and formal parameters. Because, value of actual parameter is copied to formal parameter.

Example:

```

using System;
namespace CalculatorApplication
{
    class NumberManipulator
    {

```

```

public void swap(int x, int y)
{
    int temp;
    temp = x;
    x = y;
    y = temp;
}
static void Main(string[] args)
{
    NumberManipulator n = new NumberManipulator();
    int a = 100;
    int b = 200;
    Console.WriteLine("Before swap, value of a : {0}", a);
    Console.WriteLine("Before swap, value of b : {0}", b);
    n.swap(a, b);
    Console.WriteLine("After swap, value of a : {0}", a);
    Console.WriteLine("After swap, value of b : {0}", b);
    Console.ReadLine();
}
}
}

```

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

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

Passing Parameters by Reference: In call by reference, the address of actual arguments is passed to formal arguments of the called function and any change made to the formal arguments in the called function have effect on the values of actual arguments in the calling function

Or

- ✚ In call by reference method, the address of the variable is passed to the function as parameter.
- ✚ The value of the actual parameter can be modified by formal parameter.
- ✚ Same memory is used for both actual and formal parameters since only address is used by both parameters

You can declare the reference parameters using the **ref** keyword.

Example:

```

using System;
namespace CalculatorApplication
{
    class NumberManipulator
    {
        public void swap(ref int x, ref int y)
        {
            int temp;
            temp = x;

```

```

        x = y;
        y = temp;
    }
    static void Main(string[] args)
    {
        NumberManipulator n = new NumberManipulator();
        int a = 100;
        int b = 200;
        Console.WriteLine("Before swap, value of a : {0}", a);
        Console.WriteLine("Before swap, value of b : {0}", b);
        n.swap(ref a, ref b);
        Console.WriteLine("After swap, value of a : {0}", a);
        Console.WriteLine("After swap, value of b : {0}", b);
        Console.ReadLine();
    }
}

```

Output:

```

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

```

It shows that the values have changed inside the swap function and this change reflects in the Main function.

Passing Parameters by Output: Output parameters are used to pass results back to the calling method. This is achieved by declaring the parameters with an out keyword. Output parameters are similar to reference parameters, except that they transfer data back to the calling method.

Example:

```

using System;
namespace CalculatorApplication
{
    class output
    {
        public void quare(int x,out int y )
        {
            y=x*x;
        }
        static void Main(string[] args)
        {
            int m;
            square(10, out m);
            Console.WriteLine("m="+m);
            Console.ReadLine();
        }
    }
}

```

Output: m=100

Example:

```

using System;
namespace CalculatorApplication
{
    class NumberManipulator
    {
        public void getValue(out int x )
        {
            int temp = 5;
            x = temp;
        }
        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            int a = 100;
            Console.WriteLine("Before method call, value of a : {0}", a);
            n.getValue(out a);
            Console.WriteLine("After method call, value of a : {0}", a);
            Console.ReadLine();
        }
    }
}

```

Output: Before method call, value of a : 100
After method call, value of a : 5

The variable supplied for the output parameter need not be assigned a value. Output parameters are particularly useful when you need to return values from a method through the parameters without assigning an initial value to the parameter. Go through the following example, to understand this:

```

using System;
namespace CalculatorApplication
{
    class NumberManipulator
    {
        public void getValues(out int x, out int y )
        {
            Console.WriteLine("Enter the first value: ");
            x = Convert.ToInt32(Console.ReadLine());
            Console.WriteLine("Enter the second value: ");
            y = Convert.ToInt32(Console.ReadLine());
        }
        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            int a , b;
            n.getValues(out a, out b);
            Console.WriteLine("After method call, value of a : {0}", a);
            Console.WriteLine("After method call, value of b : {0}", b);
            Console.ReadLine();
        }
    }
}

```

```

    }
Output:      Enter the first value:7
                Enter the second value:8
                After method call, value of a : 7
                After method call, value of b : 8

```

```

Example:    using System;
                using System.Collections.Generic;
                using System.Linq;
                using System.Text;
                using System.Text.RegularExpressions;
                namespace ConsoleApplication7
                {
                    class Rectangle
                    {
                        public void swap(int x,int y,out int h, out int k)
                        {
                            int temp;
                            temp = x;
                            x = y;
                            y = temp;
                            h = x;
                            k = y;
                        }
                    }
                    static void Main(string[] args)
                    {
                        Rectangle n = new Rectangle();
                        int a = 100;
                        int b = 200; int d, c;
                        Console.WriteLine("Before swap, value of a : {0}", a);
                        Console.WriteLine("Before swap, value of b : {0}", b);
                        n.swap(a,b,out d,out c);
                        Console.WriteLine("After swap, value of a : {0}", d);
                        Console.WriteLine("After swap, value of b : {0}", c);
                        Console.ReadLine();
                    }
                }

```

Variable argument list (or) Param Arrays: At times, while declaring a method, you are not sure of the number of arguments passed as a parameter. C# param arrays (or parameter arrays) come into help at such times.in C# we can define methods that can handle variable number of arguments is called parameter arrays. Parameter arrays are declared using the keyword **params**.

```

Example:    using System;
                namespace ArrayApplication
                {
                    class ParamArray
                    {

```

```

        public int AddElements(params int[] arr)
        {
            int sum = 0;
            foreach (int i in arr)
                sum += i;
            return sum;
        }
    }
}
class TestClass
{
    static void Main(string[] args)
    {
        ParamArray app = new ParamArray();
        int sum = app.AddElements(512, 720, 250, 567, 889);
        Console.WriteLine("The sum is: {0}", sum);
        Console.ReadKey();
    }
}
}

```

Output: The sum is: 2938

Recursive Method Call: A method can call itself. This is known as recursion. Following is an example that calculates factorial for a given number using a recursive function:

```

using System;
namespace CalculatorApplication
{
    class NumberManipulator
    {
        public int factorial(int num)
        {
            int result;
            if (num == 1)
                return 1;
            else
            {
                result = factorial(num - 1) * num;
                return result;
            }
        }
        static void Main(string[] args)
        {
            NumberManipulator n = new NumberManipulator();
            Console.WriteLine("Factorial of 6 is : {0}", n.factorial(6));
            Console.WriteLine("Factorial of 7 is : {0}", n.factorial(7));
            Console.WriteLine("Factorial of 8 is : {0}", n.factorial(8));
            Console.ReadLine();
        }
    }
}
}

```

Output: Factorial of 6 is: 720
 Factorial of 7 is: 5040
 Factorial of 8 is: 40320

Method overloading: C# allows us to create more than one method with same name, but with different parameter list and different definitions. This is called method overloading. Method overloading is used when methods are required to perform similar tasks but using different input parameters. Overloaded methods must differ in number and/or type of parameters they take. This enables the compiler to decide which one of the definitions to execute depending on the type and number of arguments in the method call.

Example:

```
using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication3
{
    class Method_overloading
    {
        public int Addition(int a, int b)
        {
            int x;
            return x=a+b;
        }
        public int Addition(int a, int b,int c)
        {
            int y;
            return y = a + b+ c;
        }
        public float Addition(float a, float b)
        {
            float u;
            return u = a + b;
        }
        public float Addition(float a, float b, float c)
        {
            float v;
            return v = a + b+ c;
        }
    }
}
class hub
{
    public static void Main(String[] args)
    {
        Method_overloading mthover = new Method_overloading();
        Console.WriteLine("Addition of two integers::::::::::::::::::" + mthover.Addition(2, 5));
        Console.WriteLine("Addition of two double type values:::::::" + mthover.Addition(0.40f, 0.50f));
        Console.WriteLine("Addition of three integers::::::::::::::::::" + mthover.Addition(2, 5, 5));
        Console.WriteLine("Addition of three double type values:" + mthover.Addition(0.40f, 0.50f, 0.60f));
    }
}
```

```

        Console.ReadLine();
    }
}

```

C# CONSTRUCTORS: constructor is a special type of method which is automatically executed when the object is created. A constructor has exactly the same name as that of the class name and it does not have any return type. Constructors are responsible for object initialization and memory allocation of its class. If we create any class without constructor, the compiler will automatically create one default constructor for that class. The default constructor initializes all numeric fields in the class to zero and all string and object fields to null.

Some of the key points regarding the Constructor are:

- A class can have any number of constructors.
- A constructor doesn't have any return type, not even void.
- A static constructor cannot be a parameterized constructor.
- Within a class you can create only one static constructor

Types of Constructors: Basically constructors are 5 types those are

1. Default Constructor
2. Parameterized Constructor
3. Copy Constructor
4. Static Constructor
5. Private Constructor

Default Constructor: A constructor without any parameters is called a default constructor; in other words this type of constructor does not take parameters. The drawback of a default constructor is that every instance of the class will be initialized to the same values and it is not possible to initialize each instance of the class to different values. The default constructor initializes.

1. All numeric fields in the class to zero.
2. All string and object fields to null.

Example:

```

using System;
namespace ConsoleApplication3
{
    class Sample
    {
        public string param1, param2;
        public Sample() // Default Constructor
        {
            param1 = "Welcome";
            param2 = "Aspdotnet-Suresh";
        }
    }
}

```

```

    }
}
class Program
{
    static void Main(string[] args)
    {
        Sample obj=new Sample();
        Console.WriteLine(obj.param1);
        Console.WriteLine(obj.param2);
        Console.ReadLine();
    }
}

```

Output: Welcome
Aspdotnet-Suresh

Example:

```

using System;
namespace LineApplication
{
    class Line
    {
        private double length;
        public Line()
        {
            Console.WriteLine("Object is being created");
        }
        public void setLength( double len )
        {
            length = len;
        }
        public double getLength()
        {
            return length;
        }
        static void Main(string[] args)
        {
            Line line = new Line();
            line.setLength(6.0);
            Console.WriteLine("Length of line : {0}", line.getLength());
            Console.ReadKey();
        }
    }
}

```

Result: Object is being created
Length of line : 6

Parameterized Constructors: A constructor with at least one parameter is called as parameterized constructor. The advantage of a parameterized constructor is that you can initialize each instance of

the class to different values.

Example:

```
using System;
namespace ConsoleApplication3
{
    class Sample
    {
        public string param1, param2;
        public Sample(string x, string y)
        {
            param1 = x;
            param2 = y;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Sample obj=new Sample("Welcome","Aspdotnet-Suresh");
            Sample obj1=new Sample("Welcome1","Aspdotnet-Suresh1");
            Console.WriteLine(obj.param1 +" to "+ obj.param2);
            Console.WriteLine(obj1.param1 +" to "+ obj1.param2);
            Console.ReadLine();
        }
    }
}
```

Output:
Welcome to Aspdotnet-Suresh
Welcome1 to Aspdotnet-Suresh1

Copy Constructor: The constructor which creates an object by copying variables from another object is called a copy constructor. Main purpose of copy constructor is to initialize new object to the values of an existing object. Here the constructor contains a parameter, which is of class type.

Example:

```
using System;
namespace ConsoleApplication3
{
    class Sample
    {
        public string param1, param2;
        public Sample(string x, string y)
        {
            param1 = x;
            param2 = y;
        }
        public Sample(Sample obj) // Copy Constructor
        {
            param1 = obj.param1;
            param2 = obj.param2;
        }
    }
}
```

```

class Program
{
    static void Main(string[] args)
    {
        Sample obj = new Sample("Welcome", "Aspdotnet-Suresh");
        Sample obj1=new Sample(obj);
        Console.WriteLine(obj1.param1 +" to " + obj1.param2);
        Console.ReadLine();
    }
}

```

Output: Welcome to Aspdotnet-Suresh

Static Constructor: A **static** constructor is used to initialize any static data (or) to perform a particular action that needs to be performed once only. static constructor will be invoked only once for all of objects of the class. Static constructor gets called before the first object of the class is created.

Importance points of static constructor

- Static constructor will not accept any parameters because it is automatically called by CLR.
- Static constructor will not have any access modifiers.
- Static constructor will execute automatically whenever we create first instance of class
- Only one static constructor will allowed.

Example:

```

using System;
namespace ConsoleApplication3
{
    class Sample
    {
        public string param1, param2;
        static Sample()
        {
            Console.WriteLine("Static Constructor");
        }
        public Sample()
        {
            param1 = "Sample";
            param2 = "Instance Constructor";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Here Both Static and instance constructors are invoked for first instance
            Sample obj=new Sample();
            Console.WriteLine(obj.param1 + " " + obj.param2);
            // Here only instance constructor will be invoked

```

```

        Sample obj1 = new Sample();
        Console.WriteLine(obj1.param1 + " " + obj1.param2);
        Console.ReadLine();
    }
}

```

Output: Static Constructor
Sample Instance Constructor
Sample Instance Constructor

Private Constructor: You can also create a constructor as private. When a class contains at least one private constructor, then it is not possible to create an instance for the class. Private constructor is used to restrict the class from being instantiated when it contains every member as static.

Important points of private constructor

- One use of private construct is when we have only static member.
- Once we provide a constructor that is either private or public or any, the compiler will not allow us to add public constructor without parameters to the class.
- If we want to create object of class even if we have private constructors then we need to have public constructor along with private constructor

Example:

```

using System;
namespace ConsoleApplication3
{
    public class Sample
    {
        public string param1, param2;
        public Sample(string a,string b)
        {
            param1 = a;
            param2 = b;
        }
        private Sample() // Private Constructor Declaration
        {
            Console.WriteLine("Private Constructor with no prameters");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Here we don't have chance to create instace for private constructor
            Sample obj = new Sample("Welcome","to Aspdotnet-Suresh");
            Console.WriteLine(obj.param1 + " " + obj.param2);
            Console.ReadLine();
        }
    }
}

```

```
    }
}
```

Output: Welcome to Aspdotnet-Suresh

Note: In above method we can create object of class with parameters will work fine. If create object of class without parameters it will not allow us create.

```
// it will works fine
Sample obj = new Sample("Welcome","to Aspdotnet-Suresh");
// it will not work because of inaccessability
Sample obj=new Sample();
```

Example: using System;
namespace defaultConstructor
{

```
    public class Counter
    {
        private Counter() //private constructor declaration
        {
        }
        public static int currentview;
        public static int visitedCount()
        {
            return ++ currentview;
        }
    }
    class viewCountedetails
    {
        static void Main()
        {
            // Counter aCounter = new Counter(); // Error
            Console.WriteLine("Private constructor example ");
            Console.WriteLine();
            Counter.currentview = 500;
            Counter.visitedCount();
            Console.WriteLine(" view count is: {0}",
            Counter.currentview);
            Console.ReadLine();
        }
    }
}
```

Constructor Overloading: The process of creating more than one constructor with same name, which is similar to class name, but with different parameters is called constructor overloading.

Example: using System;
namespace ConsoleApplication3
{
 class Sample
 {
 public string param1, param2;
 public Sample() // Default Constructor
 {

```

        param1 = "Hi";
        param2 = "I am Default Constructor";
    }
    public Sample(string x, string y)
    // Declaring Parameterized constructor with Parameters
    {
        param1 = x;
        param2 = y;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Sample obj= new Sample(); // Default Constructor will Called
        Sample obj1=new Sample("Welcome","Aspdotnet-Suresh");
        // Parameterized Constructor will Called
        Console.WriteLine(obj.param1 + ", "+obj.param2);
        Console.WriteLine(obj1.param1 +" to " + obj1.param2);
        Console.ReadLine();
    }
}

```

Output: Hi, I am Default Constructor
Welcome to Aspdotnet-Suresh

C# DESTRUCTORS: A destructor is a special method that is automatically executed when an object is destroyed. A destructor has exactly the same name as that of the class name with a prefixed tilde (~) and without return a value and without access specifier. Destructor can be very useful for releasing memory resources before exiting the program. Destructors cannot be inherited or overloaded.

Example:

```

using System;
namespace LineApplication
{
    class Line
    {
        private double length;
        public Line()
        {
            Console.WriteLine("Object is being created");
        }
        ~Line()
        {
            Console.WriteLine("Object is being deleted");
        }
        public void setLength( double len )
        {
            length = len;
        }
    }
}

```

```

        public double getLength()
        {
            return length;
        }
        static void Main(string[] args)
        {
            Line line = new Line();
            line.setLength(6.0);
            Console.WriteLine("Length of line : {0}", line.getLength());
        }
    }
}

```

Output: Object is being created
Length of line : 6
Object is being deleted

STATIC CONCEPTS:

Static Classes: C# provides the feature to create static classes. There are two main features of static classes

- We cannot create object for static classes
- Static classes must contain static members

The main benefit to create a static class is we do not need to make any instance of the class and all members of the can be accessed by its name (i.e. class name).A static class can be created using static keyword

The features of a static class are as follows:

- static classes can only contain static members.
- static classes cannot be instantiated.
- static classes are sealed and therefore cannot be inherited.
- static classes cannot contain Instance Constructors .

Example:

```

using System;
using System.Collections.Generic;
using System.Text;
namespace ConsoleApplication8
{
    public static class Square
    {
        public static double side;
        public static double Perimeter()
        {
            return side * 4;
        }
        public static double Area()
        {
            return side * side;
        }
    }
}

```

```

    }
    }
    public class Exercise
    {
    public static void Main()
    {
        Square.side = 36.84;
        Console.WriteLine("Square Characteristics");
        Console.Write("Side:  ");
        Console.WriteLine(Square.side);
        Console.Write("Perimeter: ");
        Console.WriteLine(Square.Perimeter());
        Console.Write("Area:  ");
        Console.WriteLine(Square.Area());
    }
    }
}

```

Output: Square Characteristics
 Side: 36.84
 Perimeter: 147.36
 Area: 1357.1856

Static Members: The members of a class that can be accessed without creating an instance and directly by using class name are called as static members.

Static variable: a variable that is declared by using a static modifier (or) a variable that is declared inside of any static block is called static variable.

Example: class MySettings

```

    {
        public static int height;
        public static int width = 20;
    }

```

a static variable is get initialized immediately once the execution of class starts.

A static variable is initialized one time in the life cycle of the class.

A static variable of the class can be accessed using the class name.

Static Methods: A method that is declared by using a static modifier is called static method. When a method is declared as static then that method can access only other static members available in the class and it is not possible to access instance members

static int max(int x, int y);

Example: using system;

```

namespace example
{
    class mathoperation
    {
        public static float mul(float x, float y)
        {

```

```

        return x*y;
    }
    public static float divide(float x, float y)
    {
        return x/y;
    }
}
class mathapplication
{
    public static void Main()
    {
        float a=mathoperation.mul(4.0f,5.0f);
        float b=mathoperation.divide(a,2.0f);
        Console.WriteLine("b="+b);
    }
}
}

```

Example: using system;
namespace example
{

```

    class MySettings
    {
        private static int height = 100;
        private static int width = 150;
        public static void MyMethod()
        {
            Console.WriteLine("{0},{1}", height, width);
        }
    }
    class AllSettings
    {
        public static void Main()
        {
            MyClass.MyMethod();
        }
    }
}

```

Example: using system;
namespace Static_var_and_fun
{
 class number
 {
 // Create static variable
 public static int num;
 //Create static method
 public static void power()
 {
 Console.WriteLine("Power of {0} = {1}", num, num * num);
 }
 }
}

```

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter a number\t");
        number.num = Convert.ToInt32(Console.ReadLine());
        number.power();
    }
}

```

CONSTANTS: constants are immutable values which are known at compile time and do not change for the life of the program. Constants are declared using const modifier.

- The const modifier is used to declare local variables that cannot be changed.
- Constant variables must be given initial values when they are declared.

public const int size=100;

- The behaviour of const variable is similar to static variable, except const variable cannot be modified.

Example:

```

using System;
namespace UsingConst
{
    class Program
    {
        const int a = 10;
        static void Main(string[] args)
        {
            const int b = 20;
            const int c = b + a;
            Console.WriteLine(c);
            Console.ReadLine();
        }
    }
}

```

readonly variable: A readonly field can be initialized either at the time of declaration or with in the constructor of same class. Therefore, readonly fields can be used for run-time constants.

Example:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication8
{
    class SampleClass
    {
        public int x;
        // Initialize a readonly field
        public readonly int y = 25;
    }
}

```

```

        public readonly int z;
        public SampleClass()
        {
            // Initialize a readonly instance field
            z = 24;
        }
        public SampleClass(int p1, int p2, int p3)
        {
            x = p1;
            y = p2;
            z = p3;
        }
        static void Main()
        {
            SampleClass k1 = new SampleClass(11, 21, 32);
            Console.WriteLine("p1:x={0},y={1},z={2}",p1.x,p1.y,p1.z);
            SampleClass k2 = new SampleClass();
            p2.x = 55;
            Console.WriteLine("p2:x={0},y={1},z={2}",p2.x,p2.y,p2.z);
        }
    }
}

```

this reference: The **this** keyword refers to the current instance of the class. This reference is available within all the member methods and always refers to the current instance. It is used to distinguish local and instance variables that have the same name.

Example:

```

using System;
class Demo
{
    int a=2,b=10;
    public void Get()
    {
        int a=23,b=34;
        Console.WriteLine("a={0} b={1}",this.a,this.b);
        Console.WriteLine("It is the class variable");
        Console.WriteLine("Now the local variables are:");
        Console.WriteLine("a={0} b={1}",a,b);
    }
}
class MainClass
{
    static void Main(string args[])
    {
        Demo d= new Demo();
        d.Get();
    }
}

```

Nesting of classes: To define a class within the scope of another class is called nesting of classes.

Syntax: **public class outer**

```

    {
        .....
        .....
        public class inner
        {
            .....
            .....
        }
    }
}

```

Example:

```

using System;
namespace innerclassexample
{
    public class nestedclass
    {
        public static void Main()
        {
            outerclass obj2=new outerclass();
            obj2.show();
            outerclass.innerclass obj1=new outerclass.innerclass();
            obj1.display();
        }
    }
    Public class outerclass
    {
        public void show()
        {
            Console.WriteLine("outer class");
        }
        public class innerclass
        {
            public void display()
            {
                Console.WriteLine("innerclass class");
            }
        }
    }
}

```

INHERITANCE: One of the most important concepts in object-oriented programming is inheritance.

Creating a new class from existing class is called as inheritance

(Or)

Acquiring (taking) the properties of one class into another class is called inheritance.

(Or)

When a new class needs same members as an existing class, then instead of creating those members again in new class, the new class can be created from existing class, which is called as inheritance.

Main advantage of inheritance is **reusability** of the code. During inheritance, the class that is inherited is called as base class and the class that does the inheritance is called as derived class.

Base class: is the class from which features are to be inherited into another class.

Syntax:

```
class base-class-name
{
    Members of class
}
```

Derived class: it is the class in which the base class features are inherited.

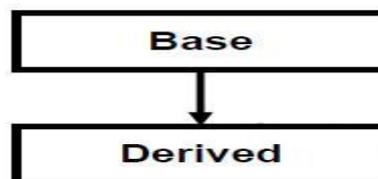
Syntax:

```
class derived-class-name: base-class-name
{
    Members of class
}
```

Types of inheritance: Inheritance can be classified into 5 types

1. Single Inheritance
2. Hierarchical Inheritance
3. Multi-Level Inheritance
4. Hybrid Inheritance
5. Multiple Inheritance

Single Inheritance: when a single derived class is created from a single base class then the inheritance is called as single inheritance.



Syntax:

```
class A
{
    ----
    ----
}
class B:A
{
    ----
    ----
}
```

Example:

```
using System;
namespace InheritanceApplication
{
    class a
    {
        public void display()
        {
```

```

        System.Console.WriteLine("hahahaha");
    }
}
class b : a //b is child of a
{
    public void display1()
    {
        System.Console.WriteLine("hihihih");
    }
}
class c
{
    public static void Main()
    {
        b x=new b();//Normally object of child
        x.display();
        x.display1();
    }
}
}

```

Example:

```

using System;
namespace InheritanceApplication
{
    class Shape
    {
        public void setWidth(int w)
        {
            width = w;
        }
        public void setHeight(int h)
        {
            height = h;
        }
        protected int width;
        protected int height;
    }
    class Rectangle: Shape
    {
        public int getArea()
        {
            return (width * height);
        }
    }
    class RectangleTester
    {
        static void Main(string[] args)
        {
            Rectangle Rect = new Rectangle();
            Rect.setWidth(5);
        }
    }
}

```

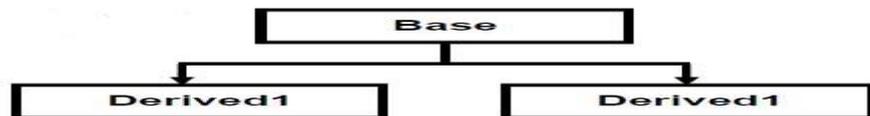
```

        Rect.setHeight(7);
        Console.WriteLine("Total area: {0}",Rect.getArea());
        Console.ReadKey();
    }
}
}

```

Output: Total area: 35

Hierarchical Inheritance: when more than one derived class are created from a single base class, then that inheritance is called as hierarchical inheritance.



Syntax:

```

class A
{
    ----
    ----
}
class B:A
{
    ----
    ----
}
class C:A
{
    ----
    ----
}

```

Example:

C# Program to Illustrate Hierarchical Inheritance

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace Inheritance
{
    class Program
    {
        static void Main(string[] args)
        {
            Principal g = new Principal();
            g.Monitor();
            Teacher d = new Teacher();
            d.Monitor();
            d.Teach();
            Student s = new Student();
            s.Monitor();
            s.Learn();
            Console.ReadKey();
        }
    }
}

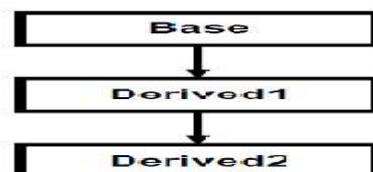
```

```

    }
    class Principal
    {
        public void Monitor()
        {
            Console.WriteLine("Monitor");
        }
    }
    class Teacher : Principal
    {
        public void Teach()
        {
            Console.WriteLine("Teach");
        }
    }
    class Student : Principal
    {
        public void Learn()
        {
            Console.WriteLine("Learn");
        }
    }
}

```

Multi-Level Inheritance: when a derived class is created from another derived class, then that inheritance is called as multi-level inheritance.



Syntax:

```

class A
{
    ----
    ----
}
class B:A
{
    ----
    ----
}
class C:B
{
    ----
    ----
}

```

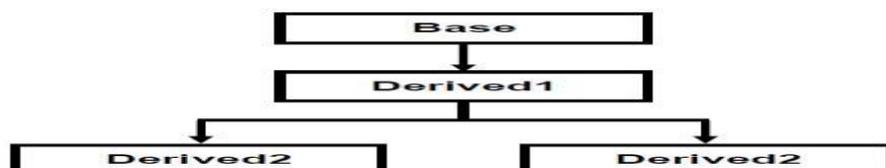
Example: C# Program to Illustrate Multilevel Inheritance with Virtual Methods using System;

```

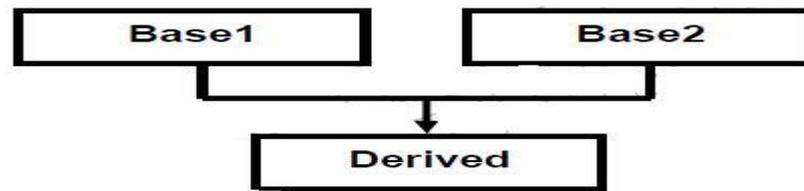
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication8
{
    class a
    {
        public void display()
        {
            Console.WriteLine("hahahaha");
        }
    }
    class b : a //b is child of a
    {
        public void display1()
        {
            Console.WriteLine("hihihih");
        }
    }
    class d : b //d is child of b
    {
        public void display2()
        {
            Console.WriteLine("hohohohoh");
        }
    }
    class c
    {
        public static void Main()
        {
            d x = new d();//Normally object of child
            x.display();
            x.display1();
            x.display2();
        }
    }
}

```

Hybrid Inheritance: Any combination of single, hierarchical and multi-level inheritances is called as hybrid inheritance.



Multiple Inheritance: when a derived class is created from more than one base class then that inheritance is called as multiple inheritance.



But multiple and hybrid inheritance is not supported by .net using classes and can be done using interfaces.

POLYMORPHISM: Polymorphism means one name many forms (ability to take more than one form). In Polymorphism poly means “multiple” and morph means “forms” so polymorphism means many. In polymorphism we will declare methods with same name and different parameters in same class or methods with same name and same parameters in different classes. Polymorphism has ability to provide different implementation of methods that are implemented with same name.

Types of polymorphism:

- **Compile Time Polymorphism(or)Early Binding (or) Overloading (or) static binding**
- **Run Time Polymorphism(or)Late Binding (or) Overriding (or) dynamic binding**

Static Polymorphism: The mechanism of linking a function with an object during compile time is called early binding. It is also called static binding. C# provides two techniques to implement static polymorphism. They are:

1. **Function (or) method overloading**
2. **Constructor overloading.**
3. **Operator overloading**

Function (or) method Overloading: The process of creating more than one method in a class with same name is called as method overloading. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list.

Example:

```

using System;
namespace PolymorphismApplication
{
    class Printdata
    {
        void print(int i)
        {
            Console.WriteLine("Printing int: {0}", i);
        }
        void print(double f)
        {
            Console.WriteLine("Printing float: {0}" , f);
        }
        void print(string s)
    }
}
  
```

```

        {
            Console.WriteLine("Printing string: {0}", s);
        }
        static void Main(string[] args)
        {
            Printdata p = new Printdata();
            p.print(5);
            p.print(500.263);
            p.print("Hello C++");
            Console.ReadKey();
        }
    }
}

```

Output:
 Printing int: 5
 Printing float: 500.263
 Printing string: Hello C++

Example:
 using System;
 namespace MethodOverloadingByManishAgrahari

```

    {
        class Program
        {
            public class TestOverloading
            {
                public void Add(string a1, string a2)
                {
                    Console.WriteLine("Adding Two String :"+ a1 + a2);
                }
                public void Add(int a1, int a2)
                {
                    Console.WriteLine("Adding Two Integer :"+ a1 + a2);
                }
            }
            static void Main(string[] args)
            {
                TestOverloading obj = new TestOverloading();
                obj.Add("Manish ", "Agrahari");
                obj.Add(5, 10);
                Console.ReadLine();
            }
        }
    }
}

```

Constructor Overloading: The process of creating more than one constructor with same name, which is similar to class name, but with different parameters is called constructor overloading.

Example:
 using System;
 namespace ConsoleApplication3
 {
 class Sample
 {

```

public string param1, param2;
public Sample() // Default Constructor
{
    param1 = "Hi";
    param2 = "I am Default Constructor";
}
public Sample(string x, string y)
// Declaring Parameterized constructor with Parameters
{
    param1 = x;
    param2 = y;
}
}
class Program
{
    static void Main(string[] args)
    {
        Sample obj= new Sample(); // Default Constructor will Called
        Sample obj1=new Sample("Welcome","Aspdotnet-Suresh");
        // Parameterized Constructor will Called
        Console.WriteLine(obj.param1 + ", "+obj.param2);
        Console.WriteLine(obj1.param1 +" to " + obj1.param2);
        Console.ReadLine();
    }
}

```

Output: Hi, I am Default Constructor
Welcome to Aspdotnet-Suresh

OPERATOR OVERLOADING: This is something very much similar to the concept of method overloading. Operator overloading allows us to define multiple behaviours to an operator. In operator overloading the behaviour of an operator changes according to the operands types which we use in an expression.

For example '+' is an overloaded operator, which can be used for both addition as well as concatenation. It works as addition operator when the operator used between numeric operands and works as concatenation operator when the operator used between string operands.

We can also add new behaviour to any existing operators by defining operator method.

Syntax: `public static returntype operator op(argumentslist)`
`{`
Method body
`}`

Where,

- They must be defined as **public and static**.
- **Return type** specifies the type of result we are expecting when the operator is used between two operands..

- **Operator** is a keyword which represents the operator method for overloading.
- **Op** is an any overloadable operator.
- The **argument list** is the list of arguments passed.

Example:

Unary minus: Public static bank operator -(bank b)

```
{
    Method body
}
```

Vector addition (Binary operator): Public static Vector operator +(Vector x ,Vector y)

```
{
    Method body
}
```

Comparison Operator: Public Static Vector operator ==(Vector x,Vector y)

```
{
    Method body
}
```

Overloadable Operators:

Operators that can be overloaded in c# as shown below:

Category	Operators
Binary arithmetic	+, -, *, /, %
Logical operator	==, !=, >=, <=, >, <
Unary arithmetic	+, ++, --
Binary Bitwise	~, &, ^, >>, <<
Unary Bitwise	~, !, true, false

Non Overloadable Operators:

Operators that can be overloaded in C# as shown below:

Category	Operators
Conditional operators	, &&
Compound Assignment	-=, +=, *=, %=
Other	(), [], =, ?:, ->, new, typeof, sizeof, as, is

Unary operator overloading: In case of unary operators, the argument must be the same type as that of enclosing class or struct

```
Example:      using System;
              class bank
              {
                  int x;
                  int y;
                  public bank(int a, int b)
                  {
                      x = a;
                      y = b;
                  }
                  public bank()
                  {
                  }
              }
```

```

        public void display()
        {
            Console.Write(" " + x);
            Console.Write(" " + y);
            Console.WriteLine();
        }
        public static bank operator -(bank b)
        {
            b.x = -b.x;
            b.y = -b.y;
            return b;
        }
    }
}
class program
{
    public static void Main()
    {
        bank ba1 = new bank(10,-20);
        ba1.display();
        bank ba2 = new bank();
        ba2.display();
        ba2 = -ba1;
        ba2.display();
        Console.ReadLine();
    }
}

```

Output:
 10 -20
 0 0
 -10 20

Example:

```

using System;
namespace OperatorOvlApplication
{
    class calculation
    {
        int a, b, c;
        public calculation(int x, int y, int z)
        {
            a = x;
            b = y;
            c = z;
        }
        public static calculation operator ++(calculation op1)
        {
            op1.a++;
            op1.b++;
            op1.c++;
            return op1;
        }
    }
}

```

```

        public void ShowTheResult()
        {
            Console.WriteLine(a + "," + b + "," + c);
            Console.ReadLine();
        }
    }
class Program
{
    static void Main(string[] args)
    {
        calculation i = new calculation(10, 20, 30);
        i++;
        i.ShowTheResult();
        Console.WriteLine();
    }
}

```

Output: 11,21,31

Binary operator overloading: In case of binary operators, the argument must be the same type as that of enclosing class or struct and second may be of any type.

Example:

```

using System;
namespace binary_overload
{
    class complexNumber
    {
        int x;
        double y;
        public complexNumber(int real, double imaginary)
        {
            x = real;
            y = imaginary;
        }
        public complexNumber()
        {
        }
        public static complexNumber operator +(complexNumber c1,
            complexNumber c2)
        {
            complexNumber c = new complexNumber();
            c.x=c1.x+c2.x;
            c.y=c1.x-c2.y;
            return c;
        }
        public void show()
        {
            Console.Write(x);
            Console.Write("+j"+y);
            Console.WriteLine();
        }
    }
}

```

```

    }
    class Program
    {
    static void Main(string[] args)
    {
        complexNumber p, q, r;
        p = new complexNumber(10, 2.0);
        q = new complexNumber(20, 15.5);
        r = p + q;
        Console.Write("p=");
        p.show();
        Console.Write("q=");
        q.show();
        Console.Write("r=");
        r.show();
        Console.ReadLine();
    }
    }
}

```

Output:

```

p=10+j2
Q=20+j15.5
R=30+j-5.5

```

Example:

```

using System;
namespace OperatorOvlApplication
{
    class Box
    {
        private double length;    // Length of a box
        private double breadth;   // Breadth of a box
        private double height;    // Height of a box
        public double getVolume()
        {
            return length * breadth * height;
        }
        public void setLength( double len )
        {
            length = len;
        }
        public void setBreadth( double bre )
        {
            breadth = bre;
        }
        public void setHeight( double hei )
        {
            height = hei;
        }
        // Overload + operator to add two Box objects.
        public static Box operator+ (Box b, Box c)
        {

```

```

        Box box = new Box();
        box.length = b.length + c.length;
        box.breadth = b.breadth + c.breadth;
        box.height = b.height + c.height;
        return box;
    }
}
class Tester
{
    static void Main(string[] args)
    {
        Box Box1 = new Box(); // Declare Box1 of type Box
        Box Box2 = new Box(); // Declare Box2 of type Box
        Box Box3 = new Box(); // Declare Box3 of type Box
        double volume = 0.0; // Store the volume of a box here
        // box 1 specification
        Box1.setLength(6.0);
        Box1.setBreadth(7.0);
        Box1.setHeight(5.0);
        // box 2 specification
        Box2.setLength(12.0);
        Box2.setBreadth(13.0);
        Box2.setHeight(10.0);
        // volume of box 1
        volume = Box1.getVolume();
        Console.WriteLine("Volume of Box1 : {0}", volume);
        // volume of box 2
        volume = Box2.getVolume();
        Console.WriteLine("Volume of Box2 : {0}", volume);
        // Add two object as follows:
        Box3 = Box1 + Box2;
        // volume of box 3
        volume = Box3.getVolume();
        Console.WriteLine("Volume of Box3 : {0}", volume);
        Console.ReadKey();
    }
}
}

```

Output: Volume of Box1 : 210
 Volume of Box2 : 1560
 Volume of Box3 : 5400

Comparison operator overloading: There are six comparison operators that can be considered in three pairs:

- == and !=
- < and <=
- and >=

Example: using System;
 namespace comparison
 {

```

class Vector
{
    int x, y, z;
    public Vector(int p, int q, int r)
    {
        x = p;
        y = q;
        z = r;
    }
    public static bool operator ==(Vector v1, Vector v2)
    {
        if (v1.x == v2.x && v1.y == v2.y && v1.z == v2.z)
            return (true);
        else
            return (false);
    }
    public static bool operator !=(Vector v1, Vector v2)
    {
        return (!(v1 == v2));
    }
}
class comparison
{
    static void Main()
    {
        Vector v1 = new Vector(10, 20, 30);
        Vector v2 = new Vector(40, 50, 60);
        if (v1 == v2)
            Console.WriteLine("v1 and v2 both are Equal");
        else
            Console.WriteLine("v1 and v2 are not equal");
        if (!(v1 == v2))
            Console.WriteLine("true");
        else
            Console.WriteLine("false");
        Console.ReadLine();
    }
}

```

Output: v1 and v2 are not equal

Dynamic Polymorphism (or) Run time polymorphism (or) late binding (or) method overriding:

The process of re-implementing the parent class method under the child class with same name and same signature is called method overriding. Method overriding means same method names with same signatures in different classes. In this run time polymorphism or method overriding we can override a method in base class by creating similar function in derived class this can be achieved by using inheritance principle and using “**virtual & override**” keywords. In base class if we declare methods with **virtual** keyword then only we can override those methods in derived class

using **override** keyword.

Syntax: class A

```

{
    public virtual void show()
    {
        ---
        ---
    }
}
class B:A
{
    public override void show()
    {
        ----
        ----
    }
}

```

Notes:

1. Overriding virtual method of parent class under child class is only optional for child classes.
2. When a method overrides under the child class. The object of the child class gives the preference to local method and invokes the method it has overridden, but not the virtual method of parent.
3. Using the **base** keyword we can invoke the virtual method of the parent class from child class after overriding. Using the base keyword in the static blocks is not possible.

Example: using System;
namespace OperatorOvlApplication

```

{
    public class Bclass
    {
        public virtual void Sample1()
        {
            Console.WriteLine("Base Class");
        }
    }
    // Derived Class
    public class DClass : Bclass
    {
        public override void Sample1()
        {
            Console.WriteLine("Derived Class");
        }
    }
    // Using base and derived class
    class Program
    {

```

```

        static void Main(string[] args)
        {
            // calling the overridden method
            DClass objDc = new DClass();
            objDc.Sample1();
            // calling the base class method
            Bclass objBc = new BClass();
            objBc.Sample1();
        }
    }
}

```

Output: Derived Class
Base Class

Example:

```

using System;
namespace methodoverriding
{
    class Program
    {
        static void Main(string[] args)
        {
            employee obj = new employee();
            obj.display();
            Console.ReadLine();
        }
    }
    public class cls
    {
        public virtual void display()
        {
            Console.WriteLine("hello");
        }
    }
    public class employee : cls
    {
        public override void display()
        {
            Console.WriteLine("welcome");
        }
    }
}

```

Example:

```

using System;
namespace PolymorphismApplication
{
    class super
    {
        protected int x;
        public super( int x)
        {

```

```

        this.x=x;
    }
    public virtual void display()
    {
        Console.WriteLine("super x="+x);
    }
}
class sub: super
{
    int y;
    public sub( int x, int y): base(x)
    {
        this.y=y;
    }
    public override void display ()
    {
        Console.WriteLine("super x=" +x);
        Console.WriteLine("super y=" +y);
    }
}
class Tester
{
    static void Main(string[] args)
    {
        sub s1= new sub(100,200);
        s1.display();
        Console.ReadKey();
    }
}

```

Output:
 super x=100
 Sub y=200

DIFFERENCES BETWEEN OVERLOADING AND OVERRIDING:

Overloading	Overriding
It is implementing multiple method with same name and different signature	It is implementing multiple method with same name and same signature
This can be performed in one single class as parent/child classes also.	This can be performed only in child class
To overload a parent class method under child class we do not need any permission from parent	To override a parent class method under child class we require an explicit permission from parent

This is all about providing multiple behaviour to a method.	This is all about changing the behaviour of the method.
---	---

METHOD HIDING (OR) SHADOWING: This is another approach for re-implementing a parent class method under child class even if they are not declared as virtual that re-implementation is performed without parent class permission.

Syntax: class A

```

{
    public void show()
    {
        ---
        ---
    }
}
class B:A
{
    public new void show()
    {
        ----
        ----
    }
}

```

Example:

```

using System;
namespace App
{
    class base
    {
        public void display()
        {
            Console.WriteLine("base method");
        }
    }
    class derived:base
    {
        public new void display()
        {
            Console.WriteLine("derived method");
        }
    }
    class test
    {
        public static void Main()
        {
            derived d=new derived();
            d.display();
        }
    }
}

```

Output: derived method

Note: we can hide an inherited member using new, new does not remove the member, its only makes the member inaccessible in the derived class.

ABSTRACT CLASS: In c sharp Abstract class is defined using "abstract" keyword. When a class contains at least one abstract method, then the class must be declared as **abstract class**. If an Abstract class contain any abstract methods, then those methods must be implemented under the child class using override modifier.

Characteristics of abstract classes:

1. Abstract Class cannot be instantiated directly.
2. Abstract Class can have both abstract methods and non-abstract methods.
3. If any child class of abstract class wants to consume non abstract method of its parent, first they require to implement all the abstract methods of parent otherwise consuming non abstract methods of parent will not be possible.
4. An abstract class can consume only by child classes that to after providing the implementation for all the abstract methods of abstract class.
5. You cannot declare an abstract method outside an abstract class
6. When a class is declared sealed, it cannot be inherited, abstract classes cannot be declared sealed.

Syntax: **abstract class class1**
 {
 public abstract void add(int x, int y);
 }

ABSTRACT METHOD: A method without method body is known as abstract method. It contains only declaration of the method. We can define abstract method using abstract keyword.

Characteristics of abstract methods:

1. Abstract method cannot have implementation.
2. Abstract method implementation must be provided in non-abstract derived classes by overriding the method using override modifier.
3. Abstract method can be declared only in Abstract classes.
4. Abstract method cannot take static and virtual modifiers.

Example: using System;
 namespace ConsoleApplication8
 {
 abstract class parent
 {
 public void add(int x, int y)
 {
 } } }

```

        Console.WriteLine(x + y);
    }
    public void sub(int x, int y)
    {
        Console.WriteLine(x - y);
    }
    public abstract void mul(int x, int y);
    public abstract void div(int x, int y);
}
class child : parent
{
    public override void mul(int x, int y)
    {
        Console.WriteLine(x * y);
    }
    public override void div(int x,int y)
    {
        Console.WriteLine(x / y);
    }
}
class test
{
    static void Main(string[] args)
    {
        child c = new child();
        c.add(10, 20);
        c.sub(20, 10);
        c.mul(10, 10);
        c.div(20, 10);
    }
}

```

Example:

```

using System;
namespace PolymorphismApplication
{
    abstract class Shape
    {
        public abstract int area();
    }
    class Rectangle:Shape
    {
        private int length;
        private int width;
        public Rectangle( int a, int b)
        {
            length = a;
            width = b;
        }
        public override int area ()
    }
}

```

```

        {
            Console.WriteLine("Rectangle class area :");
            return (width * length);
        }
    }
class RectangleTester
{
    static void Main(string[] args)
    {
        Rectangle r = new Rectangle(10, 7);
        double a = r.area();
        Console.WriteLine("Area: {0}",a);
        Console.ReadKey();
    }
}
}

```

Output: Rectangle class area: Area: 70

SEALED CLASSES (PREVENTING INHERITANCE): A class that cannot be subclassed is called sealed class. Sealed classes can be created using sealed modifier.

Some points to remember:

- a class, which restricts inheritance for security reason is declared, sealed class.
- Sealed class is the last class in the hierarchy.
- Sealed class can be a derived class but can't be a base class.
- A sealed class cannot also be an abstract class. Because abstract class has to provide functionality and here we are restricting it to inherit.

Syntax: accessmodifier sealed class classname

```

{
    -----
    -----
}

```

Example: class1

```

{
    public virtual void show()
    {
        ----
        ----
    }
}
class2
{
    public override void show() //valid
    {
        ----
        ----
    }
}
Class3
{

```

```

        public override void show()    //valid
        {
            ----
            ----
        }
    }
Example: class1
    {
        public virtual void show()
        {
            ----
            ----
        }
    }
    class2
    {
        public sealed override void show()    //valid
        {
            ----
            ----
        }
    }
Class3
    {
        public override void show()    //invalid
        {
            ----
            ----
        }
    }
Example: using System;
namespace sealed_class
{
    class Program
    {
        public sealed class BaseClass
        {
            public void Display()
            {
                Console.WriteLine("This is a sealed class which
                can;t be further inherited");
            }
        }
        //public class Derived : BaseClass
        // {
        // this Derived class can;t inherit BaseClass because it is sealed
        //}
        static void Main(string[] args)
        {

```

```

        BaseClass obj = new BaseClass();
        obj.Display();
        Console.ReadLine();
    }
}
}

```

SEALED METHOD: when an instance method declaration includes the sealed modifier, the method is called sealed method. It means a derived class cannot be override this method. Sealed keyword is always used with override keyword.

Syntax: **access_modifier sealed override returntype methodname(Params_list)**
 {

 }

Example:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace sealed_method
{
    class Program
    {
        public class BaseClass
        {
            public virtual void Display()
            {
                Console.WriteLine("Virtual method");
            }
        }
        public class DerivedClass : BaseClass
        {
            // Now the display method have been sealed and can;t be overridden
            public override sealed void Display()
            {
                Console.WriteLine("Sealed method");
            }
        }
        //public class ThirdClass: DerivedClass
        //{
            //public override void Display()
            //{
                //Console.WriteLine("Here we try again
                //to override display method which is
                //not possible and will give error");
            //}
        //}
        static void Main(string[] args)
        {

```

```

        DerivedClass ob1 = new DerivedClass();
        ob1.Display();
        Console.ReadLine();
    }
}

```

Output: sealed method

INTERFACES: An interface is a collection of declaration of the members that may be implemented by a given class.

1. Interface is a pure protocol. i.e.
 - a. It never defines data-type
 - b. It never provides a default implementation of the methods.
2. Interface never specifies as a base class
3. Never contains member that do not take an access-modifier (as all interface-members are implicitly public).

Example:

```

public interface IAmBadInterface
{
    // Error, interfaces can't define data!
    int myInt = 0;
    // Error, only abstract members allowed!
    void MyMethod()
    {
        Console.WriteLine("Hi!");
    }
}

```

Declaring Interfaces: Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default.

Declaration:

```

interface interfacename
{
    Method declarations
}

```

Where,

Interface - It is the keyword

Interfacename- It is valid C# identifier.

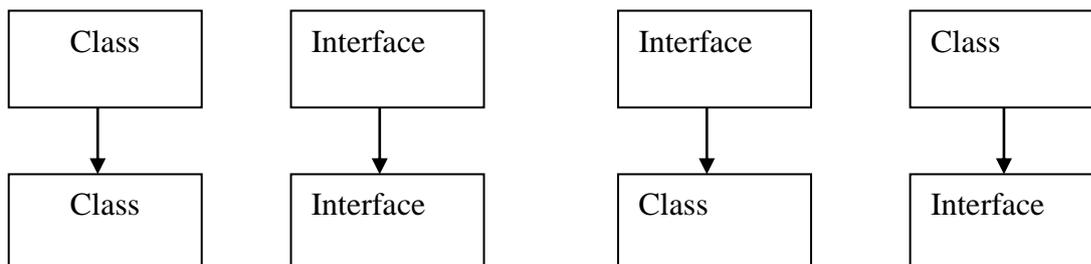
Method declarations- It will contain only list of methods without implementation.

Example:

```

public interface ITransactions
{
    // interface members
    void showTransaction();
    double getAmount();
}

```



Valid **Valid** **Valid** **InValid**
Example: The following example demonstrates implementation of the above interface:

```

using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace InterfaceApplication
{
    public interface ITransactions
    {
        // interface members
        void showTransaction();
        double getAmount();
    }
    public class Transaction : ITransactions
    {
        private string tCode;
        private string date;
        private double amount;
        public Transaction()
        {
            tCode = " ";
            date = " ";
            amount = 0.0;
        }
        public Transaction(string c, string d, double a)
        {
            tCode = c;
            date = d;
            amount = a;
        }
        public double getAmount()
        {
            return amount;
        }
        public void showTransaction()
        {
            Console.WriteLine("Transaction: {0}", tCode);
            Console.WriteLine("Date: {0}", date);
            Console.WriteLine("Amount: {0}", getAmount());
        }
    }
}
class Tester
{
    static void Main(string[] args)
    {
        Transaction t1=new Transaction("001", "8/10/2012", 7890.00);
        Transaction t2=new Transaction("002", "9/10/2012", 45190.00);
        t1.showTransaction();
        t2.showTransaction();
    }
}

```

```

        Console.ReadKey();
    }
}

```

Output:
 Transaction: 001
 Date: 8/10/2012
 Amount: 78900
 Transaction: 002
 Date: 9/10/2012
 Amount: 451900

Multiple Inheritance using interfaces: C# does not support multiple inheritance. However, you can use interfaces to implement multiple inheritance. The following program demonstrates this:

```

using System;
namespace InheritanceApplication
{
    class Shape
    {
        public void setWidth(int w)
        {
            width = w;
        }
        public void setHeight(int h)
        {
            height = h;
        }
        protected int width;
        protected int height;
    }
    public interface PaintCost
    {
        int getCost(int area);
    }
    class Rectangle : Shape, PaintCost
    {
        public int getArea()
        {
            return (width * height);
        }
        public int getCost(int area)
        {
            return area * 70;
        }
    }
    class RectangleTester
    {
        static void Main(string[] args)
        {
            Rectangle Rect = new Rectangle();

```

```

        int area;
        Rect.setWidth(5);
        Rect.setHeight(7);
        area = Rect.getArea();
        Console.WriteLine("Total area: {0}",
            Rect.getArea());
        Console.Write("Total paint cost: ${0} ", Rect.getCost(area));
        Console.ReadKey();
    }
}
}

```

Output: Total area: 35
Total paint cost: \$2450

PARTIAL CLASS: partial classes are newly introduced in C# 2.0.

A class which code can be written in multiple locations with same name is called partial classes. We can define partial classes using partial keyword.

Or

The process of splitting the definition of a class or a struct, or an interface over two or more source files. Each source file contains a section of the class definition, and all parts are combined when the application is compiled. To define a class as partial in all the files, the class name must be same and also we need to use partial modifier to specify the class is partial.

- Partial classes will be merged together at runtime and works like a single class.
- Partial classes allow multiple programmers to work on the same class at same time.
- Partial classes are only an approach of physically division of code in all the files is treated as single class only.

Syntax:

```

partial class test
{
    public void m1()
    {
        ---
        ---
    }
}
partial class test
{
    public void m2()
    {
        ---
        ---
    }
}

```

Example: using System;
namespace App

```
{
    public partial class partialclass
    {
        private int x;
        private int y;
        public partialclass(int x,int y)
        {
            this.x=x;
            this.y=y;
        }
    }
    public partial class partialclass
    {
        public void print()
        {
            Console.WriteLine("output values {0},{1}", x, y);
        }
    }
}
class test
{
    public static void Main()
    {
        partialclass p=new partialclass(10,15);
        p.print();
        Console.ReadLine();
    }
}
```

Output: output values 10, 15

Note: we use partial classes in windows application development.

WHY DELEGATES: Delegates are used in the following cases:

- Delegates can be used to handle (call/invoke) multiple methods on a single event.
- Delegates can be used to define call-back (asynchronous) methods.
- Delegates can be used for decoupling and implementing generic behaviours.
- Delegates can be invoked method at runtime.
- Delegates can be used in LINQ for parsing the **Expression Tree**.
- Delegates can be used in different Design Pattern.

DELEGATES:

Definition: A delegate object is a special type of object that contains the details of a method rather than data.

Or

Delegate is a class type object and is used to invoke a method that has been encapsulated into it at the time of its creation.

Or

A delegate (known as function pointer in C/C++) is a references type that invokes single/multiple method(s) through the delegate instance.

Or

Delegate is an object, which can be used to invoke the functionality of other object without knowing its class name or method name.

- In c# delegates means a method acting for another method.
- A delegate is a reference type variable that holds the reference to a method. The reference can be changed at runtime
- Delegate types are sealed and immutable type.
- All delegates are implicitly derived from the System.Delegate class
- Delegates are especially used for implementing events and the call-back methods.

Delegate declaration: Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to a method, which has the same signature as that of the delegate.

Example: **public delegate int MyDelegate (string s);**

The preceding delegate can be used to reference any method that has a single string parameter and returns an int type variable.

Syntax: **modifier delegate return type delegate-name (parameter list);**

Where,

Delegate: It is a keyword that specifies that the declaration represents a class type derived from System.Delegate.

Return type: It indicates the return type of the delegate.

Parameter list: It identifies the signature of the delegate.

Delegate-name: It is any valid C# identifier and is the name of the delegate that will be used to call delegate objects.

Modifier: It controls the accessibility of the delegate. It is optional depending upon the context in which they are declared, delegate may take any of the following modifiers.

- new
- public
- protected
- internal
- private

The new modifier is only permitted on delegates declared within another type. It signifies that the delegate hides an inherited member by the by the same name.

Examples:

```
delegate void sampledelegate ();
delegate int mathoperation (int x, int y);
public delegate int compareitems (object b,object b1);
private delegate string getastring ();
```

The delegate may be defined in the following places

- inside a class
- outside all classes
- as the top level object in a namespace

Delegates are implicitly sealed and therefore it is not possible to derive any type from a delegate type.

Delegate methods: the methods whose references are encapsulated into a delegate instance are known as delegate methods (or) callable entities. The return type and signature of delegate methods must exactly match the signature and return type of the delegate. Delegate methods do not care about

- What type of object the method is being called against.
- Whether the method is a static or an instance method.

Example: `delegate string getastring ();`

The above delegate can made to refer to the method ToString() using an int object N as follows

```
int N=100;
getastring s1=new getastring(N.ToString);
```

Example: `delegate void delegate1 ();`

The above delegate can encapsulate references to the following methods

```
public void F1()
{
    Console.WriteLine("F1");
}
static public void F2()
{
    Console.WriteLine("F2");
}
```

Example: `delegate double mathop (double x, double y);`

The above delegate can refer any one of the following methods.

```
public static double multiply(double a, double b)
{
    return (a*b);
}
public double divide(double a, double b)
{
    return (a/b);
}
```

}

Note: In example2 and example3, the signature and return type of methods match the signature and type of the delegate.

Delegate instantiation: the syntax for delegate instantiation is

Syntax: new delegate-type (expression)

Where,

Delegate-type: It is the name of the delegate declared earlier whose object is to be created.

Expression: It must be a method name or a value of a delegate type. If it is a method name its signature and return type must be the same as those of the delegate. If no matching method exists, or more than one matching method exists, an error occurs. The matching method may be either an instance method or a static method. If it is an instance method, we need to specify the instance as well as the name of the method. If it is a static one, then it is enough to specify the class name and method name.

Example:

```

delegate int productdelegate(int x,int y);
class delegate
{
    static float product(float a, float b) //signature does not match
    {
        return (a*b);
    }
    static int product(int a,int b) //signature matches
    {
        return (a*b);
    }
    productdelegate p=new productdelegate(product);
}

```

Here, we have two methods with same name but with different signatures. The delegate p is initialized with the reference to the second product method because that method exactly matches the signature and return type of productdelegate. If this method is not present, an error will occur.

Note: The method and instantiation statement are within the same class, we simply use the method name for creating the instance.

Example:

```

delegate void displaydelegate();
class A
{
    public void displayA()
    {
        Console.WriteLine("DisplayA");
    }
}
class B
{
    static public void display()

```

```

        {
            Console.WriteLine("DisplayB");
        }
    }
}

```

```

-----
-----

```

```

A a=new A();
displaydelegate d=new displaydelegate(a.displayA);
displaydelegate d1=new displaydelegate(B.displayB);

```

The above code defines two delegate methods in two different classes. Since class A defines an instance method, an A type object is created and used with the method name to initialize the delegate object d. The delegate method defined in class B is static and therefore the class name is used directly with the method name in creating the delegate object d1.

Delegate invocation: C# uses a special syntax for invoking a delegate. When a delegate is invoked, it in turn invokes the method whose reference has been encapsulated into the delegate (only if their signatures match).

Syntax: delegate_object (parameter list);

The parameters list provides values for the parameters of the method to be used

1. if the invocation invokes a method that returns void, the result is nothing and therefore it cannot be used as an operand of any operator. It can be simply a statement_expression.

Example: `delegate1(x,y);` //void delegate: this delegate invokes a method that does not return any value

2. If the method returns a value, then it can be used as an operand of any operator. Usually, we assign the return value to an appropriate variable for further processing

Example: `double result=delegate2 (2.56, 45.63);` //this statement invokes a method (that takes two double values as parameters and returns double type value) and then assigns the returned value to the variable result.

Types of Delegates: There are three types of delegates that can be used in C#.

- Single Delegate
- Multicast Delegate

Single Delegate: single cast delegate can point to a single method at a time and used to invoke a single method.

Example: demonstrates declaration, instantiation, and use of a delegate that can be used to reference methods that take an integer parameter and returns an integer value.

```

using System;
delegate int NumberChanger(int n);
namespace DelegateAppl

```

```

    {
        class TestDelegate
        {
            static int num = 10;
            public static int AddNum(int p)
            {
                num += p;
                return num;
            }
            public static int MultNum(int q)
            {
                num *= q;
                return num;
            }
            public static int getNum()
            {
                return num;
            }
            static void Main(string[] args)
            {
                //create delegate instances
                NumberChanger nc1 = new NumberChanger(AddNum);
                NumberChanger nc2 = new NumberChanger(MultNum);
                //calling the methods using the delegate objects
                nc1(25);
                Console.WriteLine("Value of Num: {0}", getNum());
                nc2(5);
                Console.WriteLine("Value of Num: {0}", getNum());
                Console.ReadKey();
            }
        }
    }

```

Output: Value of Num: 35

Value of Num: 175

Multicast Delegate (or) combinable delegates: Multicast delegate can point multiple methods in order to invoke multiple methods at the time of invocation.

Or

A collection of single cast delegates are called multi cast delegate.

Multicast delegates must specify the following conditions.

- The return type of the delegate must be void
- None of the parameters of the delegate type can be declared as output parameters, using out keyword.

If d is the delegate that satisfies the above conditions and d1, d2, d3 and d4 are the instances of d then the statements.

d3=d1+d2; //d3 refers two methods

d4=d3-d2; // d4 refers only d1 method

The multicast delegates uses arithmetic operators such as + and -. + is used to add a method into sequence and – is used to delete a method from sequence.

Example: using System;

```

delegate void mdelegate();
namespace DelegateAppl
{
    class dm
    {
        static public void display()
        {
            Console.WriteLine("new delhi");
        }
        static public void print()
        {
            Console.WriteLine("new york");
        }
    }
    class mtest
    {
        public static void Main()
        {
            mdelegate m1=new mdelegate(dm.display);
            mdelegate m2=new mdelegate(dm.print);
            mdelegate m3=m1+m2;
            mdelegate m4=m2+m1;
            mdelegate m5=m3-m2;
            m3();
            m4();
            m5();
        }
    }
}

```

Output:
new delhi
new york
new york
new delhi
new delhi

Example:

```

using System;
delegate int NumberChanger(int n);
namespace DelegateAppl
{
    class TestDelegate
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }
        public static int MultNum(int q)
        {
            num *= q;
        }
    }
}

```

```

        return num;
    }
    public static int getNum()
    {
        return num;
    }
    static void Main(string[] args)
    {
        //create delegate instances
        NumberChanger nc;
        NumberChanger nc1 = new NumberChanger(AddNum);
        NumberChanger nc2 = new NumberChanger(MultNum);
        nc = nc1;
        nc += nc2;           //calling multicast
        nc(5);
        Console.WriteLine("Value of Num: {0}", getNum());
        Console.ReadKey();
    }
}
}
}

```

Output: Value of Num: 75

Errors: Debugging is the process of identifying and fixing errors in a software program. In software development domain, such errors are called bugs.

Types of errors:

Compile time error: An error that occurs in a program during the compilation of a program is called

Compile time errors. These types of errors occur due to syntax mistakes under the program. All syntax errors will be detected and displayed by the c# compiler and therefore these errors are known as compile time errors. Whenever the compiler displays an error, it will not create the .cs file.

Example:

```

/* this program contained an error */
using Sytem;
class Error1
{
    static void Main()
    {
        Console.Write("hello c#");
    }
}

```

In the above program we have misspelled the System namespace; the compiler will display the following message.

Error1.cs (2.7): error CS0234: the type or namespace System does not exist in the class or namespace.

Most of the compile time errors are due to typing mistakes. Typographical errors are hard to find. We may have to check word by word or even character by character. The most common compile time errors are

1. Missing semicolons.
2. Missing brackets in classes and methods.
3. Misspelling of identifiers and keywords.
4. Missing double quotes in strings.
5. Use of undeclared variables.
6. Incompatible types in assignments/initialization.
7. Bad references to objects.
8. Use of = in place of == operator.

Run time errors: An error that occurs in a program during the execution of the program is called Run time errors. Sometimes, a program may compile successfully creating the .exe file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow. Most common runtime errors are

1. Dividing an integer by zero.
2. Accessing an element that is out of the bounds of an array.
3. Trying to store a value into an array of an incompatible type or class.
4. Passing a parameter that is not in a valid range or value for a method.
5. Attempting to use negative size for an array.
6. Using a null object reference as a legitimate object reference to access method or a variable.
7. Converting an invalid string to a number or vice versa.
8. Accessing a character that is out of bounds of a string.

When such errors are encountered, C# typically generates an error message and aborts the program.

Example:

```
using System;
class error2
{
    static void Main()
    {
        int a=10;
        int b=5;
        int c=5;
        int x=a/(b-c);
        Console.WriteLine("x="+x);
        int y=a/(b+c);
        Console.WriteLine("y="+y);
    }
}
```

The above program is syntactically correct and therefore does not cause any problem during compilation. When C# run time tries to execute a division by zero, it generates an error condition which causes the program stop after an appropriate message. The following statement is never

executed: `int y=a/(b+c);`

EXCEPTION HANDLING: An exception is a condition that is caused by a runtime error in the program. When the c# compiler encounters an error such as dividing an integer by zero, it creates an exception object and throws it(that is informs us that an error has occurred).

If the exception object not caught and handled properly, the compiler will display an error message and will terminate the program. If we want the program to continue with the execution of remaining code, then we should try to catch exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.

C# exception handling is built upon four keywords: **try, catch, finally, and throw.**

Syntax for exception handling: The basic concepts of exception handling are throwing an exception and catching it.

try: whenever an exception is expected from a block of statements, then write those statements in try block..

catch: A catch block defined by the keyword catch. A catch block catches the exception thrown by the try block and handles it appropriately. The catch block is added immediately after the try block.

Syntax:

```
try
{
    Statements; // generates an exception
}
catch (Exception e)
{
    Statements; // process the exception
}
```

The try block can have one or more statements that could generate an exception. If anyone statement generates an exception, the remaining statements in the block are skipped and execution jumps to the catch block that is placed next to the try block. The catch block also can have one or more statements that are necessary to process the exception (remember every try statement should be followed by at least one catch statement; otherwise compilation error will occur. The catch statement is passed a single parameter, which is the reference to the exception object thrown by the try block. If the catch parameter matches with the type of exception object, then the exception is caught and statements in the catch block will be executed. Otherwise, the exception is not caught and the default exception handler will cause the execution to terminate.

Example:

```
using System;
class error3
{
    public static void Main()
    {
```

```

        int a=10;
        int b=5;
        int c=5;
        int x,y;
        try
        {
            x=a/(b-c);
        }
        catch(Exception e)
        {
            Console.WriteLine("division by zero");
        }
        y=a/(b+c);
    }
}

```

Output: division by zero
y=1

Note that program did not stop at the point of exceptional condition. It catches the error condition, prints the error message and then continues the execution as if nothing has happened.

Multiple catch statements: it is possible to have more than one catch statement in the catch block.

```

try
{
    Statements; // generates an exception
}
catch(Exception-type1 e)
{
    Statements; // process the exception1
}
catch(Exception-type2 e)
{
    Statements; // process the exception2
}
.
.
.
catch(Exception-typeN e)
{
    Statements; // process the exception N
}

```

When an exception in the try block is generated, the *c#* treats the multiple catch statements like cases in a switch statement. The first statement whose parameter matches with the exception object will be executed, and the remaining statements will be skipped.

Note: *C#* does not require any processing of the exception at all. We can simply have a catch statement with an empty block to avoid the program abortion.

Example: `catch(Exception e){}`

The above statement will catch an exception and then ignore it.

Example: `using System;`

```
class error3
{
    public static void Main()
    {
        int[] a={5,10};
        int b=5;
        try
        {
            int x=a[2]/b-a[1];
        }
        catch(ArithmeticException e)
        {
            Console.WriteLine("division by zero");
        }
        catch(IndexOutOfRangeException e)
        {
            Console.WriteLine("array index error");
        }
        catch(ArithmeticException e)
        {
            Console.WriteLine("division by zero");
        }
        catch(ArrayTypeMismatchException e)
        {
            Console.WriteLine("wrong data type");
        }
        int y=a[1]/a[0];
        Console.WriteLine("y="+y);
    }
}
```

Output: `array index error`

`y=2`

Exception Hierarchy: All exceptions are derived from the class **Exception**. When an exception occurs, the proper catch handler is determined by matching the type of exception to the name of the exception mentioned. If we are going to catch exceptions at different levels in the hierarchy, we need to put them in the right order. The rule is that we must always put the handlers for the most derived exception class first.

Example: `try`

```
{
    // throws divide by zero exception
}
catch(Exception e)
{
    -----
}
```

```

        -----
    }
    catch(DivideByZeroException e)
    {
        -----
        -----
    }

```

The above code will generate a compile time error, because the exception is caught by the first catch (which is more general one) and the second catch is therefore unreachable. In C#, having unreachable code is always an error

Example:

```

try
{
    // throws divide by zero exception
}
catch(DivideByZeroException e)
{
    -----
    -----
}
catch(Exception e)
{
    -----
    -----
}

```

The order of catch blocks is important. We must start with catch blocks that are designed to trap very specific exceptions and finish with more general blocks that will cover any other exceptions for which we have not provided handlers.

General catch handler: A catch block will catch any exception is called a general catch handler. A general catch handler does not specify any parameter.

Syntax:

```

try
{
    -----//causes an exception
}
catch
{
    ----- // handles an error
    -----
}

```

Note that catch (Exception e) can handle all the exceptions thrown by the C# code and therefore can be used as a general catch handler. However, if the program uses libraries written in other languages, then there may be an exception that is not derived from the class Exception. Such exceptions can be handled by parameter-less catch statement. This handler always placed at the end. Since there is no parameter, it does not catch any information about the exception and therefore we do not know what

went wrong.

Using finally statement: C# supports another statement known as a finally statement that can be used to handle an exception that is not caught by any of the previous catch statements. A finally block can be used to handle any exception generated within a try block. It may be added immediately after the try block or after the last catch block.

Syntax:

```

try
{
    ----
    ----
}
finally
{
    ----
    ----
}

```

Or

```

try
{
    ----
    ----
}
catch(.....)
{
    ----
    ----
}
catch(.....)
{
    ----
    ----
}
.
.
.
finally
{
    ----
    ----
}

```

When finally block is defined, the program is guaranteed to execute, regardless of how control leaves the try, whether it is due to normal termination, due to an exception occurring or due to a jump statement.

Example: using System;
namespace ErrorHandlingApplication

```

    {
        class DivNumbers
        {
            int result;
            DivNumbers()
            {
                result = 0;
            }
            public void division(int num1, int num2)
            {
                try
                {
                    result = num1 / num2;
                }
                catch (DivideByZeroException e)
                {
                    Console.WriteLine("Exception caught: {0}", e);
                }
                finally
                {
                    Console.WriteLine("Result: {0}", result);
                }
            }
            static void Main(string[] args)
            {
                DivNumbers d = new DivNumbers();
                d.division(25, 0);
                Console.ReadKey();
            }
        }
    }

```

Output: Exception caught: System.DivideByZeroException: Attempted to divide by zero.
Result: 0

Nested try blocks: C# permits us to nest try blocks inside each other.

Example:

```

try
{
    ----(point p1)
    ----
    try
    {
        ----(point p2)
        ----
    }
    catch
    {
        ----(point p3);
        ----
    }
}
finally

```

```

        {
            ----
            ----
        }
        -----(point p4)
        -----
    }
    catch
    {
        ----
        ----
    }
    finally
    {
        ----
        ----
    }
}

```

When nested try blocks are executed, the exceptions that are thrown at various points are handled as follows:

- The points p1 and p4 are outside the inner try block and therefore any exceptions thrown at these points will be handled by the catch in the outer block. The inner block is simply ignored.
- Any exception thrown at point p2 will be handled by the inner catch handler and the inner finally will be executed. The execution will continue at point p4 in the program.
- If there is no suitable catch handler to catch an exception thrown at p2, the control will leave the inner block (after executing the inner finally) and look for a suitable catch handler in the outer block. If a suitable one is found, then that handler is executed followed by the outer finally code. Remember, the code at point p4 will be skipped.
- If an exception is thrown at point p3, it is treated as if it had been thrown by the outer try block and, therefore, the control will immediately leave the inner block (of course, after executing the inner finally) and search for a suitable catch handler in the outer block.
- In case, a suitable catch handler is not found, then the system will terminate program execution with an appropriate message.

Example: using System;
class nestedtry

```

{
    static int m=10;
    static int n=0;
    static void division()
    {
        try
        {
            int k=m/n;
        }
        catch(ArgumentException e)
        {
            Console.WriteLine("caught an exception");
        }
        finally
        {

```

```

        Console.WriteLine("inside division method");
    }
}
public static void Main()
{
    try
    {
        division();
    }
    catch(DivideByZeroException e)
    {
        Console.WriteLine("caught an exception");
    }
    finally
    {
        Console.WriteLine("inside main method");
    }
}
}

```

Creating User-Defined Exceptions or throwing our own exceptions: You can also throw our own exceptions. We can do this by using the keyword `throw`. User-defined exception classes are derived from the `System.ApplicationException` class.

Syntax: `throw new Throwable_subclass;`

Examples: `throw new ArithmeticException;`
`throw new FormatException;`

Example:

```

using System;
namespace ConsoleApplication8
{
    class TestTemperature
    {
        static void Main(string[] args)
        {
            Temperature t = new Temperature();
            try
            {
                t.showTemp();
            }
            catch (TempIsZeroException e)
            {
                Console.WriteLine("TempIsZeroException: {0}", e.Message);
            }
            Console.ReadKey();
        }
    }
}
public class TempIsZeroException : ApplicationException
{

```

```

        public TempIsZeroException(string message): base(message)
        {
        }
    }
    public class Temperature
    {
        int temperature = 0;
        public void showTemp()
        {
            if (temperature == 0)
            {
                throw(new TempIsZeroException("ZeroTemperature
                                                found"));
            }
            else
            {
                Console.WriteLine("Temperature: {0}", temperature);
            }
        }
    }
}

```

Output: TempIsZeroException: ZeroTemperature found

Exception Classes in C#: C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **System.Exception** class. Some of the exception classes derived from The **System.Exception** class are the **System.ApplicationException** and **System.SystemException** classes. The **System.ApplicationException** class supports exceptions generated by application programs. Hence the exceptions defined by the programmers should derive from this class. The **System.SystemException** class is the base class for all predefined system exception. The following table provides some of the predefined exception classes derived from the **Sytem.SystemException** class:

Exception Class	Description
SystemException	A failed run time check; used as a base class for other exceptions
AccessException	Failure to access a type member, such as a method or field
ArgumentException	An argument to a method was invalid
ArgumentNullException	A null argument was passed to a method that does not accept it
ArgumentOutOfRangeException	Argument value is out of range
ArithmeticException	Arithmetic over or underflow has occurred
ArrayTypeMismatchException	Attempt to store the wrong type of object in an array
BadImageFormatException	Image is in the wrong format
CoreException	Base class for exceptions thrown by the runtime
System.IO.IOException	Handles I/O errors.
IndexOutOfRangeException	An array index is out of range.
DivideByZeroException	Handles errors generated from dividing a dividend with zero

InvalidCastException	Handles errors typecasting
InvalidOperationException	A method was called at an invalid time
MissingMemberException	An invalid version of a DLL was accessed
NotFiniteNumberException	A number is not valid
NotSupportedException	Indicates that a method is not implemented by a class,
NullReferenceException	Handles errors generated from dereferencing a null object.
OutOfMemoryException	Handles errors generated from insufficient free memory.
StackOverflowException	Handles errors overflow

UNIT-III

WINDOWS APPLICATIONS

In development of any application we need a user interface to communicate with end users. User interfaces are two types.

1. **CUI(Character user interface) (or) console applications**
2. **GUI(graphical user interface) (or) windows applications**

Initially we have only CUI applications, these are also called as console applications

Example: Dos, UNIX OS etc.,

A console application is an application that runs in a console window same as a C and C++ program. Console programs are easy to develop, because console based applications perform all their input and output operations at the command line. To work with console applications in .NET you have to use a class called **Console** that is available within the namespace **System**, which is the root namespace. It doesn't have any graphical user interface. Console Applications will have character based interface.

THE DRAWBACKS OF CONSOLE APPLICATIONS ARE

1. They were not user friendly, as we need to learn the commands first to use them.
2. They did not allow navigating one field to another.
3. They do not support to develop GUI applications.
4. In console applications once the data is entered that is not editable.

To solve the above problems, in early 1990's GUI applications are introduced by Microsoft with its windows OS. Which has a beautiful feature known as Look and Feel. To develop GUI's Microsoft has provided a language in 1990's is visual basic, later when .NET was introduced the support for GUI has been given in all languages of .NET.

Developing graphical user interface: To develop GUI's we need some special components known as controls and these controls are readily available in .NET languages as classes under the namespace **System.Windows.Forms**.

Controls are categorized into two types

1. Container controls**2. Non-container controls**

Container controls: Container controls are controls which are capable of holding other controls on them.

Example: Form, Panel, GroupBox, Split, Tab control

Non-container controls: Non-container controls are controls which are capable of holding any controls on them. They can be used only after being placed on the container.

Example: TextBox, Button, Treeview, DataGridView.

Every control has three things in common like

Properties: these are attributes of a control which have their impact on Look of the control.

Example: Width, Height, BackColor, ForeColor .

Methods: these are actions performed by control.

Example: Clear(),Focus(),Close().

Events: these are time periods which specify when an action has to be performed.

Example: Click, Load, KeyPress, MouseOver.

FROM WHARE WE CAN DEVELOP A WINDOWS APPLICATION (GUI)?

To develop a windows application (GUI) first we need to create the base control that is "Form". To create the form first we need to define a class which is inheriting from the pre-defined class "Form".so that new class becomes a Form.

Example: public class Form1: Form

To run the Form we have created call the static method Run of Application class by passing the object of Form we have created as a parameter.

Example: Application.Run(new Form1());

Or

Form1 f=new Form1();

Application.Run(f);

Developing windows Application: we can develop the windows application in two ways.

- 1. By using a notepad following the above process.**
- 2. By using a visual studio by using Windows Forms Application" project template.**

Developing windows application using notepad: write the code, in a notepad, save, compile and then execute.

Example:

```
using System;
using System.Windows.Forms;
public class Form1:Form
{
    static void Main()
    {
        Form1 f=new Form1();
        Application.Run(f);
    }
}
```

Developing windows application using visual studio: to develop windows application under visual studio

1. **Open the new project window.**
2. **Select Windows Forms Application project template and specify the name of the project.**

For example **windowsproject**. By default the project comes with two classes

1. **Form1**
2. **Program**

Form1 is the class which is inherited from pre-defined class **Form**.

Example: public partial class Form1:Form

Here the class Form1 is partial which means it is defined on multiple files.

1. **Form1.cs**
2. **Form1.Designer.cs**

Note: we will find the **Form1.Designer.cs** file open by default. To open it go to solution explorer expand the node **Form1.cs** and under it we find **Form1.Designer.cs** file, double click on it to open.

The execution of windows application starts from program.cs (or) program class, under which we find a Main method under which object of class Form1 is created for execution.

Example: **Application.Run(new Form1());**

Or

```
Form1 f=new Form1();
```

```
Application.Run(f);
```

Note: Program.cs (or) program class is the main entry point of the project from where the execution starts.

- Windows applications developed under visual studio have two places to work with.

- 1. Design view**

- 2. Code view**

Design view: It is the place where we design the application; This is accessible both to programmers and end user's.

Code view: It is the place where we write the code for the execution of application, this is accessible only to programmers.

Note: because of the design view what visual studio provides we call it as WYSIWYG IDE (what you see is what You Get).

- The code that is present under the windows application is divided into two categories

- 1. Designer code**

- 2. Business Logic**

Designer code: The code which is responsible for construction of the form is known as designer code. Designer code is generated by visual studio under the **IntializeComponent** method of **Designer.cs** file.

Business Logic: The code which is responsible for execution of the form is known as business logic. Business logic is written by the programmers in the form of event procedures.

- Before .NET 2.0 designer code and business logic were defined in a class present under a single file as following.

Form1.cs

```
public class Form1:Form
{
    Designer code
    Business logic
}
```

- From .NET 2.0 with introduction of partial classes, designer code and business logic were separated into two different files but of the same class name.

Form1.cs

```
public partial class Form1:Form
{
    Business logic
}
```

Form1.Designer.cs

```
public partial class Form1
{
    Designer logic
}
```

Events: It is a **time periods which tells when an action to be performed that is when exactly we want to execute a method**. Every control will have number of events under it, where each event occurs on a particular time period. We can access the events of a control also under property window. If you want to write any code under an event that should be executed when the event raises, select the appropriate event in property window and double click on it, that will take you to code view and provides a special method under which we need to write the code.

Now in our project we go to form properties select events, and double click on load event and write the following code under method

Example:

```
private void Form1_Load(..)
{
    MessageBox.Show("welcome to windows applications");
}
```

Again go to design view, double click on the Click Event and write the following code under method.

Example:

```
private void Form1_click(..)
{
    MessageBox.Show("You have clicked on form");
}
```

What happens when we double click on an event of a control in the property window?

When we **double click on an event in property window internally a method gets generated for writing the code and that method has a special name is called event procedure**. Even procedure is a block of code that is bound with an event of control and gets executed whenever the event occurs (or) raises. The written under event procedure will be executed by the event with the help of delegate internally.

Bound



Whenever an event occurs it will call the delegate which will take the responsibility of executing the code that is under event procedure.

Syntax: <control>.<event> +=new <delegate>(<event procedure>);

Example: this. Load +=new EventHandler(Form3_Load);
 button1.Click +=new EventHandler(button1_Click);
 textBox1.KeyPress +=new KeyPressEventHandler(textBox1_KeyPress);

Events and delegates are pre-defined under Base Class Libraries (**events** are defined in **control classes** and the **delegates** are defined under **namespaces**), what is defined here is event procedures. After defining the event procedure under the Form class visual studio links the event, delegate and event procedure with each other .we can see the above code under InitializeComponent method.

Note: one delegate can be used by multiple events to execute event procedures, but all events will not use the same delegates, where different events may use different delegates to execute event procedures.

HOW TO DEFINE EVENT PROCEDURES MANUALLY?

We can define event procedures manually by using the following syntax.

Syntax: <modifiers> void <name>(<object sender, EventArgs e>
 {
 <statements>;
 }

- Event procedures are non-value returning methods, so they are **always void**.
- We can **specify any name to event procedure** , but visual studio follows a **convention while naming them** that is

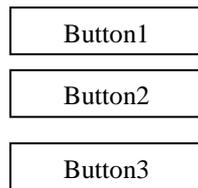
Syntax: <control name>_<event>

Example: **Form1_Load**
 button1_Click
 textbox1_KeyPress

- Event procedure will take two mandatory parameters.
 1. Object sender
 2. EventArgs e

BINDING AN EVENT PROCEDURE WITH MULTIPLE CONTROLS:

- Add a new form to the project that is form4 and place the three buttons.



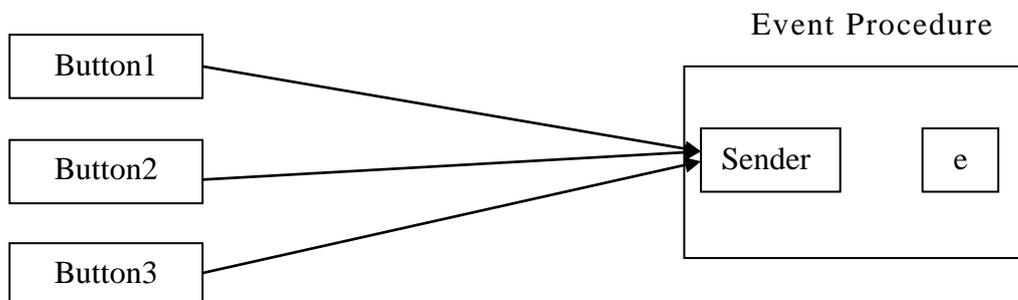
- Now go to events of button1 and double click on Click event to define **button1_Click** event procedure, bind the event procedure with button2 and button3 also and write the following code under the event procedure.

Example: Button b=sender as Button; (or) Button b= (Button) sender;

```

if(b.name="button1")
    MessageBox.Show("button1 is clicked");
else if(b.name="button2")
{
    MessageBox.Show("button2 is clicked");
}
else
{
    MessageBox.Show("button3 is clicked");
}
  
```

When an **event procedure is bound with multiple controls**, any of the control can raise the event in runtime and execute the event procedure, but the object of the control which is raising the event will be coming into the event procedure and captured under the parameter "sender" of event handler as follows.



Every control has number of events under them. But one event will be default for the control. To write the code under default event procedures we can directly double click on the control.

Control	Default Event
Form	Load
Button	Click
TextBox	Text Changed
CheckBox and RadioButton	Checked Changed
Timer	Tick
ListView, ComboBox, ListBox	SelectedIndex Changed

BINDING AN EVENT PROCEDURE WITH MULTIPLE EVENTS OF A CONTROL:

go to form of a new project and double click on form which defines an event procedure Form1_Load, now bind the same event procedure with click event of form also, to do this go to events of form, select click event and drop down beside, which displays the list of event procedures available, select Form1_Load event procedure that is defined previously which binds the event procedure with click event also now under the event procedure write the code.

Types of Events: In c# events are divided into five groups

1. Mouse
2. Focus
3. Drag
4. Key
5. Other Related Events.

Mouse Related Events:

Event	Description
MouseClick	Occurs when the control is clicked by the mouse.
MouseDoubleClick	Occurs when the control is double clicked by the mouse.
MouseDown	Occurs when the mouse pointer is over the control and a mouse button is pressed.
MouseUp	Occurs when the mouse pointer is over the control and a mouse button is released.
MouseEnter	Occurs when the mouse pointer enters onto the control.
MouseLeave	Occurs when the mouse pointer leaves the control.
MouseMove	Occurs when the mouse pointer is moved over the control.

MouseHover	Occurs when the mouse pointer rests on the control.
-------------------	--

Example: Event procedure for MouseEnter.

```
Syntax:    private void button1_MouseEnter(object sender, EventArgs e)
            {
                button1.Height += 30;
                button1.Width += 30;
                button1.Top -= 15;
                button1.Left -= 15;
            }
```

Example: Event procedure for MouseLeave.

```
Syntax:    private void button1_MouseLeave(object sender, EventArgs e)
            {
                button1.Height -= 30;
                button1.Width -= 30;
                button1.Top += 15;
                button1.Left += 15;
            }
```

Example: Event procedure for MouseClick

```
Syntax:    private void button1_MouseClick(object sender, MouseEventArgs e)
            {
                MessageBox.Show(String.Format("Clicked at point ({0}, {1})", e.X, e.Y));
            }
```

Example: Event procedure for MouseDown

```
Syntax:    private void button1_MouseDown(object sender, MouseEventArgs e)
            {
                if (e.Button == MouseButtons.Left)
                    MessageBox.Show("Left Click");
                else if (e.Button == MouseButtons.Right)
                    MessageBox.Show("Right Click");
                else
                    MessageBox.Show("Middle Click");
            }
```

Example: Write a windows application in c#, to create a company logo as a mouse

pointer using MouseEventArgs.

Step1: goto toolbox and Place pictureBox control on Form

Step2: go to Properties of pictureBox and Select an image from harddisk that is

Image:properties.Resources.CTS_logo

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication4
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_MouseMove(object sender, MouseEventArgs e)
        {
            pictureBox1.Location = new Point(e.X, e.Y);
            //Cursor.Hide();
        }
    }
}
```

Example: write a windows application in c#, to display mouse coordinates while your mouse pointer is moving using MouseEventArgs event.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```

using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication10
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void label1_MouseMove(object sender, MouseEventArgs e)
        {
            label1.Text = "x" + e.X + "y=" + e.Y;
        }
    }
}

```

Focus Related Events:

Event	Description
Leave	It occurs while the cursor is coming out from a control
Enter	It occurs while the cursor is entering into a control

Drag Related Events:

Event	Description
DragEnter	it occurs when an object is dragged into the control bounds
DragLeave	it occurs when an object is dragged out of the control bounds
DragOver	it occurs when an object is dragged over the control bounds
DragDrop	it occurs when an drag-and-drop operation is completed

Example: create a windows application in c# to copy data from ms-word to your textbox control using DragEnter event.

Requirements:

Step1: goto toolbox and place textbox on the form

Step2: goto properties of textbox set multiline as true

Step3: goto properties of textbox set allowdrop as true

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;

```

```

using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication17
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void textBox1_DragEnter(object sender, DragEventArgs e)
        {
            if (e.Data.GetDataPresent(DataFormats.Text) == true)
            {
                string s = (string)e.Data.GetData(DataFormats.Text);
                textBox1.Text = s;
            }
            else
            {
                MessageBox.Show("not supported");
            }
        }
    }
}

```

Key Related Events:

Event	Description
KeyUp	Occurs when a key is released
KeyDown	This event is raised when a user presses a physical key
KeyPress	This event is raised Occurs when a key is pressed. This event is not raised by noncharacter keys, unlike KeyDown and KeyUp, which are also raised for noncharacter keys

Example: create windows application in c#, to develop Registration form using validating and keypress events.(name, password, confirm password ,age and

address)

Properties:

1. **Textbox2->passwordchar=*,maxlength=10**

2. **Textbob3->passwordchar=***

3. **Textbox5->multiline=true**

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication21
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void textBox1_Validating(object sender, CancelEventArgs e)
        {
            TextBox tb = sender as TextBox;
            if (tb.Name == "textBox1")
            {
                if (tb.Text.Trim().Length == 0)
                {
                    MessageBox.Show("cannot leave empty");
                    e.Cancel = true;
                    return;
                }
            }
        }
        private void textBox2_Validating(object sender, CancelEventArgs e)
        {
            TextBox tb = sender as TextBox;
            if (tb.Name == "textBox2")
            {
                if (tb.Text.Trim().Length == 0)
                {
                    MessageBox.Show("cannot leave empty");
                    e.Cancel = true;
                    return;
                }
            }
        }
        private void textBox3_Validating(object sender, CancelEventArgs e)
        {
            TextBox tb = sender as TextBox;
            if (tb.Name == "textBox3")
            {
                if (textBox2.Text.Trim() != textBox3.Text.Trim())
```

```

        {
            MessageBox.Show("not matched");
            e.Cancel = true;
            return;
        }
    }
}
private void textBox4_KeyPress(object sender, KeyPressEventArgs e)
{
    if (char.IsDigit(e.KeyChar) == false)
    {
        MessageBox.Show("enter only numbers");
        e.Handled = true;
        return;
    }
}
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("name=" + textBox1.Text + "\n password=" +
        textBox2.Text + "\n confirm password=" + textBox3.Text + "\n age="
        + textBox4.Text + "\n address=" + textBox5.Text);
}
}
}

```

Other Related Events:

Event	Description
Validating	It occurs when the control is validating.
Validated	It occurs when the control is finished validating.
TextChanged	It occurs when the Text property value changes.
Move	It occurs when the control is moved
Click	It occurs when the control is clicked

Example: create a windows application in c#, having two text boxes and three buttons names as factorial, prime and Fibonacci series when you click any button the resultant value will be displayed on the second textbox using button click event.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

```

```
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            int n, i = 1, fact = 1;
            n=int.Parse(textBox1.Text);
            while (i <= n)
            {
                fact = fact * i;
                i++;
            }
            textBox2.Text = fact.ToString();
        }
        private void button2_Click(object sender, EventArgs e)
        {
            int i, j, c;
            int n = int.Parse(textBox1.Text);
            textBox2.Text = "";
            for (j = 2; j <= n; j++)
            {
                c = 0;
                i = 1;
                while (i <= j)
                {
                    if (j % i == 0)
                    {
                        c++;
                    }
                    i++;
                }
                if (c <= 2)
                {
                    textBox2.Text = textBox2.Text + j.ToString() + ",";
                }
            }
        }
        private void button3_Click(object sender, EventArgs e)
        {
            int prev=0, cur=1, next;
            int n = int.Parse(textBox1.Text);
            textBox2.Text = prev.ToString() + cur.ToString();
            next = prev + cur;
            while (next <=n)
```

```

        {
            textBox2.Text = textBox2.Text + next.ToString();
            prev = cur;
            cur = next;
            next = prev + cur;
        }
    }
}

```

Example: create a windows application in c#, to develop a calculator application using button click event.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication12
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        int num1, num2;
        string s, cal;
        private void button1_Click(object sender, EventArgs e)
        {
            cal += button1.Text;
            textBox1.Text = cal;
        }
        private void button2_Click(object sender, EventArgs e)
        {
            cal += button2.Text;
            textBox1.Text = cal;
        }
        private void button3_Click(object sender, EventArgs e)
        {
            cal += button3.Text;
            textBox1.Text = cal;
        }
        private void button4_Click(object sender, EventArgs e)
        {
            cal += button4.Text;
            textBox1.Text = cal;
        }
    }
}

```

```
}
private void button5_Click(object sender, EventArgs e)
{
    cal += button5.Text;
    textBox1.Text = cal;
}
private void button6_Click(object sender, EventArgs e)
{
    cal += button6.Text;
    textBox1.Text = cal;
}
private void button7_Click(object sender, EventArgs e)
{
    cal += button7.Text;
    textBox1.Text = cal;
}
private void button8_Click(object sender, EventArgs e)
{
    cal += button8.Text;
    textBox1.Text = cal;
}
private void button9_Click(object sender, EventArgs e)
{
    cal += button9.Text;
    textBox1.Text = cal;
}
private void button10_Click(object sender, EventArgs e)
{
    cal += button10.Text;
    textBox1.Text = cal;
}
private void button11_Click(object sender, EventArgs e)
{
    s = "+";
    num1 = int.Parse(textBox1.Text);
    textBox1.Text = "+";
    cal = " ";
}
private void button12_Click(object sender, EventArgs e)
{
    num2 = int.Parse(textBox1.Text);
    if (s == "+")
        num1 = num1 + num2;
    else if (s == "-")
        num1 = num1 - num2;
    else if (s == "*")
        num1 = num1 * num2;
    else if (s == "/")
        num1 = num1 / num2;
```

```

        textBox1.Text = num1.ToString();
        cal = " ";
    }
    private void button13_Click(object sender, EventArgs e)
    {
        s = "-";
        num1 = int.Parse(textBox1.Text);
        textBox1.Text = " -";
        cal = " ";
    }
    private void button14_Click(object sender, EventArgs e)
    {
        s = "*";
        num1 = int.Parse(textBox1.Text);
        textBox1.Text = " *";
        cal = " ";
    }
    private void button15_Click(object sender, EventArgs e)
    {
        s = "/";
        num1 = int.Parse(textBox1.Text);
        textBox1.Text = " /";
        cal = " ";
    }
    private void button16_Click(object sender, EventArgs e)
    {
        textBox1.Text = " ";
        cal = " ";
    }
}
}

```

Properties:

```

button1->Text = "0";
button2->Text = "1";
button3->Text = "2";
button4->Text = "3";
button5->Text = "4";
button6->Text = "5";
button7->Text = "6";
button8->Text = "7";
button9->Text = "8";
button10->Text = "9";
button11->Text = "+";
button12->Text = "=";
button13->Text = "-";
button14->Text = "*";
button15->Text = "/";
button16->Text = "c";

```

Example: write a windows application in c#, having two textbox controls and a button control, validate that the textbox values are not blank. Clicking the button will interchange the textbox values and the button disappears using validating and click and leave events.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication5
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void textBox1_Validating(object sender, CancelEventArgs e)
        {
            TextBox tb = sender as TextBox;
            if (tb.Name == "textBox1")
            {
                if (tb.Text.Trim().Length == 0)
                {
                    MessageBox.Show("cant leave empty");
                    e.Cancel = true;
                    return;
                }
            }
        }
        private void textBox2_Validating(object sender, CancelEventArgs e)
        {
            TextBox tb = sender as TextBox;
            if (tb.Name == "textBox2")
            {
                if (tb.Text.Trim().Length == 0)
                {
                    MessageBox.Show("cant leave empty");
                    e.Cancel = true;
                    return;
                }
            }
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string temp;
```

```

        temp = textBox1.Text;
        textBox1.Text = textBox2.Text;
        textBox2.Text = temp;
        button1.Visible = false;
    }
    private void textBox2_Leave(object sender, EventArgs e)
    {
        TextBox tb = sender as TextBox;
        if (tb.Name == "TextBox2")
        {
            if (tb.Text.Trim().Length == 0)
            {
                MessageBox.Show("cant leave empty");
                return;
            }
        }
    }
    private void textBox1_Leave(object sender, EventArgs e)
    {
        TextBox tb = sender as TextBox;
        if (tb.Name == "TextBox1")
        {
            if (tb.Text.Trim().Length == 0)
            {
                MessageBox.Show("cant leave empty");
                return;
            }
        }
    }
}

```

Example: create a windows application, which implements exceptions concept using TextChanged event.

```

using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void textBox1_TextChanged(object sender, KeyPressEventArgs e)
        {
            try
            {
                int a=int.Parse(textBox1.Text);
                a=a+1000;
            }
            catch { }
        }
    }
}

```

```

        MessageBox.Show("a="+a);
    }
    catch(FormatException e)
    {
        MessageBox.show(e.Message);
        MessageBox.show(e.GetType().ToString());
    }
    finally
    {
        MessageBox.show("end of process");
    }
}
}

```

Example: create windows application in c#, to copy the selected data from first listbox to second listbox.

Process:

1. Goto properties for selecting multiple items we use set selectionmode=multisimple

```

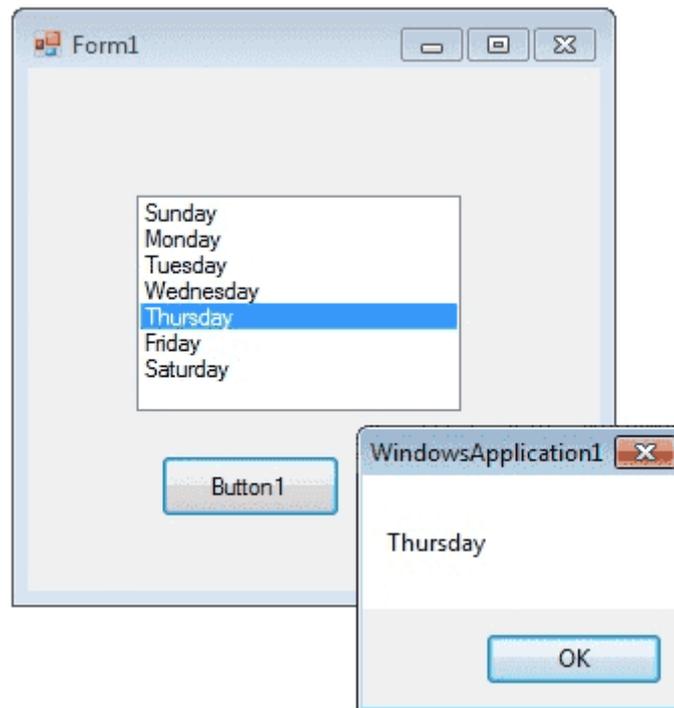
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication18
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            listBox2.Items.Clear();
            string str = "";
            foreach (object ob in listBox1.SelectedItems)
            {
                str += ob.ToString() + "\t";
            }
            str = str.Substring(0, str.Length);
            listBox2.Items.Add(str);
        }
    }
}

```

Example: Create a windows application which demonstrates the concept of listbox using button click and form load events.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            listBox1.Items.Add("Sunday");
            listBox1.Items.Add("Monday");
            listBox1.Items.Add("Tuesday");
            listBox1.Items.Add("Wednesday");
            listBox1.Items.Add("Thursday");
            listBox1.Items.Add("Friday");
            listBox1.Items.Add("Saturday");
            listBox1.SelectionMode = SelectionMode.MultiSimple;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            foreach (Object obj in listBox1.SelectedItems )
            {
                MessageBox.Show(obj.ToString ());
            }
        }
    }
}
```

Output:



Note: Handled property when set as true will restrict the key value to enter into the control.

Example: create a windows application in c# to demonstrate comboBox and ListBox controls using button click event.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication2
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(comboBox1.SelectedItem.ToString());
            string str = " ";
            foreach (object ob in listBox1.SelectedItems)
            {
                str += ob.ToString() + " ,";
            }
            str = str.Substring(0, str.Length);
            MessageBox.Show(str);
        }
    }
}

```

```

    }
  }
}

```

Example: create a windows application in c# to demonstrate comboBox and CheckedListBox controls using button click event.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication22
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show(comboBox1.SelectedItem.ToString());
            string str = " ";
            foreach (object ob in checkedListBox1.CheckedItems)
            {
                str += ob.ToString() + " ,";
            }
            str = str.Substring(0, str.Length);
            MessageBox.Show(str);
        }
    }
}

```

Example: create a windows application which demonstrates the concept of check boxes using button click event.

```

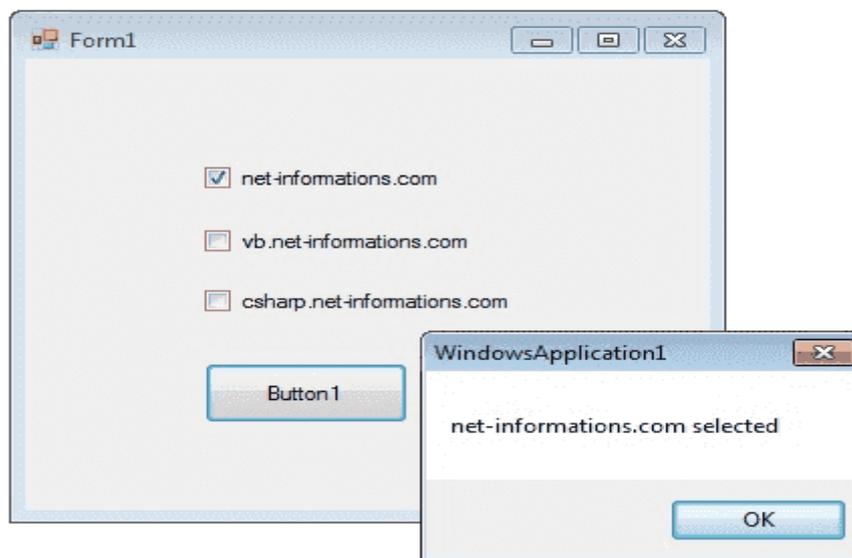
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication6
{
    public partial class Form1 : Form
    {

```

```
public Form1()
{
    InitializeComponent();
}
private void button1_Click(object sender, EventArgs e)
{
    string msg = "";
    if (checkBox1.Checked == true)
    {
        msg = "net-informations.com";
    }

    if (checkBox2.Checked == true)
    {
        msg = msg + " vb.net-informations.com";
    }
    if (checkBox3.Checked == true)
    {
        msg = msg + " csharp.net-informations.com";
    }
    if (msg.Length > 0)
    {
        MessageBox.Show(msg + " selected ");
    }
    else
    {
        MessageBox.Show("No checkbox selected");
    }
    checkBox1.ThreeState = true;
}
}
```

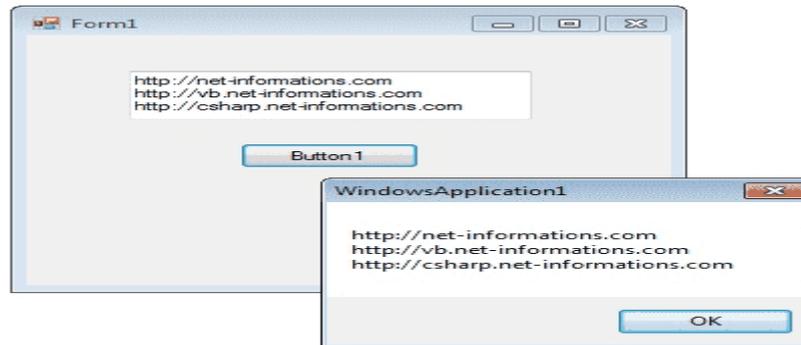
Output:



Example: create a windows application which demonstrates the concept of textbox using button click and form load events.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Width = 250;
            textBox1.Height = 50;
            textBox1.Multiline = true;
            textBox1.BackColor = Color.Blue;
            textBox1.ForeColor = Color.White;
            textBox1.BorderStyle = BorderStyle.Fixed3D;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string var;
            var = textBox1.Text;
            MessageBox.Show(var);
        }
    }
}
```

Output:



Example: create a windows application which changes the properties of form using form load event.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            this.Text = "Change Prperties Through Coding";
            this.BackColor = Color.Brown;
            this.Size = new Size(350, 125);
            this.Location = new Point(300, 300);
            this.MaximizeBox = false;
        }
    }
}
```

Output:

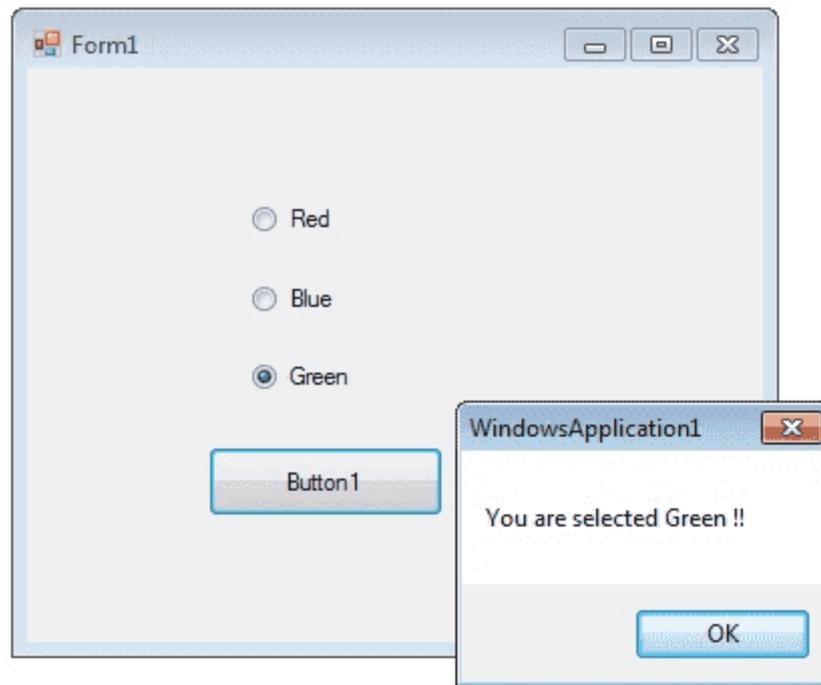


Example: create a windows application which demonstrates the concept of Radio buttons using button click event.

```
using System;
using System.Drawing;
using System.Windows.Forms;
namespace WindowsFormsApplication1
```

```
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            radioButton1.Checked = true;
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (radioButton1.Checked == true)
            {
                MessageBox.Show ("You are selected Red !! ");
                return;
            }
            else if (radioButton2.Checked == true)
            {
                MessageBox.Show("You are selected Blue !! ");
                return;
            }
            else
            {
                MessageBox.Show("You are selected Green !! ");
                return;
            }
        }
    }
}
```

Output:



Example: create a windows application which checks whether two checkbox values are same or not using button click event.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication9
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            string a = textBox1.Text;
            string b = textBox2.Text;
            if (a.Equals(b))
            {
                MessageBox.Show("Both values are same.");
            }
            else
```

```

        {
            MessageBox.Show("Values are different.");
        }
    }
}

```

WORKING WITH CONTROLS:

Radiobutton: a radio button control provides a round interface to select one option from a number of options when paired with other RadioButton controls. When a user clicks on a radio button, it becomes checked, and all other radio buttons with same group become unchecked

Property	description
Appearance	A circular button with a label beside it.
CheckAlign	Determines the alignment of the button. The default is middle left, which will show the button to the left of the label
Checked	When set true, the radio button will be in its ON state and a dot will be seen inside the button.
Text	Sets the text inside the label of the radio button.

Creating a radio button: we can create a RadioButton control in two ways.

Creating a RadioButton using a Forms designer at design time: To create a RadioButton control at design time, you simply drag and drop a RadioButton control from ToolBox to a Form in visual studio. Once a RadioButton is on the Form, You can move it around and resize it using mouse and set its properties and events.

Creating RadioButton dynamically: creating a RadioButton control at runtime is merely work of creating an instance of radiobutton class, set its properties and add RadioButton class to the Form controls.

Procedure:

1. Create a dynamic radiobutton is to create an instance of radiobutton class. The following code creates radiobutton control object. Write this code in button click event.

Syntax: RadioButton dynamicRadioButton=new RadioButton();

2. you may set properties of RadioButton control. The following code sets location, width, height, background color, foreground color, Text, Name, and font properties of a RadioButton.

**Syntax: dynamicRadioButton.Left=20;
 dynamicRadioButton.Top=100;**

```

dynamicRadioButton.Width=200;
dynamicRadioButton.Height=30;
dynamicRadioButton.BackColor=Color.Orange;
dynamicRadioButton.ForeColor=Color.Black;
dynamicRadioButton.Text="I am dynamic radio button";
dynamicRadioButton.Name="dynamicRadioButton";
dynamicRadioButton.Font=new Font("Georgia",12);

```

once a radiobutton control is ready with its properties, next step is to add the radiobutton control to the form. To do so, we use **Form.Controls.Add** method. The following code adds a radiobutton control to the current Form.

Syntax: `Controls.Add(dynamicRadioButton);`

Checkbox control: checkbox control is also a type of button and appears as an empty box with a label beside it. By default, when the empty box is clicked, a check will show up inside the box

Property	description
Checked	Determines if the check box is checked
CheckedState	Tells whether the checkbox is checked, unchecked
ThreeState	If true, the checkbox can accept an Intermediate state.

The checkbox control has three states by setting the ThreeState property to true. Those three states are Checked, Unchecked or Intermediate. Intermediate indicates that value of the checkbox is invalid or cannot be determined.

You can also change the state of the control in code by using the CheckState property. The CheckState accepts values from the CheckState enumeration.

```

checkBox1.CheckState=CheckState.Checked;
checkBox2.CheckState=CheckState.Unchecked;
checkBox3.CheckState=CheckState.Intermediate;

```

If the checkbox is set to only accept two states, on or off (by setting threestate property to false), then you can simply use the checked property of the checkbox which accepts either true to make the checkbox checked, or false to make it unchecked.

BUTTON, LABEL AND TEXTBOX CONTROLS:

Button: It is used for taking acceptance from a user to perform an action.

Label: It used for displaying static text on the user interface.

TextBox: It is used for taking text input from the user, this control can be used in three

different ways

1. Single line text(default)
2. Multiline text(text area)
3. Password field

The default behavior of control is single line text; to make it multiple **set the property multiline of the control as true**.by default the text area will not have any scroll bars to navigate up and down or left and right, to get them **set the scrolls property either as vertical or horizontal or both**, default is none.

To use the control as password field either **set the PasswordChar property of control with a character**, which we want to use as password character like * or # or \$.(or) **UseSystemPasswordChar property as true** which indicates the text in the control should appear as the default password character.

ComboBox, ListBox, CheckedListBox:

ComboBox: It is the combination of both **TextBox** and **ListBox** controls. **Combobox** allows only single selection but it is editable which gives a chance to either select from the list of values available or enter a new value.

ListBox Control: This control is used to display group of items to the user can select a single item **or** group of items randomly **or** range of items.

CheckedListBox: **CheckedListBox** by default allows multi selection.

Adding values to the controls: we can add values to the controls in 4 ways

1. Go to properties of the control select Items and click on the button beside it, which opens a window, enter the values we want to add,but each in newline.
2. By using Items.Add method of the control we add values, but only one at a time.

Syntax: `control.Items.Add(object value);`

Example: `listBox1.Items.Add("ap");`

3. By using Items.AddRange method of the control an array of values can be added at a time.

Syntax: `control.Items.AddRange(object[] values);`

Example: `string[] cities={"a","b","c"};`

`CheckedListBox1.Items.AddRange(cities);`

4. Using the data source property of the control we can bind values of a table to the control.

Syntax: `control.DataSource=datatable`

For displaying single column we need to specify the column to be displayed using the DisplayMember property.

Syntax: control.DisplayMember=col name

TIMER CONTROL:

- A timer control which can raise an event at user defined intervals that is we can specify an interval time period and once the interval elapses automatically the event procedure associated with the control gets executed.
- The timer's logic will be executed only when **Enabled=true**.
- Timer control is having only one event called as tick.
- Timer1_Tick will be executed after equal intervals which need to be specified in terms of **milliseconds**.

Example: create a windows application which demonstrates timer control using Tick event.

Procedure:

Step1: go to toolbox and place the label on form

Step2: go to toolbox and place the timer on form

Step3: set the property of timer **Enabled=true**

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication23
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void timer1_Tick(object sender, EventArgs e)
        {
            label1.Text = DateTime.Now.ToLongTimeString();
        }
    }
}
```

Example: create a windows application in c# to display current time for the local system using timer tick event.

```

using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void timer_Tick(object sender, EventArgs e)
        {
            Label1.Text=DateTime.Now.ToLongTimeString();
        }
    }
}

```

Example: create a windows application in c# to develop paranoid game using keydown and tick events.

Process:

- 1. Place three labels and one pictureBox**
- 2. Place two timers on a form**

Prproperties:

Form: **set FormBorderStyle=Fixed3d**
 Start position=center of screen

Label: **Autosize=false**
 Labelbackcolor=controlText
 Location=25,510

pictureBox: **select an image from hadrdisk**
 size=25,25
 location=1,180

timer1: enable=true

```

using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
    }
}

```

```
        private void Form1_KeyDown(object sender,EventArgs e)
        {
            int k=(int)e.KeyCode;
            if(k==37&&label1.Left>0)
                label1.Left=label1.Left-8;
            if(k==39&&label1.Right<this.width-10)
                label1.Left=label1.Left+8;
        }
        int y=2;
        private void timer1_Tick(object sender,EventArgs e)
        {
            pictureBox1.Top=pictureBox1.Top-5;
            pictureBox1.Left=pictureBox1.Left+y;
            if(pictureBox1.Top==0)
            {
                timer1.Enabled=false;
                timer2.Enabled=true;
            }
            if(pictureBox1.Right>this.width-10)
                y=y-2;
            if(pictureBox1.Left<0)
                y=2;
        }
        int x=2,score=0,level,k=1;
        private void timer2.Tick(object sender,EventArgs e)
        {
            pictureBox1.Top=pictureBox1.Top+5;
            pictureBox1.Left=pictureBox1.Left+x;
            if(pictureBox1.Bottom==label1.Top&&
                pictureBox1.Left==label1.Left&&
                pictureBox1.Right==label1.Right)
            {
                timer2.Enabled=false;
                timer1.Enabled=true;
                score=score+1;
                label2.Text=score.ToString();
                if(score==k)
                {
                    level=level+1;
                    k=k+5;
                }
                Label3.Text=level.Tostring();
            }
            if(pictureBox1.Bottom>label1.Top)
            {
                timer2.Enabled=false;
                timer1.Enabled=false;
                MessageBox("game over");
            }
        }
    }
```

```

timer1.Interval=10;
timer2.Interval=10;
if(pictureBox1.Right>this.Width-10)
x=x-2;
if(pictureBox1.Left<0)
x=2;
}
}
}

```

Notes:

1. **Char.IsDigit(char):** it will return true if the given char is a numeric or else returns false.
2. **Convert.ToInt32(char):** it will return ascii value of the given character.

InitializeComponent() method in Visual Studio.NET C# is method that is automatically created and managed by Windows Forms designer and it defines everything you see on the form. Everything done on the form in VS.NET using designers generates code. Every single control added and property set will generate code and that code goes into InitializeComponent() method. When you run the app (or open the form in VS.NET), that code creates and configures controls just as you did in the designer. So every single change you make and control you add ends up in InitializeComponent() method. This means that doing things at design-time and run-time is essentially the same. Note that you should not modify this method manually since it might confuse the VS.NET designer. However you should use it to learn how to do things from code and how to setup controls and components correctly. In C# access to this method is always visible through form constructor. Simply position the cursor in InitializeComponent method call and press F12 in VS.NET and Form .designer file will open to show the implementation of the InitializeComponent method.

Problems with File Handling:

- Column information is not allowed.
Example: 5 raj 5000
- Numbers of columns cannot be restricted.
- Data type of column is not allowed.
- There is a possibility of data redundancy (duplication of data).
Example: 5 raj 5000
 5 raj 5000
- It is difficult to handle data manipulations.
- There is no security for the data.

To overcome all these problems ANSI introduced a concept called as **Database**. Database is a **collection of inter-related data** and databases are divided into three

types.

- **DBMS(Database management systems):** It just stores the data.

Examples: sysbase, Foxpro etc.

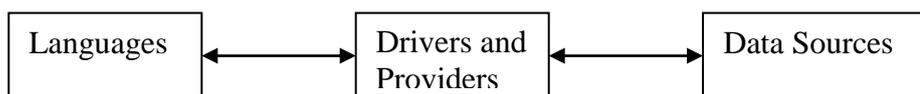
- **RDBMS(Relational Database management systems):** stores information about the data and how they are related.

Examples: IBM's DB2, Informix, Oracle, Sybase, Microsoft's Access, Fox Base, Paradox etc

- **ORDBMS(Object Relational Database management systems):** It is a combination of relational database management with object oriented features.

Example: IBM, Informix, Objectstore, Oracle, Versant

DATA SOURCE COMMUNICATION: Every application deals with data in some manner, whether that comes from memory, databases, XML files, text files, or something else. The location where we store the data can be called as a data source or data store where data source can be a file, database, or indexing server etc. Programming languages cannot communicate with data sources directly because each data source adopts a different protocol (set of rules) for communication. To facilitate the process of communication Microsoft has provided the solution in the form of Intermediate Components like “Drivers and Providers”.



DRIVERS: Drivers are used to communicate with databases.

Types of drivers: We have two types of drivers

1. JET(Joint Engine Technology)Drivers
2. ODBC(Open Database Connectivity)Drivers

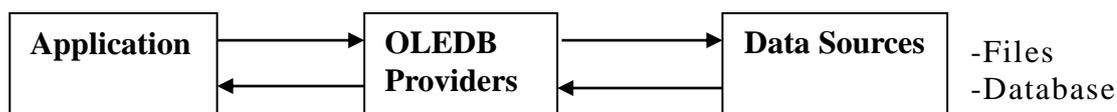
JET (Joint Engine Technology) Drivers: These are designed to communicate with local databases. A local database means which are loaded in client application memory.

ODBC (Open Database Connectivity) Drivers: These are designed to communicate with remote databases like SQL server, oracle etc.

Drawbacks of Drivers: Driver is not a part of an application, first it should be explicitly install on the machine where the applications installed and then configure with an application.

PROVIDERS: To overcome the problems in Drivers Microsoft has provided one solution known as OLEDB (Object Linking and Embedding Database) Providers.

Providers are used for communicating with Data Sources and they become a part of application which does not require explicit installation.



But both Drivers and providers suffers from a common problem that is, as they were designed using Native Code Languages, which leads platform dependency, purely for windows.

INTRODUCTION OF ADO.NET: Before ADO.NET we use ADO (**Active Database Object**) to access data from database. Basically ADO has **automatic driver detection technique** and it has only one drawback that is only provides a connected environment so efficiency of system may decrease. To solve this problem Microsoft introduced ADO.NET (**ActiveX Data Objects**), which is a new database technology used by .Net platform (introduced in 2002). In fact it is a set of classes used to provide communication between an application and a database.

Or

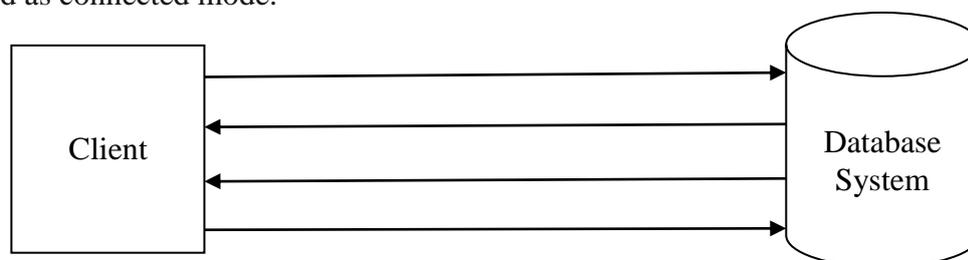
ADO.NET is a data access technology which provides communication between relational and non-relational systems through a common set of components.

Or

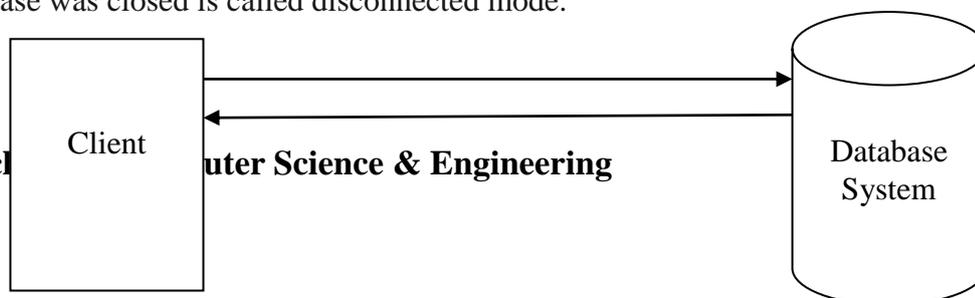
ADO.NET is an object oriented set of libraries that allows you to interact with data sources (databases, text file, XLspreadsheet, XML file).

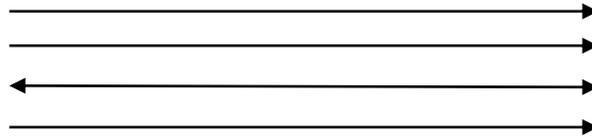
- ADO.NET is a part of .NET Framework.
- It supports both **connected & disconnection mode of data access**.

Connected mode: In this the connection must be opened to access the data from the database is called as connected mode.



Disconnection mode: In this we can access the data from database even when connection to database was closed is called disconnected mode.





- ADO.NET uses XML to store and transfer data among applications, which is not only an industry standard but also provide fast access of data for desktop and distributed applications.
- ADO.NET is scalable and Interoperable.
- ADO.NET is used to develop client-server architecture.
- ADO.NET supports two types of database connections.
 1. **Managed connection:** In this we can establish a connection with specific database. Managed connection works based on **Tabular Data Stream (TDS) protocol (used to transfer data between a database server and a client)**
 2. **Unmanaged Connection:** In this we can establish a connection with any type of database. Unmanaged connection works based on OleDb provider.

ADO.NET NAMESPACES: Namespaces is collection of classes. To work with ADO.NET Microsoft has introduced **System.Data.Dll** Assembly, which contains namespaces. They are

System.Data: It contains the common classes for connecting, fetching data from database. Classes are like as DataTable, DataSet, DataView etc.

System.Data.SqlClient: It contains classes for connecting, fetching data from Sql Server database. Classes are like as SqlDataAdapter, SqlDataReader etc.

System.Data.OracleClient: It contains classes for connecting, fetching data from Oracle database. Classes are like as OracleDataAdapter, OracleDataReader etc.

System.Data.OleDb: It contains classes for connecting, fetching data from any database (like msaccess, db2, oracle, sqlserver, mysql). Classes are like as OleDbDataAdapter, OleDbDataReader etc.

System.Data.Odbc: It contains classes for connecting, fetching data from any database (like msaccess, db2, oracle, sqlserver, mysql). Classes are like as OdbcDataAdapter, OdbcDataReader etc.

ADO.NET DATA PROVIDERS AND OBJECTS:

- The .NET framework includes mainly three data providers for ADO.NET. They are
 1. SQL server Data Provider
 2. OLEDB Data Provider

3. ODBC Data Provider

- The .NET framework has four objects which provide the functionalities of Data Providers in ADO.NET. They are
 1. Connection Object
 2. Command Object
 3. DataReader Object
 4. DataAdapter Object
- Each and every operation we perform on a Data Source involves three steps
 1. Establishing a connection with data source
 2. Sending request to data source as a SQL statement
 3. Capturing the results given by data source

1. **Connection Object:** The connection object is the part of the ADO.NET Data Provider and it is a unique session with the data source. The Connection Object is handling the part of physical communication between the C# application and the Data Source. The Connection Object, connect to the specified Data Source and open a connection between the C# application and the Data Source, depends on the parameter specified in the Connection String. When the connection is established, SQL Commands will execute with the help of the Connection Object and retrieve or manipulate data in the Data Source. Once the Database activity is over, Connection should be closed and releases the Data Source resources. To open the channel for communication we can use the **connection class**.

Constructors of the connection class:

- **Connection()**
- **Connection(string ConnectionString)**

ConnectionString: Connection string is a collection of attributes that are used for connecting with a DataSource, those are

- Provider
- Data Source
- User Id and Password
- Database or Initial Catalog
- Trusted_Connection or Integrated security
- DSN

Provider: provider is required for connecting with any data sources, we have

a different provider available for each data source.

Data Source	Provider
Oracle	MSDAORA.1
SQL Server	SQLOLEDB
MS-Access or MS-Excel	Microsoft.Jet.Oledb.4.0
MS-Indexing Server	Msidxs

Data Source: It is the name of target machine to which we want to connect with but it is **optional when the data source is on a local machine.**

User Id and Password: As databases are secured places for storing data, to connect with them we require a valid id and password.

Database or Initial Catalog: These attributes are used while connecting with SQL Server database to specify the name of database we want to connect them.

Trusted_Connection or Integrated security: These attributes are also used while connecting with SQL Server Database only to specify that we want to connect with the server using Windows Authentication. In this case we should not again use User Id and Password attributes.

DSN: This attribute is used to connect with data sources by using Odbc Drivers.

Connection string for Oracle:

Syntax: `ConnectionString="Provider=Msdora;user Id=Scott; password=tiger;"`;

Connection string for SQL Server:

Syntax:

`ConnectionString="Provider=SQLOledb;user Id=Scott; password=tiger; Database=database name;"`;

Connection string for OLEDB:

Syntax: `connectionString = "Provider=Microsoft.Jet.OLEDB.4.0; Data Source=yourdatabasename.mdb;"`;

Connection string for ODBC:

Syntax: `connectionString = "Driver={Microsoft Access Driver (*.mdb)}; DBQ=yourdatabasename.mdb;"`;

Methods of Connection class:

- **Open():** This method opens a connection with data source.
- **Close():** this method closes the connection that is open


```

        id=system; password=raja;");
        ocon.Open();
        MessageBox.Show(ocon.State.ToString());
        ocon.Close();
        MessageBox.Show(ocon.State.ToString());
    }
}
}

```

Example: create a ADO.NET application to establish unmanaged connection with SQL server (using OleDb provider).

1. open a new project of type Windows Forms Application and name it is an DBOperations.
2. Place one button on the form and set their caption as “Connect with oracle using OLEDB Provider.
3. Goto to code view and write the following code.

```
using System.Data.OleDb;
```

Declarations: OleDbConnection scon;

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication25
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection ocon;
        private void button1_Click(object sender, EventArgs e)
        {
            ocon=new OleDbConnection("Provider= SQLOLEDB;user id=system;
                                     password=raja;databade=databasename;");
            scon.Open();
            MessageBox.Show(ocon.State.ToString());
            scon.Close();
            MessageBox.Show(ocon.State.ToString());
        }
    }
}

```

```

    }
}

```

2. Command Object: The Command Object in ADO.NET executes SQL statements and Stored Procedures against the data source specified in the C# Connection Object. The Command Object requires an instance of a C# Connection Object for executing the SQL statements. In order to retrieve a resultset or execute an SQL statement against a Data Source, first you have to create a Connection Object and open a connection to the Data Source specified in the connection string. Next step is to assign the open connection to the connection property of the Command Object. Then the Command Object can execute the SQL statements. After the execution of the SQL statement, the Command Object will return a result set. We can retrieve the result set using a Data Reader. In this process we send a request to Data Source by specifying the type of action we want to perform using a SQL Statement like Select, Insert, Update, and Delete or by calling a stored procedure. To send and execute those statements on data source we use the class command.

➤ **Constructors of the class:**

➤ **Command()**

➤ **Command(string CommandText, Connection con)**

➤ **Properties of command class:**

Connection: This property sets or gets the connection object associated with command object.

CommandText: This property contains a String value that represents the command that will be executed against the Data Source.

The object of class Command can be created in any of the following ways:

```

Command cmd=new Command();
cmd.Connection=con;
cmd.CommandText=SQL statement;

```

or

```

Command cmd=new Command("SQL statement",con

```

➤ **Methods of Command class:**

➤ ExecuteReader()

➤ ExecuteScalar()

➤ ExecuteNonQuery()

ExecuteReader(): this method is used when we want to execute a Select statement that returns data as rows and columns. This method returns an object of class DataReader which holds data that is retrieved from data source in the form of rows and columns.

Syntax: `cmd.ExecuteReader()`

ExecuteScalar(): This method is used when we want to execute Select Statement that returns a single value result from Database after the execution of the SQL Statement. This method returns result of the query in the form of an object. If the **ResultSet** contains more than one columns or rows , it will take only the value of first column of the first row, and all other values will ignore. If the Result Set is empty it will return a **NULL**reference.

Syntax: `cmd.ExecuteScalar()`

ExecuteNonQuery(): This method is used when we want to execute any SQL Statement other than select, like **Insert, Update or delete** etc. This method returns an integer that tells the number of rows affected by the statement.

Syntax: `cmd.ExecuteNonQuery()`

Example: create a ADO.NET application in C#, to create a table and insert values into created table.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication25
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection ocon;
        OleDbCommand cmd1,cmd2;
        private void button1_Click(object sender, EventArgs e)
        {
            ocon =new OleDbConnection("Provider=MSDAORA.1;
```

```

        user id=system;password=raja;");
        ocon.Open();
        cmd1=new OleDbCommand("create table student1(sid number,sname
                                varchar(10))",ocon);
        cmd2=new OleDbCommand("insert into student1 values('1','raja')",ocon);
        cmd1.ExecuteNonQuery();
        MessageBox.Show("table created");
        cmd2.ExecuteNonQuery();
        MessageBox.Show("1 row inserted");
    }
}
}

```

DataReader Object: The DataReader Object **provides a connection oriented data access** to the Data Sources. A **Connection Object** can contain only one DataReader at a time and the connection in the DataReader remains open, also it cannot be used for any other purpose while data is being accessed. When we started to read from a DataReader it should always be open and positioned **prior to the first record**. A DataReader **provides an easy way for the programmer to read the data from a database**. DataReader is a **stream-based, forward read-only cursor because it moves forward through the data**. The DataReader not only allows you to move forward through each record of the database, but it also **enables you to parse the data from each column**.

Syntax: **COMmand cmd= new Command (“select * from students”,Con);**
 DataReader dr = cmd.ExecuteReader();

DataReader Methods:

Read(): The Read() method in the DataReader is used to **read the rows from DataReader** and it always moves forward to a new valid row, if any row exist .

Syntax: **DataReader.Raed();**

NextResult(): This method is used to **navigate from current table to next table**.

Systax: **dr.NextResult();**

GetValue(int index):This method is used for **getting the column value from the row** which pointer was pointing by specifying column index position.

Systax: **dr.GetValue(0).ToString();**

Or

dr[0].ToString();

Or

dr[“sal”].ToString();

GetName(int index): This method returns the **name of the column for specified index**.

Syntax: **dr.GetName(0);**

DataReader properties:

fieldcount: It returns the number of columns present in the result set.

Syntax: dr.fieldcount;

Example: create an ADO.NET application in C#, to retrieve the values from the table using DataReader object.

Procedure:

Step1: place two buttons on form and name it as next and stop.

Step2: place two labels for representing column names.

Step3: place two text boxes on form for displaying values from the table.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication25
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection ocon;
        OleDbDataReader dr;
        private void Form1_Load(object sender, EventArgs e)
        {
            ocon = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system; password=raja");
            ocon.Open();
            OleDbCommand cmd2 = new OleDbCommand("select * from
                                                student4", ocon);

            dr = cmd2.ExecuteReader();
            label1.Text = dr.GetName(0);
            label2.Text = dr.GetName(1);
        }
        private void button1_Click(object sender, EventArgs e)
        {
            if (dr.Read())
            {
                textBox1.Text = dr.GetValue(0).ToString();
                textBox2.Text = dr.GetValue(1).ToString();
            }
        }
    }
}
```

```

        else
        {
            MessageBox.Show("no more records");
        }
    }
    private void button2_Click(object sender, EventArgs e)
    {
        ocon.Close();
        MessageBox.Show(ocon.State.ToString());
    }
}
}

```

Example: create an ADO.NET application in C#, to retrieve the values based on the user input using DataReader object.

Procedure:

Step1: place one button on the form and name it as show.

Step2: place two labels for representing column names.

Step3: place two text boxes one for user input and another one for getting the value from the database.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication25
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbDataReader dr;
        OleDbConnection ocon;
        OleDbCommand cmd1;
        private void button1_Click(object sender, EventArgs e)
        {
            ocon = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system; password=raja");
            ocon.Open();
            OleDbCommand cmd1 = new OleDbCommand("select * from
                                                student4 where sid="+textBox1.Text+"", ocon);
            dr = cmd1.ExecuteReader();
            while (dr.Read())
            {

```


- The class which is responsible for loading data into DataReader from a DataSource is called Command, in the same way DataAdapter class is used to provide communication between DataSource and DataSet. DataAdapter is used to transfer data between DataSet and DataSource (DataAdapter is not used for holding the data).

DataReader ← Command → DataSource

DataSet ↔ DataAdapter ↔ DataSource

- Constructors for DataAdapter class

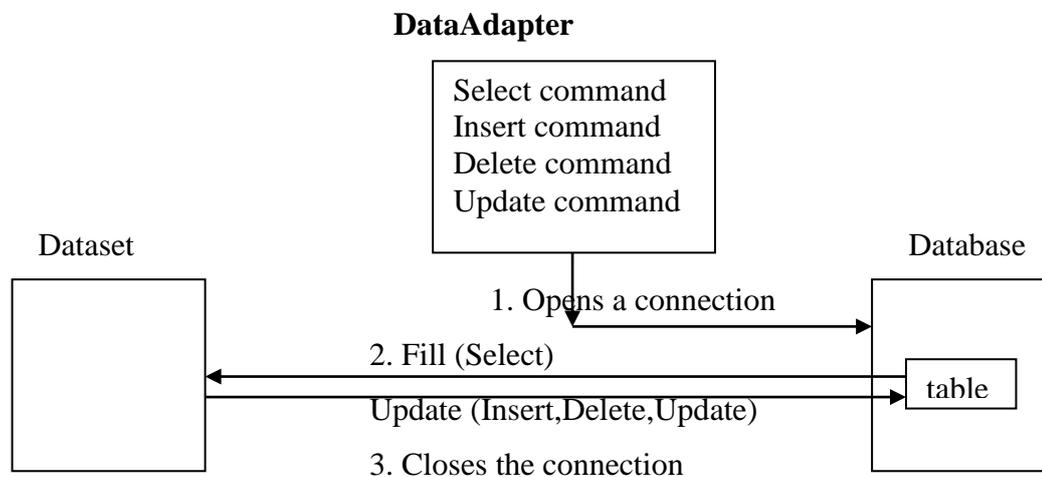
DataAdapter(string statement, Connection con)

DataAdapter(Command cmd)

- Methods of DataAdapter class are:

Fill(DataSet ds, string tablename): This method is used for loading data from DataSource to DataSet.

Update(DataSet ds, string tablename): This method is used to transfer changes made on data from DataSet to DataSource.



- DataAdapter internally contains four commands in it
 - **Select command**
 - **Insert command**
 - **Update command**
 - **Delete command**
- When we call Fill() method on a DataAdapter following actions takes place internally.
 - **Open a connection with a DataSource.**
 - **Executes the select command present under it on the DataSource and loads data from DataSource to DataSet.**
 - **Closes the connection.**

- DataSet is updatable, once the data is loaded into it we can make changes on data like adding or modifying or deleting of records, after making changes on data in DataSet if we want to send those changes back to DataSource we need to call Update method on DataAdapter, which performs the following
 - **Reopens the connection with the DataSource.**
 - **Changes that are made on data in DataSet will be sent back to corresponding DataSource, where in this process it will make use of Insert, Update and Delete commands of DataAdapter.**
 - **Closes the connection.**

ACCESSING DATA FROM DATASET: DataSet is a collection of tables where each table is represented as a class **DataTable** and **identified by its index position or name**. Every DataTable again is a **collection of rows and columns** where each row is represented as a class **DataRow** and **identified by its index position** and each column is represented as a class **DataColumn** and **identified by its index position or name**.

The object can be created for DataSet class by using the following syntax:

Syntax: `DataSet ds=new DataSet();`

1. Accessing DataTable from DataSet:

Syntax: `ds.Tables[index]`

Or

`ds.Tables["tablename"]`

Example: `ds.Tables[0]`

Or

`ds.Tables["emp"]`

2. Accessing DataRow From DataTable

Syntax: `ds.datatable.Rows[index]`

Example: `ds.Tables[0].Rows[0]`

3. Accessing DataColumn From DataTable

Syntax: `ds.datatable.Columns[index]`

Or

`ds.datatable.Columns["name"]`

Example: `ds.Tables[0].Column[0]`

Or

`ds.Tables[0].Column["eno"]`

4. Accessing a cell from DataTable

Syntax: ds.datatable.Rows[row][col]

Example: ds.Tables[0].Rows[0][0]
 ds.Tables[0].Rows[0][“eno”]

Note: DataReader provides pointer based access to the data, so we can get data only in a sequential order whereas DataSet provides index based access to the data, so we can get data from any location randomly.

Example: create an ADO.NET application in C#, to demonstrate DataAdapter object.

Procedure:

Step1: place three buttons on the form and name it as first, next, previous and last.

Step2: place two labels for representing column names.

Step3: place two text boxes one for getting the values from the database.

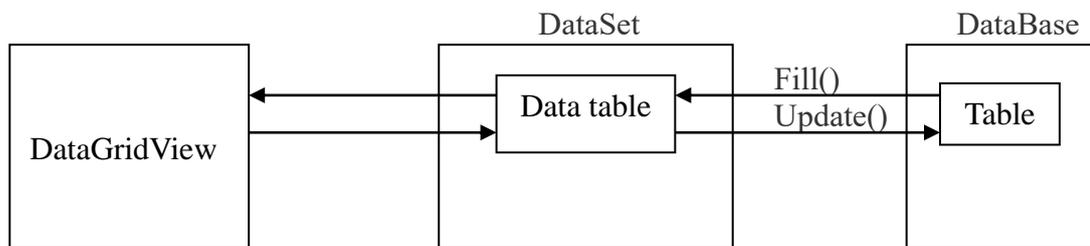
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication25
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection ocon;
        OleDbDataReader dr;
        DataSet ds = new DataSet();
        OleDbDataAdapter da;
        int i, c, n;
        private void Form1_Load(object sender, EventArgs e)
        {
            ocon = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system; password=raja");
            OleDbCommand cmd = new OleDbCommand("select * from
                                                student4", ocon);
            OleDbDataAdapter da = new OleDbDataAdapter("select * from
                                                        student4", ocon);

            ocon.Open();
            dr = cmd.ExecuteReader();
            label1.Text = dr.GetName(0);
        }
    }
}
```

```
        label2.Text = dr.GetName(1);
        da.Fill(ds, "student4");
        while (dr.Read() != false)
        {
            c++;
        }
        n = c-1;
    }
    private void button1_Click(object sender, EventArgs e)
    {
        i = 0;
        textBox1.Text = ds.Tables["student4"].Rows[i][0].ToString();
        textBox2.Text = ds.Tables["student4"].Rows[i][1].ToString();
    }
    private void button2_Click(object sender, EventArgs e)
    {
        if (i < n)
        {
            i++;
            textBox1.Text = ds.Tables["student4"].Rows[i][0].ToString();
            textBox2.Text = ds.Tables["student4"].Rows[i][1].ToString();
        }
        else
        {
            MessageBox.Show("no more records");
        }
    }
    private void button3_Click(object sender, EventArgs e)
    {
        if (i > 0)
        {
            i--;
            textBox1.Text = ds.Tables["student4"].Rows[i][0].ToString();
            textBox2.Text = ds.Tables["student4"].Rows[i][1].ToString();
        }
        else
        {
            MessageBox.Show("no more records");
        }
    }
    private void button4_Click(object sender, EventArgs e)
    {
        i = n;
        textBox1.Text = ds.Tables["student4"].Rows[i][0].ToString();
        textBox2.Text = ds.Tables["student4"].Rows[i][1].ToString();
    }
}
}
```

DataGridView control: The DataGridView control provides a powerful and flexible way to display data in a tabular format. DataGridView control is updatable which allows us to add, modify or delete records. By using DataSource property we can directly bind DataGridView control to the DataTable. The speciality of DataGridView is any changes made on data which is present on DataGridView, reflects directly into the datatable to which it was bound. Control is directly binded to the datatable by using its DataSource property.

Syntax: `DataSet ds = new DataSet();`
`dataGridView1.DataSource=ds.Tables[0];`



Example: create an ADO.NET application in C#, to demonstrate dataGridView Control.

Procedure:

Step1: Place one DataGridView on the form

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication27
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        DataSet dset=new DataSet();
        OleDbConnection ocon;
        private void Form1_Load(object sender, EventArgs e)
        {
            ocon = new OleDbConnection("Provider=MSDAORA.1;user id=system;
                                     password=raja");
            OleDbDataAdapter ad = new OleDbDataAdapter(" select * from student4
                                                         ", ocon);
            ad.Fill(dset,"student4");
        }
    }
}
  
```

```

        dataGridView1.DataSource = dset.Tables["student4"];
    }
}

```

Example: create an ADO.NET application in C#, that impliments the following tasks using dataGridView object.

- (i) Insert a row
- (ii) Delete a row
- (iii) Update a row

Procedure:

Step1: Place one DataGridView on the form

Step2: Place three buttons on the form and name it as insert, update and delete.

Step3: Place two labels on the form which represents the column names.

Step4: Place two textboxes for taking the values from the user.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication28
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            DisplayData();
        }
        OleDbConnection con= new OleDbConnection("Provider=MSDAORA.1;
                                                    user id=system;password=raja");
        OleDbCommand cmd;
        OleDbDataAdapter adapt;
        //ID variable used in Updating and Deleting Record
        int ID = 0;
        private void DisplayData()
        {
            con.Open();
            DataTable dt=new DataTable();
            adapt=new OleDbDataAdapter("select * from student4",con);
            adapt.Fill(dt);
            dataGridView1.DataSource = dt;
            con.Close();
        }
        private void ClearData()
        {

```

```
        textBox1.Text = "";
        textBox2.Text = "";
        ID = 0;
    }
    private void dataGridView1_RowHeaderMouseClick(object sender,
                                                    DataGridViewCellMouseEventArgs e)
    {
        ID = Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString());
        textBox1.Text = dataGridView1.Rows[e.RowIndex].Cells[0].Value.ToString();
        textBox2.Text = dataGridView1.Rows[e.RowIndex].Cells[1].Value.ToString();
    }
    private void button1_Click(object sender, EventArgs e)
    {
        if (textBox1.Text != "" && textBox2.Text != "")
        {
            cmd = new OleDbCommand("insert into student4
                values '"+textBox1.Text+"','"+textBox2.Text+"'", con);
            con.Open();
            cmd.ExecuteNonQuery();
            con.Close();
            MessageBox.Show("Record Inserted Successfully");
            DisplayData();
            ClearData();
        }
        else
        {
            MessageBox.Show("Please Provide Details!");
        }
    }
    private void button2_Click(object sender, EventArgs e)
    {
        if (textBox1.Text != "" && textBox2.Text != "")
        {
            cmd = new OleDbCommand("update student4 set
                SNAME='"+textBox2.Text+"' where SID='"+textBox1.Text+"", con);
            con.Open();
            cmd.ExecuteNonQuery();
            MessageBox.Show("Record Updated Successfully");
            con.Close();
            DisplayData();
            ClearData();
        }
        else
        {
            MessageBox.Show("Please Select Record to Update");
        }
    }
    private void button3_Click(object sender, EventArgs e)
    {
        if (ID != 0)
```

```

        {
            cmd = new OleDbCommand("delete from student4 where
                                   SID="+ID+"", con);
            con.Open();
            cmd.ExecuteNonQuery();
            MessageBox.Show("Record Deleted Successfully!");
            con.Close();
            DisplayData();
            ClearData();
        }
        else
        {
            MessageBox.Show("Please Select Record to Delete");
        }
    }
}
}
}

```

DataView:

- The DataView provides **different views of the data** stored in a DataTable.
- DataView can be used to **sort, filter, and search** the data in a DataTable, additionally we can **add new rows, modify the data in existing rows** and delete **the content** in a DataTable .
- Whatever Changes made to a DataView affect the underlying DataTable automatically, and changes made to the underlying DataTable automatically affect any DataView objects that are viewing the DataTable.
- We can **create DataView in two different ways**. We can use the **DataView Constructor**, or you can **create a reference to the DefaultView Property of the DataTable**.
- The DataView constructor can be empty, or it can take either a DataTable as a single argument, or a DataTable along with filter criteria, sort criteria, and a row state filter.
- **Creating Dataview object as**

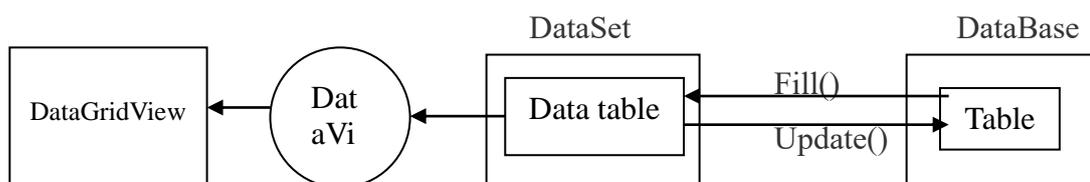
```

Dataview dv=new Dataview();
dv = ds.Tables[0].DefaultView;

```

- DataView filters the data with the help of RowFilter property.

Syntax: `dv.RowFilter="value";`



Example: create an ADO.NET application in C# to demonstrate the DataView object(this program shows selected records).

Procedure:

Step1: place combobox on form.

Step2: place the DataGridView control on the form.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication30
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection con;
        OleDbDataAdapter da;
        DataSet ds;
        private void Form1_Load(object sender, EventArgs e)
        {
            con = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system;password=raja");
            da = new OleDbDataAdapter("select * from student4", con);
            ds = new DataSet();
            da.Fill(ds, "student4");
            dataGridView1.DataSource = ds.Tables["student4"].DefaultView;
            comboBox1.DataSource = ds.Tables["student4"];
            comboBox1.DisplayMember = "SID";
        }
        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            DataView dv = new DataView();
            dv = ds.Tables["student4"].DefaultView;
            dv.RowFilter = "SID=7";
        }
    }
}

```

Example: create an ADO.NET application in C# to demonstrate the DataView object (this program sorts the table).

Procedure:

Step1: place combobox on form.

Step2: place the DataGridView control on the form.

```

using System;
using System.Collections.Generic;
using System.ComponentModel;

```

```

using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication30
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection con;
        OleDbDataAdapter da;
        DataSet ds;
        private void Form1_Load(object sender, EventArgs e)
        {
            con = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system;password=raja");
            da = new OleDbDataAdapter("select * from student4", con);
            ds = new DataSet();
            da.Fill(ds, "student4");
            dataGridView1.DataSource = ds.Tables["student4"].DefaultView;
            comboBox1.DataSource = ds.Tables["student4"];
            comboBox1.DisplayMember = "SID";
        }
        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            DataView dv = new DataView();
            dv = ds.Tables["student4"].DefaultView;
            dv.Sort = "SID";
        }
    }
}

```

How to Search record in a DataView: We can search in a DataView according to the sort key values by using the **Find method**. The **Find method returns an integer with the index of the DataRowView that matches the search criteria**. If more than one row matches the search criteria, **only the index of the first matching DataRowView is returned**. If no matches are found, Find returns -1

Syntax: `int index = dv.Find("PRODUCT5");`

Example: create an ADO.NET application to search record in a DataView.

Procedure:

Step1:place a DataGridView and a Button on Form.

```

using System;
using System.Collections.Generic;

```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication30
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbDataAdapter adapter;
        OleDbConnection con;
        OleDbCommand command
        private void button1_Click(object sender, EventArgs e)
        {
            con = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system;password=raja");
            DataSet ds = new DataSet();
            DataView dv;
            con.Open();
            command = new OleDbCommand("select * from student4", con);
            adapter = new OleDbDataAdapter("select * from student4", con);
            adapter.Fill(ds, "Find Row DataView");
            con.Close();
            dv = new DataView(ds.Tables[0]);
            dv.Sort = "SID";
            int index = dv.Find("8");
            if (index == -1)
            {
                MessageBox.Show("Item Not Found");
            }
            else
            {
                MessageBox.Show(dv[index]["SID"].ToString() + " " +
                                dv[index]["SNAME"].ToString());
            }
        }
    }
}
```

How to add new row in a DataView: We can add new rows in the DataView using **AddNew method** in the DataView.

Example: create an ADO.NET application to add new row in a DataView.

Procedure:

Step1: place a DataGridView and a Button on Form.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication30
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        OleDbConnection con;
        DataView dv;
        OleDbCommand command;
        OleDbDataAdapter adapter
        private void button1_Click(object sender, EventArgs e)
        {
            con = new OleDbConnection("Provider=MSDAORA.1;
                                     user id=system;password=raja");

            DataSet ds = new DataSet();
            con.Open();
            command = new OleDbCommand("select * from student4", con);
            adapter = new OleDbDataAdapter("select * from student4", con);
            adapter.Fill(ds, "Add New");
            con.Close();
            dv = new DataView(ds.Tables[0]);
            DataRowView newRow = dv.AddNew();
            newRow["SID"] = 11;
            newRow["SNAME"] = "Product 7";
            newRow.EndEdit();
            dv.Sort = "SID";
            dataGridView1.DataSource = dv;
        }
    }
}
```

How to update row in a DataView:**Example: create an ADO.NET application to update row in a DataView.****Procedure:****Step1:place a DataGridView and a Button on Form.**

```

using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Data.OleDb;
namespace WindowsFormsApplication30
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            OleDbConnection con = new OleDbConnection("Provider=MSDAORA.1;user
id=system;password=raja");
            OleDbDataAdapter adapter = new OleDbDataAdapter();
            DataSet ds = new DataSet();
            DataView dv;
            con.Open();
            OleDbCommand command = new OleDbCommand("select * from student4", con);
            adapter.SelectCommand = command;
            adapter.Fill(ds, "Update");
            con.Close();
            dv = new DataView(ds.Tables[0], "", "SID", DataViewRowState.CurrentRows);
            int index = dv.Find("8");
            if (index == -1)
            {
                MessageBox.Show("Product not found");
            }
            else
            {
                dv[index]["SNAME"] = "Product11";
                MessageBox.Show("Product Updated !");
            }
            dataGridView1.DataSource = dv;
        }
    }
}

```

HOW TO DELETE ROW IN A DATAVIEW:

Example: create an ADO.NET application to delete row in a DataView.

Procedure:

Step1:place a DataGridView and a Button on Form.

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Windows.Forms;
namespace WindowsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            OleDbConnection con = new
                OleDbConnection("Provider=MSDAORA.1;
                    user id=system;password=raja");
            OleDbDataAdapter adapter = new OleDbDataAdapter();
            DataSet ds = new DataSet();
            DataView dv;
            con.Open();
            OleDbCommand command = new OleDbCommand("select *
                from student4", con);
            adapter.SelectCommand = command;
            adapter.Fill(ds, "Delete Row");
            adapter.Dispose();
            command.Dispose();
            dv = new DataView(ds.Tables[0], "", "SID",
                DataViewRowState.CurrentRows);
            dv.Table.Rows[3].Delete();
            dataGridView1.DataSource = dv;
        }
    }
}
```

UNIT IV

HTML:

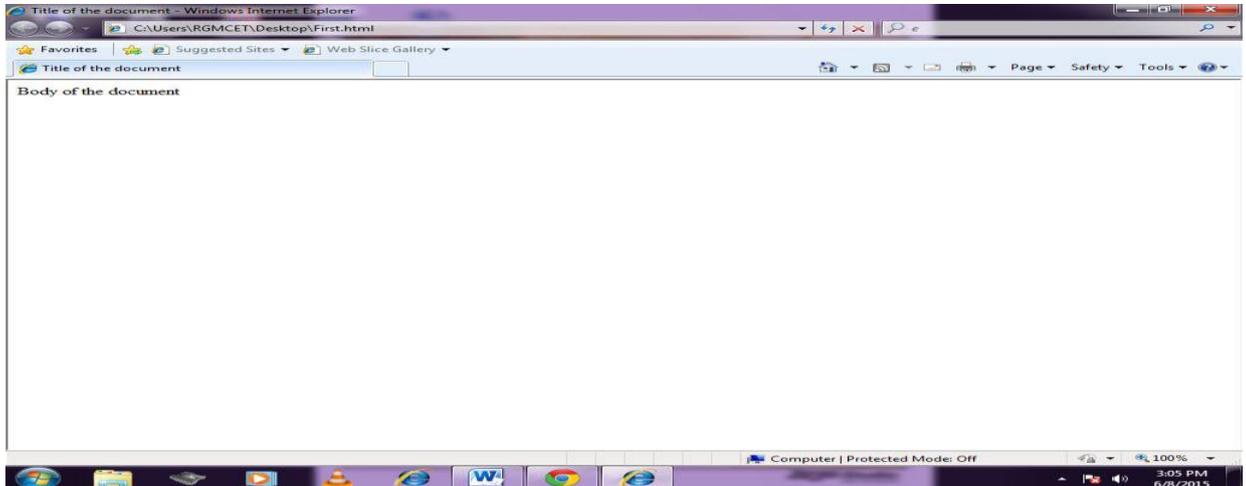
- HTML stands **for HyperText Markup Language**
 - Hypertext refers to **the way in which Web pages (HTML documents) are linked together**. Thus the link available on a webpage is called **Hypertext**.
 - HTML is a Markup Language which means you use **HTML to simply "markup" a text document with tags that tell a Web browser how to structure it to display**.
- HTML is used to **create static webpages**.
- HTML is a **presentation language**.
- HTML is **not a case-sensitive**.
- It is only a **formatting language** and not a programming language.
- It is mainly used for **describing structured documents**.
- HTML documents are **cross platform and device independent**. We need only a HTML readable browser to view them
- The **W3C (World Wide Web Consortium) and the WHATWG (Web Hypertext Application Technology Working Group) maintains the HTML (HyperText Markup Language) international standards and specifications**.

Structure of HTML Program:

```

<html>
  <head>
    <title> Title of the document</title>
  </head>
  <body> Body of the document </body>
</html>

```

Output:

Basic HTML Tags: As we have seen earlier, HTML is a markup language and makes use of various tags to format the content. These tags are enclosed within **angle braces** <Tag Name>. Except few tags, most of the tags have their corresponding closing tags. For example <html> has its closing tag </html> and <body> tag has its closing tag </body> tag etc.

<!DOCTYPE...> TAG: This tag defines the **document type and HTML version**.

Syntax: <!DOCTYPE html> for html 5 and above versions

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">

```

<HTML>...</HTML> TAG: This tag is used to tell the browser that everything contained between them is a HTML code for creating a webpage. Every html document starts with a <html> tag and ends with </html>.

<HEAD>..</HEAD>: This tag contains the head of an html document. This tag holds information about the document, such as <title>, <link> ,<meta>.

<TITLE>..</TITLE> TAG: This tag gives the title of a web page. This title appears in the web browsers title bar and is used by search engines to refer the document.

<BODY>..</BODY> TAG: This tag contains the body of the html document, which includes creating web page contents like text, images, links, forms etc. that appear in the web browser.

COMMENT TAG: This tag is used to write comments in the html source code. The comments that are written in the html documentation are not executed but they are used for user

understandability.

Single line comment tag

```
<!-- this is a single line comment tag.-->
```

Multiple line comment tag:

```
<!--this is a multiple line comment tag.--
--comment continues with two hyphen--
-- comments ends-->
```

HEADER TAG: These tags are used for printing headings. The tags are <h1>,<h2>,<h3>,<h4>,<h5> and <h6>. These tags will display bold text in different sizes.<h1> is the highest level heading and <h6> is the lowest level heading.

Example:

```
<html>
  <head><title>Heading Example</title></head>
  <body>
    <h1>This is heading 1</h1>
    <h2>This is heading 2</h2>
    <h3>This is heading 3</h3>
    <h4>This is heading 4</h4>
    <h5>This is heading 5</h5>
    <h6>This is heading 6</h6>
  </body>
</html>
```

Font tag: Font tag sets textsize, color and style.

Attributes:

- COLOR
- SIZE
- STYLE or FACE

Example: here write your own text

TEXT FORMATTING TAGS:

BOLD TAG: It displays the bold text.

Syntax:

Example:

```
<html>
  <head>
    <title>Bold Text Example</title>
  </head>
  <body>
    <p>The following word uses a <b>bold</b> typeface.</p>
  </body>
</html>
```

Output: The following word uses a bold typeface.

ITALIC TAG: It displays the text in italic form**Syntax:** <i>.....</i>

Example: <html>
 <head>
 <title>Italic Text Example</title>
 </head>
 <body>
 <p>The following word uses a <i>italicized</i> typeface.</p>
 </body>
 </html>

Output: The following word uses a italicized typeface.**STRIKEOUT TAG: It displays striked text.****Syntax:** <s>.....</s> or <strike>.....</strike>

Example: <html>
 <head>
 <title>Strike Text Example</title>
 </head>
 <body>
 <p>The word uses a <strike>strikethrough</strike> typeface.</p>
 </body>
 </html>

Output: The following word uses a ~~strikethrough~~ typeface.**Inserted Text:** Anything that appears within <ins>...</ins> element is displayed as inserted text.

Example: <html>
 <head>
 <title>Inserted Text Example</title>
 </head>
 <body>
 <p>I want to drink cola <ins>wine</ins></p>
 </body>
 </html>

Output: I want to drink ~~cola~~ wine**Deleted Text tag:** Anything that appears within ... element, is displayed as deleted text.

Example: <html>
 <head>
 <title>Deleted Text Example</title>
 </head>
 <body>
 <p>I want to drink cola <ins>wine</ins></p>
 </body>
 </html>

Output: I want to drink ~~cola~~ wine**UNDERLINE TAG: it displays underlined text.****Syntax:** <u>.....</u>

Example <html>
 <head>
 <title>Underlined Text Example</title>
 </head>

```

        <body>
            <p>The word uses a <u>underlined</u> typeface.</p>
        </body>
    </html>

```

Output: The following word uses a underlined typeface.

<BIG> AND <SMALL> TAG: this tags makes the text to look bigger and smaller.

Syntax: <big>.....</big>

<small>.....</small>

Example:

```

<html>
    <head>
        <title>Larger Text Example</title>
    </head>
    <body>
        <p>The following word uses a <big>big</big> typeface.</p>
    </body>
</html>

```

Output: The following word uses a big typeface.

Example:

```

<html>
    <head>
        <title>Smaller Text Example</title>
    </head>
    <body>
        <p>The word uses a <small>small</small> typeface.</p>
    </body>
</html>

```

Output: The following word uses a small typeface.

STRONG TAG: It displays strongly emphasized text.

Syntax:

Example:

```

<html>
    <head>
        <title>Strong Text Example</title>
    </head>
    <body>
        <p>The following word uses a <strong>strong</strong> typeface.</p>
    </body>
</html>

```

Output: The following word uses a **strong** typeface.

CENTER TAG: It displays the text at the Center of the web page.

Syntax: <center>.....</center>

<SUP> TAG: It displays the super script text.

Syntax: ^{....}

Example: this formula is = (a+b)²

Example:

```

<html>
    <head>
        <title>Superscript Text Example</title>

```

```

</head>
<body>
  <p>The word uses a <sup>superscript</sup> typeface.</p>
</body>
</html>

```

Output: The following word uses a ^{superscript} typeface.

<SUB> TAG: it displays subscript text.

Syntax: _{...}

Example: this formula is = h₂o₂

Example:

```

<html>
  <head><title>Subscript Text Example</title></head>
  <body>
    <p>The following word uses a <sub>subscript</sub> typeface.</p>
  </body>
</html>

```

Output: The following word uses a subscript typeface.

<MARQUEE> TAG: This tag displays the scrolling text.

Syntax: <marquee>.....</marquee>

ATTRIBUTES:

- **ALIGN:** sets the alignment of the text to **TOP (by default), MIDDLE or BOTTOM.**
- **BEHAVIOUR:** Its shows how the text in the marquee tag should move. It can **SCROLL (by default): the text , SLIDE (text enters from one side and stops at the other end), or ALTERNATE (text seems to bounce from one side to another side)**
- **BGCOLOR:** sets the background colour for the marquee box.
- **DIRECTION:** sets the direction of the text for scrolling. It can **LEFT (by default), RIGHT, DOWN or UP).**

Example: <MARQUEE ALIGN="MIDDLE" BEHAVIOUR="ALTERNATE" BGCOLOR="BLUE" DIRECTION="UP">this is an example of marquee tag</MARQUEE>

BLINK TAG: it displays blink text.

Syntax: <blink> text blinks</blink>

LINE BREAK TAG: It inserts a line break into a page. This tag did not have any closing tag.

Syntax:

PARAGRAPH TAG: It displays the text in a paragraph format.

Syntax: <p>.....</p>

ATTRIBUTES:

- **ALIGN:** sets the alignment of text in the paragraph where it sets the para either **LEFT (the default), RIGHT, CENTER or JUSTIFY.**

Example: <p ALIGN=CENTER>.....</p>

<div> tag: This tag selects a block of text for applying styles that is left, right and center.

Syntax: `<div align="center">.....</div>`

Text Abbreviation tag: You can abbreviate a text by putting it inside opening `<abbr>` and closing `</abbr>` tags. If present, the title attribute must contain this full description and nothing else.

Example

```
<html>
  <head>
    <title>Text Abbreviation</title>
  </head>
  <body>
    <p>My friend's name is <abbr title="Abhishek">Abhy</abbr>.</p>
  </body>
</html>
```

Output: My best friend's name is Abhy.

HTML Lists: HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. There are three kinds of lists:

1. Unordered List.
2. Ordered List.
3. Definition List

HTML Unordered Lists: An unordered list is a collection of related items that have no special order or sequence. We create unordered list with element `` and items of lists are created using ``. Important attribute in unordered list is **“type”** that customizes unordered lists. Each item in the list is marked with a bullet.

Example:

```
<html>
  <head>
    <title>HTML Unordered List</title>
  </head>
  <body>
    <ul>
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ul>
  </body>
</html>
```

Output:

- Beetroot
- Ginger
- Potato
- Radish

The type Attribute: You can use type attribute for tag to specify the type of bullet you like. **By default it is a disc.** Following are the possible options:

```
<ul type="square">
<ul type="disc">
<ul type="circle">
```

Example: Following is an example where we used <ul type="square">

```
<html>
    <head>
        <title>HTML Unordered List</title>
    </head>
    <body>
        <ul type="square">
            <li>Beetroot</li>
            <li>Ginger</li>
            <li>Potato</li>
            <li>Radish</li>
        </ul>
    </body>
</html>
```

Output:

- Beetroot
- Ginger
- Potato
- Radish

Example: Following is an example where we used <ul type="disc"> :

```
<html>
    <head>
        <title>HTML Unordered List</title>
    </head>
    <body>
        <ul type="disc">
            <li>Beetroot</li>
            <li>Ginger</li>
            <li>Potato</li>
            <li>Radish</li>
        </ul>
    </body>
</html>
```

Output:

- Beetroot
- Ginger
- Potato
- Radish

Example: Following is an example where we used <ul type="circle"> :

```
<html>
    <head>
        <title>HTML Unordered List</title>
    </head>
    <body>
```

```

        <ul type="circle">
            <li>Beetroot</li>
            <li>Ginger</li>
            <li>Potato</li>
            <li>Radish</li>
        </ul>
    </body>
</html>

```

Output:

- Beetroot
- Ginger
- Potato
- Radish

HTML Ordered Lists: If you are required to put your items in a numbered list instead of bulleted then HTML ordered list will be used. This list is created by using **** tag **** tag for list items. The numbering starts at one and is incremented by one for each successive ordered list element tagged with ****.

Example:

```

<html>
    <head>
        <title>HTML Ordered List</title>
    </head>
    <body>
        <ol>
            <li>Beetroot</li>
            <li>Ginger</li>
            <li>Potato</li>
            <li>Radish</li>
        </ol>
    </body>
</html>

```

Output:

1. Beetroot
2. Ginger
3. Potato
4. Radish

The type Attribute:

You can use type attribute for **** tag to specify the **type of numbering you like**. By default it is a number. Following are the possible options:

- <ol type="1"> - Default-Case Numerals.
- <ol type="I"> - Upper-Case Numerals.
- <ol type="i"> - Lower-Case Numerals.
- <ol type="a"> - Lower-Case Letters.
- <ol type="A"> - Upper-Case Letters.

Example: Following is an example where we used **<ol type="1">**

```

<html>

```

```
<head><title>HTML Ordered List</title></head>
<body>
  <ol type="1">
    <li>Beetroot</li>
    <li>Ginger</li>
    <li>Potato</li>
    <li>Radish</li>
  </ol>
</body>
</html>
```

Output:

1. Beetroot
2. Ginger
3. Potato
4. Radish

Example: Following is an example where we used <ol type="I">

```
<html>
  <head>
    <title>HTML Ordered List</title>
  </head>
  <body>
    <ol type="I">
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ol>
  </body>
</html>
```

Output:

- I. Beetroot
- II. Ginger
- III. Potato
- IV. Radish

Example: Following is an example where we used <ol type="i">

```
<html>
  <head>
    <title>HTML Ordered List</title>
  </head>
  <body>
    <ol type="i">
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ol>
  </body>
</html>
```

Output:

- i. Beetroot
- ii. Ginger
- iii. Potato
- iv. Radish

Example: Following is an example where we used <ol type="A">

```
<html>
    <head>
        <title>HTML Ordered List</title>
    </head>
    <body>
        <ol type="A">
            <li>Beetroot</li>
            <li>Ginger</li>
            <li>Potato</li>
            <li>Radish</li>
        </ol>
    </body>
</html>
```

Output:

- A. Beetroot
- B. Ginger
- C. Potato
- D. Radish

Example: Following is an example where we used <ol type="a">

```
<html>
    <head>
        <title>HTML Ordered List</title>
    </head>
    <body>
        <ol type="a">
            <li>Beetroot</li>
            <li>Ginger</li>
            <li>Potato</li>
            <li>Radish</li>
        </ol>
    </body>
</html>
```

Output:

- a. Beetroot
- b. Ginger
- c. Potato
- d. Radish

The start Attribute: You can use start attribute for tag to specify the starting point of numbering you need. **Following are the possible options:**

- <ol type="1" start="4"> - Numerals starts with 4.
- <ol type="I" start="4"> - Numerals starts with IV.
- <ol type="i" start="4"> - Numerals starts with iv.
- <ol type="a" start="4"> - Letters starts with d.
- <ol type="A" start="4"> - Letters starts with D.

Example: Following is an example where we used `<ol type="i" start="4" >`

```
<html>
  <head>
    <title>HTML Ordered List</title>
  </head>
  <body>
    <ol type="i" start="4">
      <li>Beetroot</li>
      <li>Ginger</li>
      <li>Potato</li>
      <li>Radish</li>
    </ol>
  </body>
</html>
```

Output:

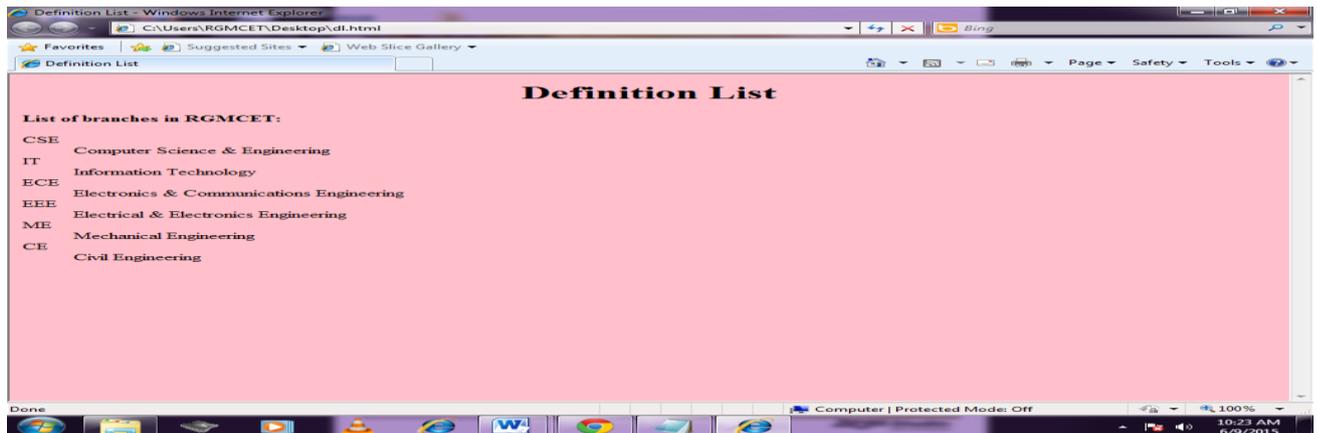
- iv. Beetroot
- v. Ginger
- vi. Potato
- vii. Radish

Definition Lists: The definition list includes both **terms and their definitions**. We use `<dl>` element to create definition lists, `<dt>` element for definition terms and `<dd>` element for defining definition of each term.

Example:

```
<html>
  <head>
    <title>Definition List</title>
  </head>
  <body bgcolor="pink">
    <h1 align="center">Definition List</h1>
    <h4>List of branches in RGM CET:</h4>
    <dl>
      <dt>CSE<dd>Computer Science & Engineering
      <dt>IT<dd>Information Technology
      <dt>ECE<dd>Electronics & Communications Engineering
      <dt>EEE<dd>Electrical & Electronics Engineering
      <dt>ME<dd>Mechanical Engineering
      <dt>CE<dd>Civil Engineering
    </dl>
  </body>
</html>
```

Output:

**Example**

```

<html>
  <head>
    <title>HTML Definition List</title>
  </head>
  <body>
    <dl>
      <dt><b>HTML</b></dt>
      <dd>This stands for Hyper Text Markup Language</dd>
      <dt><b>HTTP</b></dt>
      <dd>This stands for Hyper Text Transfer Protocol</dd>
    </dl>
  </body>
</html>

```

Output:

HTML

This stands for Hyper Text Markup Language

HTTP

This stands for Hyper Text Transfer Protocol

Creating Hyperlinks:

- The main **purpose of hyperlinks is to provide navigation from one page to another page (or) within the page** i.e., by using hyperlinks we are going to link one page with other. We create hyperlinks by using “Anchor” (<a>) tag.
- The main form of hyperlink is text hyperlink, where the text which we define as hyperlink usually appears underlined and in a different color from surrounding text. In addition to text we can also use images as hyperlinks. This is done by putting image tag () in anchor tag instead of text. The hyperlink format is defined as follows:

Syntax: Hyperlink Text

Attributes:

HREF: This attribute **defines the link address**

Example: click here to visit w3schools

TARGET: This attribute defines where the linked document will be opened.

Option	Description
_blank	Opens the linked document in a new window or tab.
_self	Opens the linked document in the same frame.
_parent	Opens the linked document in the parent frame.
_top	Opens the linked document in the full body of the window.

Example: `click here to visit w3schools`

NAME: when name attribute is used the `<a>` element defines a named anchor inside an html document.

Example:

```
<html>
  <head>
    <title>Creating Hyperlinks</title>
  </head>
  <body bgcolor="pink">
    <center>
      <h1>Creating Hyperlinks</h1>
    </center>
    <b>This is Hyperlink Page<b>
    <a href="dl1.html">Click Here</a><b>to goto Definition List Page
  </body>
</html>
```

Output:



HTML Images:

- In HTML we have the capability of displaying images in our web pages. The images must be in a format that the browsers can handle, **such as GIF (or) JPEG** and for **some browsers PNG formats**.
- There are **plenty of graphic file formats in existence**, but we can't count on browsers knowing more than

two standard formats i.e., **GIF and JPEG**.

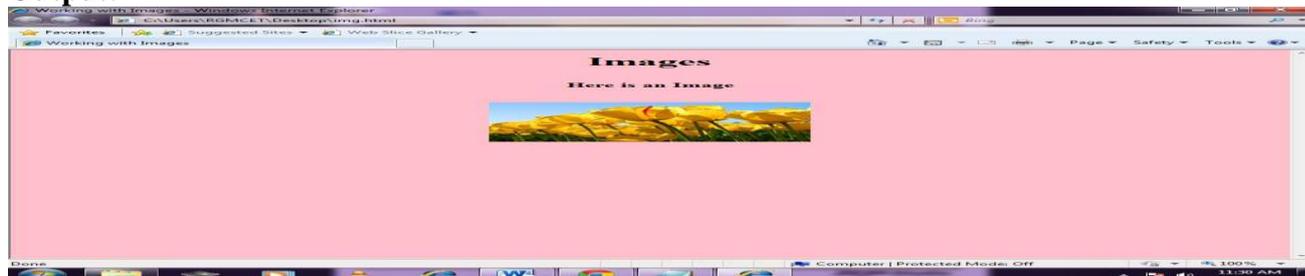
- Displaying of images in web pages is done by using **image (tag**.
- Some browsers may or may not support graphics i.e., they may not be able to display images. For such browsers we just provide an alternative that is we just give a text in place of image by using “**alt**” **attribute of image tag**. Format of tag is:

Attribute	Value	Description
align	top bottom middle left right	Specifies the alignment for the image.
alt	text	It is used to specify the text to be displayed in place of images for browsers that cannot handle graphics.
border	pixels	Specifies the width of the image border.
height	pixels or %	Specifies the height of the image.
src	URL	The url of an image

Example:

```
<html>
  <head>
    <title>Working with Images</title>
  </head>
  <body bgcolor=“pink”>
    <center>
      <h1> Images</h1><br>
      <h3> Here is an Image</h3><br>
      <img src=“C:\Users\Public\Pictures\Sample Pictures\Tulips.jpg” width=“252”
        height=“110” alt=“Here is an image”>
    </center>
  </body>
</html>
```

Output:



HTML Tables: The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

- Table is created using **<table>** tag.

- Rows of table are created using `<tr>` tag.
- To insert data into the table we use `<td>` tag.
- To create table headings `<th>` tag is used.
- Format for creating a table is:

```
<table>
  <tr>
    <th>Table Heading</th>
    <th>Table Heading</th>
  </tr>
  <tr>
    <td>Table Data</td>
    <td>Table Data</td>
  </tr>
</table>
```

Example

```
<html>
  <head>
    <title>HTML Tables</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <td>Row 1, Column 1</td>
        <td>Row 1, Column 2</td>
      </tr>
      <tr>
        <td>Row 2, Column 1</td>
        <td>Row 2, Column 2</td>
      </tr>
    </table>
  </body>
</html>
```

Output:

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

Here border is an attribute of `<table>` tag and it is used to put a border across all the cells. If you do not need a border then you can use `border="0"`.

Table Heading: Table heading can be defined using `<th>` tag. This tag will be put to replace `<td>` tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use `<th>` element in any row.

Example

```
<html>
  <head>
    <title>HTML Table Header</title>
```

```

</head>
<body>
  <table border="1">
    <tr>
      <th>Name</th>
      <th>Salary</th>
    </tr>
    <tr>
      <td>Ramesh Raman</td>
      <td>5000</td>
    </tr>
    <tr>
      <td>Shabbir Hussein</td>
      <td>7000</td>
    </tr>
  </table>
</body>
</html>

```

output:

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

Cellpadding and Cellspacing Attributes: There are two attributes called cellpadding and cellspacing which you will **use to adjust the white space in your table cells**. The **cellspacing** attribute defines the width of the border, while **cellpadding** represents the distance between cell borders and the content within a cell.

Example

```

<html>
  <head>
    <title>HTML Table Cellpadding</title>
  </head>
  <body>
    <table border="1" cellpadding="5" cellspacing="5">
      <tr>
        <th>Name</th>
        <th>Salary</th>
      </tr>
      <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
      </tr>
      <tr>
        <td>Shabbir Hussein</td>
        <td>7000</td>
      </tr>
    </table>
  </body>
</html>

```

```

        </body>
    </html>

```

Output:

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

Colspan and Rowspan Attributes : You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

Example

```

<html>
  <head>
    <title>HTML Table Colspan/Rowspan</title>
  </head>
  <body>
    <table border="1">
      <tr>
        <th>Column 1</th>
        <th>Column 2</th>
        <th>Column 3</th>
      </tr>
      <tr>
        <td rowspan="2">Row 1 Cell 1</td>
        <td>Row 1 Cell 2</td>
        <td>Row 1 Cell 3</td>
      </tr>
      <tr>
        <td>Row 2 Cell 2</td>
        <td>Row 2 Cell 3</td>
      </tr>
      <tr>
        <td colspan="3">Row 3 Cell 1</td>
      </tr>
    </table>
  </body>
</html>

```

Output:

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Tables Backgrounds: You can set table background using one of the following two ways:

bgcOLOR attribute : You can set background color for whole table or just for one cell.

background attribute : You can set background image for whole table or just for one cell.

You can also set border color also using bordercolor attribute.

Example

```
<html>
  <head>
    <title>HTML Table Background</title>
  </head>
  <body>
    <table border="1" bordercolor="green" bgcolor="yellow">
      <tr>
        <th>Column 1</th>
        <th>Column 2</th>
        <th>Column 3</th>
      </tr>
      <tr>
        <td rowspan="2">Row 1 Cell 1</td>
        <td>Row 1 Cell 2</td>
        <td>Row 1 Cell 3</td>
      </tr>
      <tr>
        <td>Row 2 Cell 2</td>
        <td>Row 2 Cell 3</td>
      </tr>
      <tr>
        <td colspan="3">Row 3 Cell 1</td>
      </tr>
    </table>
  </body>
</html>
```

Output:

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Example:

```
<html>
  <head>
    <title>HTML Table Background</title>
  </head>
  <body>
    <table border="1" bordercolor="green" background="/images/test.png">
      <tr>
        <th>Column 1</th>
        <th>Column 2</th>
        <th>Column 3</th>
      </tr>
      <tr>
```

```

        <td rowspan="2">Row 1 Cell 1</td>
        <td>Row 1 Cell 2</td>
        <td>Row 1 Cell 3</td>
    </tr>
    <tr>
        <td>Row 2 Cell 2</td>
        <td>Row 2 Cell 3</td>
    </tr>
    <tr>
        <td colspan="3">Row 3 Cell 1</td>
    </tr>
</table>
</body>
</html>

```

Height and Width attributes: You can set a table width and height using width and height attributes. You can specify table width or height in **terms of pixels or in terms of percentage of available screen area.**

Example:

```

<html>
  <head>
    <title>HTML Table Width/Height</title>
  </head>
  <body>
    <table border="1" width="400" height="150">
      <tr>
        <td>Row 1, Column 1</td>
        <td>Row 1, Column 2</td>
      </tr>
      <tr>
        <td>Row 2, Column 1</td>
        <td>Row 2, Column 2</td>
      </tr>
    </table>
  </body>
</html>

```

Output:

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

Table Caption: The caption tag will **serve as a title** or explanation for the table and it shows up at the top of the table.

Example:

```

<html>
  <head>
    <title>HTML Table Caption</title>
  </head>
  <body>
    <table border="1" width="100%">
      <caption>This is the caption</caption>
    </table>
  </body>
</html>

```

```

        <tr>
            <td>row 1, column 1</td><td>row 1, columnn 2</td>
        </tr>
        <tr>
            <td>row 2, column 1</td><td>row 2, columnn 2</td>
        </tr>
    </table>
</body>
</html> Output:

```

This is the caption	
row 1, column 1	row 1, columnn 2
row 2, column 1	row 2, columnn 2

Nested Tables: You can use one table inside another table. Not only tables you can use almost all the tags inside table data tag <td>.

Example: Following is the example of using another table and other tags inside a table cell.

```

<html>
    <head>
        <title>HTML Table</title>
    </head>
    <body>
        <table border="1" width="100%">
            <tr>
                <td>
                    <table border="1" width="100%">
                        <tr>
                            <th>Name</th>
                            <th>Salary</th>
                        </tr>
                        <tr>
                            <td>Ramesh Raman</td>
                            <td>5000</td>
                        </tr>
                        <tr>
                            <td>Shabbir Hussein</td>
                            <td>7000</td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>
    </body>
</html>

```

Output:

Name	Salary
Ramesh Raman	5000

Shabbir Hussein

7000

HTML Frames: HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset.

Creating Frames

<frameset>: <frameset> tag defines how to divide the window into frames. The rows attribute of <frameset> tag defines horizontal frames and cols attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Example: Following is the example to create three horizontal frames:

```
<html>
  <head>
    <title>HTML Frames</title>
  </head>
  <frameset rows="10%,80%,10%">
    <frame name="top" src="/html/top_frame.htm" />
    <frame name="main" src="/html/main_frame.htm" />
    <frame name="bottom" src="/html/bottom_frame.htm" />
  </frameset>
</html>
```

Output:



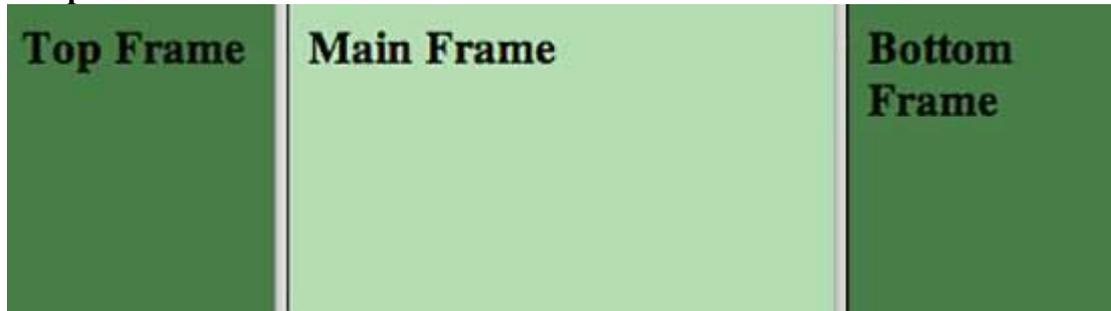
Example: Let's put above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically:

```
<html>
  <head>
    <title>HTML Frames</title>
  </head>
  <frameset cols="25%,50%,25%">
    <frame name="left" src="/html/top_frame.html" />
    <frame name="center" src="/html/main_frame.html" />
  </frameset>
</html>
```

```

        <frame name="right" src="/html/bottom_frame.html" />
        <body> Your browser does not support frames.</body>
    </frameset>
</html>

```

Output:**The <frameset> Tag Attributes**

Attribute	Description
cols	<p>specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of four ways:</p> <ul style="list-style-type: none"> • Absolute values in pixels. For example to create three vertical frames, use cols="100, 500,100". • A percentage of the browser window. For example to create three vertical frames, use cols="10%, 80%,10%". • Using a wildcard symbol. For example to create three vertical frames, use cols="10%, *,10%". In this case wildcard takes remainder of the window. • As relative widths of the browser window. For example to create three vertical frames, use cols="3*,2*,1*". This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.
rows	<p>This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example to create two horizontal frames, use rows="10%, 90%". You can specify the height of each row in the same way as explained above for columns.</p>
border	<p>This attribute specifies the width of the border of each frame in pixels. For example border="5". A value of zero means no border.</p>
frameborder	<p>This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example frameborder="0" specifies no border.</p>
framespacing	<p>This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing="10" means there should be 10 pixels spacing between each frames.</p>

The <frame> Tag Attributes

Attribute	Description
src	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src="/html/top_frame.htm" will load an HTML file available in html directory.
name	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame , in which case the second frame needs a name to identify itself as the target of the link.
frameborder	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
marginwidth	This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth="10".
marginheight	This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight="10".
noresize	By default you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize="noresize".
scrolling	This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling="no" means it should not have scroll bars.

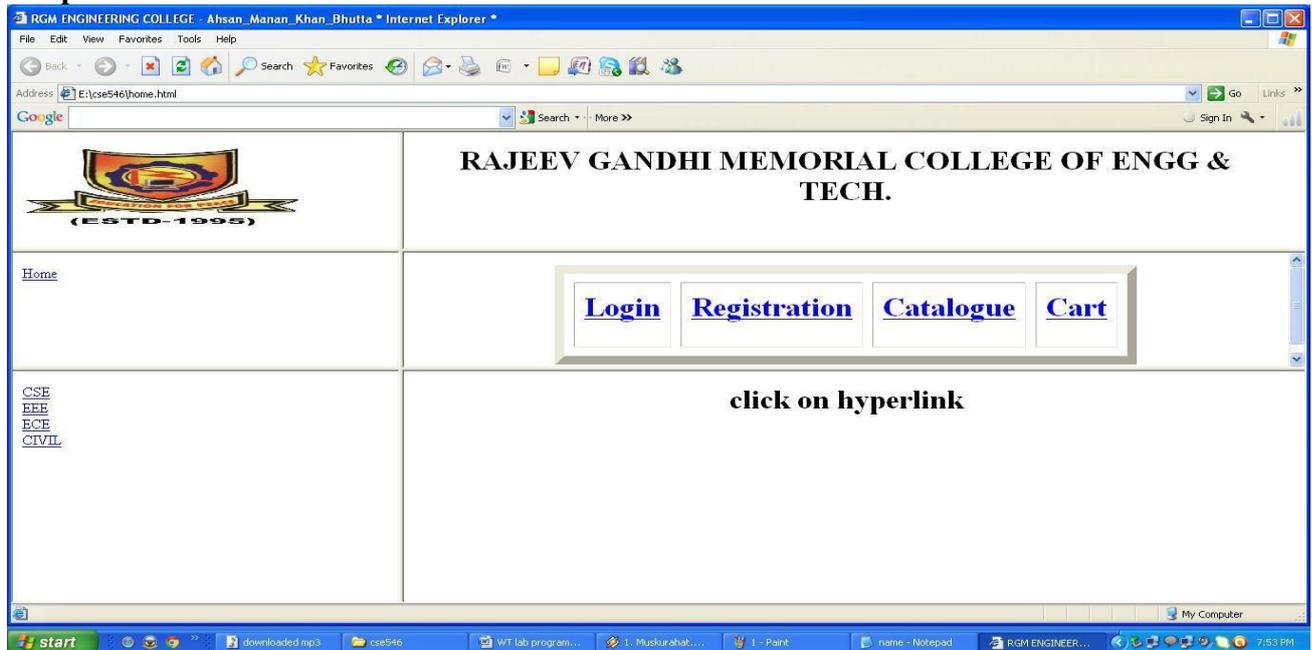
Example:

```

<html>
  <head>
    <title>RGM ENGINEERING COLLEGE</title>
  </head>
  <frameset cols="30%,70%">
    <frameset rows="25%,25%,50%">
      <frame src="e:\cse546\logo.html">
      <frame src="e:\cse546\home1.html">
      <frame src="e:\cse546\courses.html">
    </frameset>
    <frameset rows="25%,25%,50%">
      <frame src="e:\cse546\name.html">
      <frame src="e:\cse546\table.html">
      <frame src="e:\cse546\default.html" name="display">
    </frameset>
  </frameset>

```

</html>

Output:

HTML Forms: HTML Forms are required when you want to collect some data from the user. For example during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the user and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application. There are various form elements available like **text fields, text area fields, drop-down menus, radio buttons, checkboxes**, etc.

The HTML **<form>** tag is used to create an HTML form and it has following syntax:

Syntax: **<form action="Script URL" method="GET|POST">**
 form elements like input, textarea etc.
 </form>

Form Attributes:

Attribute	Description
action	This attribute specifies the name and location of URL what will be used to process the data.
method	Data can be sent in one of two ways that is either GET or POST methods. GET: It is used to get or retrieve information from a server. POST: It is used to send information to a server
Input	This attribute sets an area in a form for user input. Syntax: <INPUT>

Type	This attribute specifies what type of input control such as text, password, radio, etc. Syntax: <input type="any control">
Name	This attribute specifies the name of the form and also specifies the control type if it is used with the controls for processing the results. Syntax: <input type="any control" name="give own name"> Syntax: <form name="form1"> Id: This is used to give the unique identity to a form Syntax: <form id="f1">
Value	This attribute provides default values. Syntax: <input type="any control" name="give your own name" value="any value/name for control"> Example: <input type="radio" name="breakfast" value="bread">bread
Size	This attribute specifies the size of the text fields. Syntax: <input type="any control type" name="give your own name" size="n"> Example: <input type="Text" name="username" size="20">username
Maxlength	This attribute specifies the maximum input size for text fields. Syntax: <input type="any control type" name="give your own name" size="n" maxlength="n"> Example: <input type="Text" name="username" size="25" maxlength="50">username
Selected	This attribute specifies the default selection when form is initially displayed or reloaded. Syntax: <select name="user defined name"> <option selected>select default value <option>value1<option>value n </select> Example: <select name="month"> <option selected>select month <option>jan <option>feb <option>mar </select>
target	Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.

HTML Form Controls: There are different types of form controls that you can use to collect data

using HTML form:

Text field: This control is used to create text field where user can enter text inside it. It is created using HTML <input> tag.

Example: Here is a basic example of a single-line text input used to take first name and last name:

```
<html>
  <head>
    <title>Text Input Control</title>
  </head>
  <body>
    <form >
      First name: <input type="text" name="first_name" /><br>
      Last name: <input type="text" name="last_name" />
    </form>
  </body>
</html>
```

Attributes

Following is the list of attributes for <input> tag for creating text field.

Attribute	Description
type	Indicates the type of input control and for text input control it will be set to text.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	This can be used to provide an initial value inside the control.
size	Allows to specify the width of the text-input control in terms of characters.
maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

Password field controls: This control creates a password textfield which shows asterisk (*) on the textfield.. They are also created using HTML <input> tag but type attribute is set to password.

Example: Here is a basic example of a single-line password input used to take user password:

```
<html>
  <head>
    <title>Password Input Control</title>
  </head>
  <body>
    <form >
      User ID : <input type="text" name="user_id" /><br>
      Password: <input type="password" name="password" />
    </form>
  </body>
</html>
```

Attributes

Following is the list of attributes for <input> tag for creating password field.

Attribute	Description
type	Indicates the type of input control and for password input control it will be set to password.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	This can be used to provide an initial value inside the control.
size	Allows to specify the width of the text-input control in terms of characters.
maxlength	Allows to specify the maximum number of characters a user can enter into the text box.

Text area: This control creates a text area where multiple lines of data can be entered. This controls is created using <textarea> tag.

Example: Here is a basic example of a multi-line text input used to take item description:

```
<html>
    <head>
        <title>Multiple-Line Input Control</title>
    </head>
    <body>
        <form>
            Description :<textarea rows="5"cols="50" name="description">
                Enter description here...</textarea>
        </form>
    </body>
</html>
```

Attributes

Following is the list of attributes for <textarea> tag.

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
rows	Indicates the number of rows of text area box.
cols	Indicates the number of columns of text area box

Checkbox Control: Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to **checkbox**.

Example

Here is an example HTML code for a form with two checkboxes:

```
<html>
    <head>
        <title>Checkbox Control</title>
```

```

        </head>
        <body>
            <form>
                <input type="checkbox" name="maths" value="on"> Maths
                <input type="checkbox" name="physics" value="on"> Physics
            </form>
        </body>
    </html>

```

Attributes

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to checkbox.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the checkbox is selected.
checked	Set to checked if you want to select it by default.

Radio Button Control: Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML `<input>` tag but type attribute is set to radio.

Example

Here is example HTML code for a form with two radio buttons:

```

<html>
    <head>
        <title>Radio Box Control</title>
    </head>
    <body>
        <form>
            <input type="radio" name="subject" value="maths"> Maths
            <input type="radio" name="subject" value="physics"> Physics
        </form>
    </body>
</html>

```

Attributes

Attribute	Description
type	Indicates the type of input control and for checkbox input control it will be set to radio.
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
value	The value that will be used if the radio box is selected.
checked	Set to checked if you want to select it by default.

Select ion lists: This control creates a drop-down lists, where the user can select his choice from list.

Syntax:

```
<select name="user defined name">
    <option selected>select default value
    <option>value1<option>value n
</select>
```

Example:

```
<select name="month">
    <option selected>select month
    <option>jan
    <option>feb
    <option>mar
</select>
```

Example

Here is example HTML code for a form with one drop down box

```
<html>
    <head>
        <title>Select Box Control</title>
    </head>
    <body>
        <form>
            <select name="dropdown">
                <option value="Maths" selected>Maths</option>
                <option value="Physics">Physics</option>
            </select>
        </form>
    </body>
</html>
```

Attributes

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
size	This can be used to present a scrolling list box.
multiple	If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag:

Attribute	Description
value	The value that will be used if an option in the select box box is selected.
selected	Specifies that this option should be the initially selected value when the page loads.
label	An alternative way of labeling options

File Upload Box: This control is used to upload the file. This is also created using the <input> element but type attribute is set to file.

Example: Here is example HTML code for a form with one file upload box:

```
<html>
    <head>
        <title>File Upload Box</title>
    </head>
    <body>
        <form>
            <input type="file" name="fileupload" accept="image/*" />
        </form>
    </body>
</html>
```

Attributes

Following is the list of important attributes of file upload box:

Attribute	Description
name	Used to give a name to the control which is sent to the server to be recognized and get the value.
accept	Specifies the types of files that the server accepts.

Button Controls: This control is used to create a button on the form. You can create a button using <input> tag by setting its type attribute to button. The type attribute can take the following values:

Type	Description
submit	This creates a submit button on the form. when activated, a submit button submits the form. Form may contain more than one submit button <input type="submit">
reset	This creates a submit button on the form. when activated, it resets form controls to their initial values. <input type="RESET">
button	This creates a button that is used to trigger a client-side script when the user clicks that button.
image	This creates a clickable button but we can use an image as background of the button.

Example: Here is example HTML code for a form with three types of buttons:

```
<html>
    <head>
        <title>File Upload Box</title>
    </head>
    <body>
        <form>
            <input type="submit" name="submit" value="Submit" />
            <input type="reset" name="reset" value="Reset" />
            <input type="button" name="ok" value="OK" />
        </form>
    </body>
</html>
```

```

        <input type="image" name="imagebutton" src="logo.png" />
    </form>
</body>
</html>

```

Hidden Form Controls: It stores the hidden data that is data is not visible to users unless they view the source code.

Example:Here is example HTML code to show the usage of hidden control:

```

<html>
  <head>
    <title>File Upload Box</title>
  </head>
  <body>
    <form>
      <p>This is page 10</p>
      <input type="hidden" name="pagename" value="10" />
      <input type="submit" name="submit" value="Submit" />
      <input type="reset" name="reset" value="Reset" />
    </form>
  </body>
</html>

```

Label: This control creates label on the form which can be used to describe the other controls.Each label element is associated with exactly one form control.

```
<input type="LABEL">
```

Example:

```

<html>
  <head>
    <title>Registration</title>
  </head>
  <body bgcolor=lightblue>
    <h1 align=center><u>Registration Form</u></h1> <br><br><br>
    <div>
      <strong>
        First Name <input type="text" value=" " name="txt1"><br><br>
        Last Name <input type="text" value=" " name="txt2"><br><br>
        UserName <input type="text" value="" name="txt3"><br><br>
        Password <input type="password" value="" name="pwd1"><br>
        Confirm Password <input type="password" value="" name="pwd2"><br>
        Address <textarea rows=3 cols=60></textarea><br><br>
        Date of Birth
        dd<select name="sel1">
          <option>--</option>
          <option>01</option>
          <option>02</option>
          <option>03</option>
          <option>04</option>
          <option>05</option>
          <option>27</option>
          <option>28</option>
          <option>29</option>
          <option>30</option>
        </select>
      </strong>
    </div>
  </body>
</html>

```

```
        <option>31</option>
</select>
mm<select name="sel2">
    <option>--</option>
    <option>01</option>
    <option>02</option>
    <option>03</option>
    <option>04</option>
    <option>05</option>
    <option>06</option>
    <option>07</option>
    <option>08</option>
    <option>09</option>
    <option>10</option>
    <option>11</option>
    <option>12</option>
</select>
yyyy<select name="sel3">
    <option>----</option>
    <option>1987</option>
    <option>1988</option>
    <option>1989</option>
    <option>1990</option>
    <option>1991</option>
    <option>1992</option>
    <option>1993</option>
    <option>1994</option>
    <option>1995</option>
    <option>1996</option>
    <option>1997</option>
    <option>1998</option>
    <option>1999</option>
    <option>2000</option>
    <option>2001</option>
    <option>2002</option>
    <option>2003</option>
    <option>2004</option>
    <option>2005</option>
    <option>2006</option>
    <option>2007</option>
    <option>2008</option>
    <option>2009</option>
    <option>2010</option>
    <option>2011</option>
    <option>2012</option>
    <option>2013</option>
    <option>2014</option>
    <option>2015</option>
    <option>2016</option>
```

```

        <option>2017</option>
    </select><br><br>
    Sex
    <input name="rb1" type="radio" value="radiobutton">Male
    <input name="rb1" type="radio" value="radiobutton">Female
    <br><br>
    Marital Status
    <input name="rb2" type="radio" value="radiobutton">Single
    <input name="rb2" type="radio" value="radiobutton">Married
    <br><br>
    Mobile Number <input type="text" name="txt4"><br><br>
    Branch
    <input name="rb3" type="radio" value="radiobutton">CSE
    <input name="rb3" type="radio" value="radiobutton">IT
    <input name="rb3" type="radio" value="radiobutton">ECE
    <input name="rb3" type="radio" value="radiobutton">EEE
    <input name="rb3" type="radio" value="radiobutton">MECH
    <br><br>
    Languages Known
    <input name="cb1" type="checkbox" value="checkbox">English
    <input name="cb1" type="checkbox" value="checkbox">Telugu
    <input name="cb1" type="checkbox" value="checkbox">Hindi
    <input name="cb1" type="checkbox" value="checkbox">Kannada
    <input name="cb1" type="checkbox" value="checkbox">Tamil
    <br><br>
    <center>
        <input type="submit" value="SUBMIT" name="btn1">
        <input type="reset" value="CANCEL" name="btn1">
    </center>
</strong>
</body>
</html>

```

Output:

The screenshot shows a web browser window titled "Registration - Microsoft Internet Explorer". The address bar shows "E:\html\registration.html". The main content area displays a "Registration Form" with the following fields and controls:

- First Name:
- Last Name:
- UserName:
- Password:
- Confirm Password:
- Address:
- Date of Birth: dd: , mm: , yyyy:
- Sex: Male Female
- Marital Status: Single Married
- Mobile Number:
- Branch: CSE IT ECE EEE MECH
- Languages Known: English Telugu Hindi Kannada Tamil

At the bottom of the form, there are two buttons: "SUBMIT" and "CANCEL".

Cascading Style Sheet: Cascading Style Sheets (CSS) describe how documents are presented on screens, in print. To determine style and layout of the webpage we can use Cascading Style Sheets. Cascading Style Sheets (CSS) provide easy and effective alternatives to specify various attributes for the HTML tags. Using CSS, you can specify a number of style properties for a given HTML element. Each property has a name and a value, separated by a colon (:). Each property declaration is separated by a semi-colon (;).

Syntax: `selector{ property:value;property1:value1}`

External Style Sheet: when the style is applied to many pages we use external style sheet. With an external style sheet, you can change the look of an entire web site by changing one file. Each page must link to the style sheet using the <link> tag. The <link> tag goes inside the head section.

Example: `<head>`
 `<link rel="stylesheet" type="text/css" href="mystyle.css"/>`
 `</head>`

An external style sheet can be written in any text editor. The file should not contain any html tags. The style sheet should be saved with a .css extension.

Example:

extern.css:

```
body {background-color: #d0e4fe;}
h1 { color: orange; text-align: center; }
p { font-family: "Times New Roman"; font-size: 20px; }
```

extern.html:

```
<html>
  <head>
    <title>External Style Sheets</title>
    <link rel="stylesheet" type="text/css" href="extern.css">
  </head>
  <body>
    <h1>External Style Sheets</h1><br>
    <p>This is a paragraph
  </body>
</html>
```

Output:



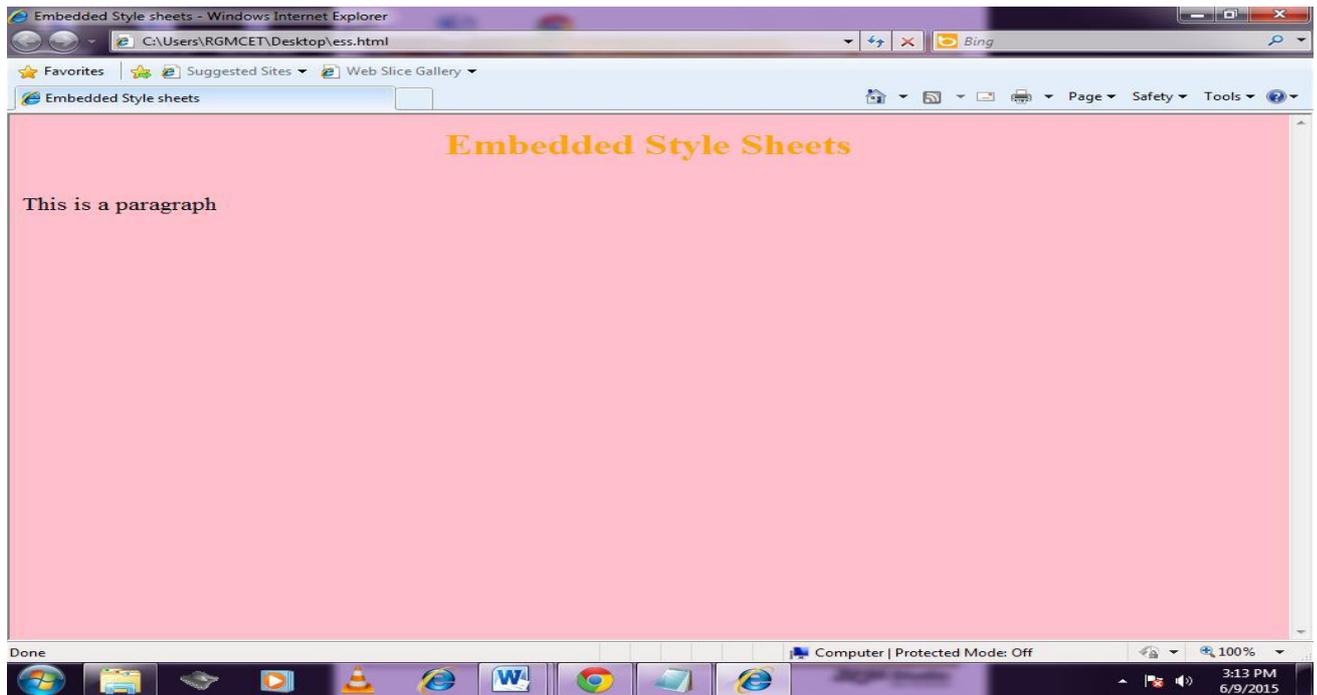
Internal(Embedded) Style Sheet: when the Style is applied to a single document(page) then we can use internal style sheet. We define internal style sheet in the head section of an html page,by using the <style> tag.

Example:

```
<html>
  <head>
    <title>Embedded Style sheets</title>
    <style type="text/css">
      body {background-color: pink;}
      h1 { color: orange; text-align: center; }
      p { font-family: "Times New Roman"; font-size: 20px; }
    </style>
  </head>
  <body>
    <h1>Embedded Style Sheets</h1><br>
    <p>This is a paragraph
  </body>
```

</html>

Output:

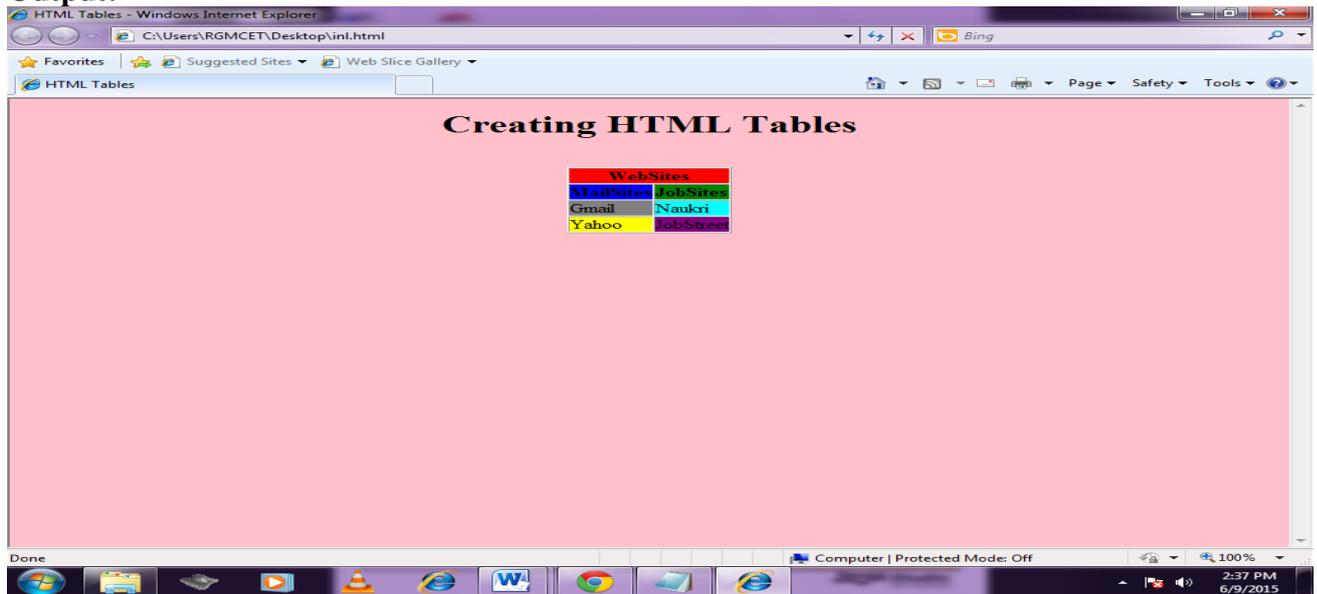


Inline Style Sheet: when we want to apply styles to only single HTML element we use inline style sheet attribute in the relevant tag. The style attribute can contain any CSS property.

Example:

```
<html>
  <head>
    <title>HTML Tables</title>
  </head>
  <body bgcolor="pink">
    <center>
      <h1>Creating HTML Tables</h1><br>
      <table border="2" cellpadding="4" cellspacing="4">
        <tr>
          <th colspan="2" style="background-color:red"> WebSites</th>
        </tr>
        <tr>
          <th style="background-color:blue">MailSites</th>
          <th style="background-color:green">JobSites</th>
        </tr>
        <tr>
          <td style="background-color:grey">Gmail</td>
          <td style="background-color:aqua">Naukri</td>
        </tr>
        <tr>
          <td style="background-color:yellow">Yahoo</td>
          <td style="background-color:purple">JobStreet</td>
        </tr>
      </table>
    </center>
```

```
</body>
</html>
```

Output:

Scripting Languages: It is a programming language that allows us to write programs in form of script, these scripts are interpreted not compiled and executed line by line.

Or

It is a programming language **designed for integrating and communicating with other programming languages**. Some of the most widely used scripting languages are **JavaScript**, VBScript, PHP, Perl, Python, Ruby, ASP and Tcl.

Types: scripting languages can be classified into two types.

- Client-side scripting
- Server-side scripting

Client-side scripting (front-end):

- **Client-side scripting** generally refers to the **programs on the web that are executed client-side, by the user's web browser**, instead of **server-side**.
- The best example of client-side script is **JavaScript**.
- Client-side script **executes in the browser after the page is loaded**.
- Client-side script programs can be embedded into HTML files or also can be kept as separate files.
- If the Client-side scripts are embedded with an HTML(or) XHTML document then those are called **Embedded script**.
- If the Client-side script kept as a separate file which is referenced by the documents then

those are called **External script**.

- Client-side scripts **contain the instruction for the browser to be executed in response to certain user's action.**
- Client-side scripts **have greater access to the information and functions available** on the browser.
- Due to security instructions **client-side scripts may not be allowed to access the user's computer beyond the web browser application.**

JavaScript introduction: html web pages are **static in nature where they do not react to any events and do not produce different outputs when different user browses them or same user browses them. This means that there is no programming involved in it. Therefore to add interactivity to html pages we use JavaScript, it is a quick programming language which can be used on the client side scripting.java script is mainly used for performing a number of tasks such as validating input, doing local calculations, window pop-ups.**

Java script is originated from a language called **LiveScript** and was **developed by sunmicrosystems and netscape navigator.** java script is **used to provide client-side in browser application** and it is **not complicated like java.**java script official name is **ECMA script. ECMA script is developed and maintained by the ECMA organization.ECMA-262 is the official java script standard.**

The JavaScript was invented by **Brendan Eich at Netscape** and has appeared in all Netscape and Microsoft browsers since 1996.

Javascript:

- JavaScript is an interpreted programming language from Netscape.
- JavaScript is a scripting language designed for adding interactivity to Web pages and creating Web applications.
- JavaScript is widely supported for following browsers:
 - Netscape Navigator (beginning with version 2.0)
 - Microsoft Internet Explorer (beginning with version 3.0)
 - Firefox
 - Safari
 - Opera
 - Google Chrome

- Java script is object based language.
- Java script is loosely typed language.

Key points in java script:

- Each line of the code is terminated by a semicolon.
- Block of code must be surrounded by a pair of curly brackets. A block of code is a set of instructions that are to be executed together as a unit, This is because they are to be optional and dependent upon a Boolean condition or because they are to be executed repeatedly.
- Functions have parameters which are passed inside the parenthesis.
- Variables are declared using keyword var.
- Script requires neither a main function nor an exit condition.
- Execution of script starts with the first line of code and runs until there no more code.
- Java script programs are stored with .js extension.
- Java script is case sensitive
- Java script is directly embedded into html pages.
- Java script is interpreted language(means script executes without compilation)

Editing java script:

- For writing java script we need only notepad or any text editor.
- Java script will run in any web server anywhere! For this we need browser.

Basic structure of javascript:

```
<html>
  <head>
    <script type="text/javascript">
      // write your java script code here
    </script>
  </head>
  <body>
    <script type="text/javascript">
      // write your java script code here
    </script>
  </body>
</html>
```

We can place unlimited number scripts in your document, so you can have scripts in both the body and the head section.

Writing the script in body section executes automatically on its own.**Example:**

```
<html>
  <head>
    <script type="text/javascript">
      // write your java script code here
```

```

        document.writeln("message from head section");
    </script>
</head>
<body>
    <script type="text/javascript">
        // write your java script code here
        document.writeln("message from body section");
    </script>
</body>
</html>

```

Writing JavaScript in the head section **executes only when explicitly called**. The explicit calling a script is done by the event **OnLoad**.

Example:

```

<html>
    <head>
        <script type="text/javascript">
            function message()
            {
                document.writeln("message called from body section");
            }
        </script>
    </head>
    <body onload="message()">
    </body>
</html>

```

Using an external java script: If you want run the same java script on several pages, without having to write the same script on every page, you can write a java script in an external file. save the java script file with **.js** extension.

Syntax:

```

<html>
    <head>
        <script type="text/javascript" src="externalfilename.js">
        </script>
    </head>
    <body >
    </body>
</html>

```

Example:

```

<html>
    <head>
        <script type="text/javascript"src="extjs1.js">
        </script>
    </head>
    <body onload="message()">
    </body>
</html>

```

exjs1.js (external JavaScript file):

```
function message()  
{  
    document.writeln("message called from body section");  
}
```

Displaying information: To display the output information on a web page we use the following method.

```
document.writeln ("welcome");
```

Variables: variables in JavaScript are used to store the information. The variables in JavaScript are declared by using keyword var.

Declaration:

```
Syntax:    var money;  
           var name;
```

You can also declare multiple variables with the same var keyword as follows:

```
Syntax:    var money, name;
```

Initialization: Storing a value in a variable is called variable initialization. You can do variable initialization at the time of variable creation or at a later point in time when you need that variable.

```
Syntax:    var name = "Ali";  
           var money;  
           money = 2000.50;
```

Note:

- Use the var keyword only for declaration or initialization, once for the life of any variable name in a document. You should not re-declare same variable twice.
- JavaScript is untyped language. This means that a JavaScript variable can hold a value of any data type. Unlike many other languages, you don't have to tell JavaScript during variable declaration what type of value the variable will hold.
- The value type of a variable can change during the execution of a program and JavaScript takes care of it automatically.

Local variables: if you declare a variable within a function, the variable can only be accessed within that function. When you exit the function, the variable is destroyed. These variables are called local variables.

Global Variables: If you declare a variable outside a function, all the functions on your page can access it. The lifetime of these variables starts when they are declared, and ends when the page is closed. These variables are called global variables.

Example:

```

<script type="text/javascript">
    var myVar = "global"; // Declare a global variable
    function checkscope()
    {
        var myVar = "local"; // Declare a local variable
        document.write(myVar);
    }
</script>

```

Output: Local

JavaScript Variable Names: While naming your variables in JavaScript, keep the following rules in mind.

- You should not use any of the JavaScript reserved keywords as a variable name. For example, break or boolean variable names are not valid.
- JavaScript variable names should not start with a numeral (0-9). They must begin with a letter or an underscore character. For example, 123test is an invalid variable name but _123test is a valid one.
- JavaScript variable names are case-sensitive. For example, Name and name are two different variables.

Alert(): This command will make a popup box apper. This can be useful for warning users about the things or giving them instructions

```
alert("welcome");
```

prompts(); This command is useful to get some information from the user.

```
var firstname=prompt("please enter your name:","firstname");
```

The text between the first set of quotes is what is written on the prompt box.The text between the second set of quotes is what is the default text for the input section of the box.

Example:

```

<html>
    <head>
        <title>prompt ,alert,variable</title>
    </head>
    <body>
        <script type="text/javascript">
            var firstname;
            firstname=prompt("enter first name","firstname");
            document.writeln("your firstname is"+firstname);
            alert("entered the firstname");
        </script>
    </body>
</html>

```

Opening a new window in JavaScript: To open a new window we use **window.open** command

Syntax: `window.open("link.html","mywindow");`

Confirm box: A confirm box is used if you want the **user to verify or accept something**. When a confirm box pops up, the user will have to click either ok or cancel to proceed.

Syntax: `confirm("some text");`

Example:

```
<html>
  <head>
    <title>new window and confirm box</title>
  </head>
  <body>
    <script type="text/javascript">
      window.open("js4.html","mywindow");
      confirm("you have opened a new window");
    </script>

  </body>
</html>
```

Comments in JavaScript: JavaScript supports both C-style and C++-style comments. Thus:

- Any text between a // and the end of a line is treated as a comment and is ignored by JavaScript.
- Any text between the characters /* and */ is treated as a comment. This may span multiple lines.
- JavaScript also recognizes the HTML comment opening sequence <!--. JavaScript treats this as a single-line comment, just as it does the // comment.
- The HTML comment closing sequence --> is not recognized by JavaScript so it should be written as //-->.

Example:

```
<script language="javascript" type="text/javascript">
<!--
  // This is a comment. It is similar to comments in C++
  /*
   * This is a multiline comment in JavaScript
   * It is very similar to comments in C Programming
   */

  //-->
</script>
```

OPERATORS: Operators are used to perform operations on operands.

- Arithmetic Operators
- Comparison Operators
- Logical (or Relational) Operators
- Assignment Operators
- Conditional (or ternary) Operators

Arithmetic Operators: Arithmetic operators are used to perform arithmetic operations between the variables and/or values. For example take $y=5$, the below table explains the arithmetic operators.

Operator	Description	Example	result
----------	-------------	---------	--------

+	Addition	$X=y+2$	$X=7$
-	Subtraction	$X=y-2$	$X=3$
*	Multiplication	$X=y*2$	$X=10$
/	Division	$X=y/2$	$X=2.5$
%	Modulus	$X=y\%2$	$X=1$
++	Increment	$X=++y$	$X=6$
--	Decrement	$X=--y$	$X=4$

Example: The following code shows how to use arithmetic operators in JavaScript.

```
<html>
  <body>
    <script type="text/javascript">
      var a = 33;
      var b = 10;
      var c = "Test";
      var linebreak = "<br />";
      document.write("a + b = ");
      result = a + b;
      document.write(result);
      document.write(linebreak);
      document.write("a - b = ");
      result = a - b;
      document.write(result);
      document.write(linebreak);
      document.write("a / b = ");
      result = a / b;
      document.write(result);
      document.write(linebreak);
      document.write("a % b = ");
      result = a % b;
      document.write(result);
      document.write(linebreak);
      document.write("a + b + c = ");
      result = a + b + c;
      document.write(result);
      document.write(linebreak);
      a = a++;
      document.write("a++ = ");
      result = a++;
      document.write(result);
      document.write(linebreak);
      b = b--;
      document.write("b-- = ");
      result = b--;
      document.write(result);
      document.write(linebreak);
    </script>
  </body>
</html>
```

Output: a + b = 43

```

a - b = 23
a / b = 3.3
a % b = 3
a + b + c = 43
a++ = 33
b-- = 10

```

Comparison Operators: comparison operators are used in logical statements to determine equality or difference between the variables or values. For example take $x=5$, the below table explains the comparison operators.

Operator	Description	Example
==	Is equal to	$X==8$ is false
===	Is exactly equal to	$X===5$ is true $X===\text{"5"}$ is false
!=	Is not equal	$X!=8$ is true
>	Is greater than	$x>8$ is false
<	Is less than	$X<8$ is true
>=	Is greater than or equal to	$x>=8$ is false
<=	Is less than or equal to	$X<=8$ is true

Example: The following code shows how to use comparison operators in JavaScript.

```

<html>
  <body>
    <script type="text/javascript">
      var a = 10;
      var b = 20;
      var linebreak = "<br />";
      document.write("(a == b) => ");
      result = (a == b);
      document.write(result);
      document.write(linebreak);
      document.write("(a < b) => ");
      result = (a < b);
      document.write(result);
      document.write(linebreak);
      document.write("(a > b) => ");
      result = (a > b);
      document.write(result);
      document.write(linebreak);
      document.write("(a != b) => ");
      result = (a != b);
      document.write(result);
      document.write(linebreak);
      document.write("(a >= b) => ");
      result = (a >= b);
      document.write(result);
      document.write(linebreak);
      document.write("(a <= b) => ");

```

```

        result = (a <= b);
        document.write(result);
        document.write(linebreak);
    </script>
</body>
</html>

```

Output: (a == b) => false
(a < b) => true
(a > b) => false
(a != b) => true
(a >= b) => false
(a <= b) => true

Logical Operators: Logical operators are used in logic between variables or values. For example take x=6 and y==3, the below table explains the logical operators.

Operator	Description	Example
&&	and	(x<10&&y>1) is true
	or	(x==5 y==5) is false
!	not	!(x==y) is true

Example: Try the following code to learn how to implement Logical Operators in JavaScript.

```

<html>
  <body>
    <script type="text/javascript">
      var a = true;
      var b = false;
      var linebreak = "<br />";
      document.write("(a && b) => ");
      result = (a && b);
      document.write(result);
      document.write(linebreak);
      document.write("(a || b) => ");
      result = (a || b);
      document.write(result);
      document.write(linebreak);
      document.write("!(a && b) => ");
      result = !(a && b);
      document.write(result);
      document.write(linebreak);
    </script>
  </body>
</html>

```

Output: (a && b) => false
(a || b) => true
!(a && b) => true

Assignment Operators: assignment operators are used to assign values to the variables. For example take x=10, the below table explains the assignment operators

Operator	Example	result
=	X=y	X=5

+=	X+=y	X=15
-=	x-=y	X=5
=	X=y	X=50
/=	x/=y	X=2
%=	X%=y	X=0

Example: Try the following code to implement assignment operator in JavaScript.

```

<html>
  <body>
    <script type="text/javascript">
      var a = 33;
      var b = 10;
      var linebreak = "<br />";
      document.write("Value of a => (a = b) => ");
      result = (a = b);
      document.write(result);
      document.write(linebreak);
      document.write("Value of a => (a += b) => ");
      result = (a += b);
      document.write(result);
      document.write(linebreak);
      document.write("Value of a => (a -= b) => ");
      result = (a -= b);
      document.write(result);
      document.write(linebreak);
      document.write("Value of a => (a *= b) => ");
      result = (a *= b);
      document.write(result);
      document.write(linebreak);
      document.write("Value of a => (a /= b) => ");
      result = (a /= b);
      document.write(result);
      document.write(linebreak);
      document.write("Value of a => (a %= b) => ");
      result = (a %= b);
      document.write(result);
      document.write(linebreak);
    </script>
  </body>
</html>

```

Output:

```

Value of a => (a = b) => 10
Value of a => (a += b) => 20
Value of a => (a -= b) => 10
Value of a => (a *= b) => 100
Value of a => (a /= b) => 10
Value of a => (a %= b) => 0

```

Conditional Operator (? :): The conditional operator first evaluates an expression for a true or false value and then executes one of the two given statements depending upon the result of the evaluation.

Syntax: **variablename= (Condition)?value1:value2**

Example:

```

<html>
  <body>
    <script type="text/javascript">
      var a = 10;
      var b = 20;
      var linebreak = "<br />";
      document.write ("((a > b) ? 100 : 200) => ");
      result = (a > b) ? 100 : 200;
      document.write(result);
      document.write(linebreak);
      document.write ("((a < b) ? 100 : 200) => ");
      result = (a < b) ? 100 : 200;
      document.write(result);
      document.write(linebreak);
    </script>
  </body>
</html>

```

Output: ((a > b) ? 100 : 200) => 200
 ((a < b) ? 100 : 200) => 100

Data types: JavaScript allows you to work with four primitive data types:

- **NUMERIC:** Basic numbers, they can be integer, floating point.
Example: 123, 120.50 etc.
- **STRING:** Strings of text.
Example: var variablename= "This text string" etc.
- **Boolean:** Boolean variables hold the values true and false.
- **NULL:** A null value means one that has not yet been decided; it does not mean null or zero and should not be used in that way ever.

Conditional statements: conditional statements are used to perform different actions based on different conditions.

If statement: It is used to execute some code only if a specified condition is true.

Syntax: **if(condition)**
 {
 Code to be executed if condition is true
 }

Example:

```

<html>
  <body>
    <script type="text/javascript">
      var age = 20;
      if( age > 18 )
      {
        document.write("<b>Qualifies for driving</b>");
      }
    </script>
  </body>
</html>

```

```

        </script>
    </body>
</html>

```

IF-ELSE statement: It is used to execute some code only if a condition is true and another code if the condition is false.

Syntax:

```

if(condition)
{
    Code to be executed if condition is true
}
else
{
    Code to be executed if condition is false
}

```

Example:

```

<html>
<body>
    <script type="text/javascript">
        var age = 15;
        if( age > 18 ){
            document.write("<b>Qualifies for driving</b>");
        }
        else
        {
            document.write("<b>Does not qualify for driving</b>");
        }
    </script>
</body>
</html>

```

if...else, if...else Statement: It is used to execute one of several blocks of code.

Syntax

```

if(condition1)
{
    Code to be executed if condition1 is true
}
else if(condition2)
{
    Code to be executed if condition2 is true
}
else
{
    Code to be executed if condition1 and condition2 are false
}

```

Example:

```

<html>
<body>
    <script type="text/javascript">
        var book = "maths";
        if( book == "history" )

```

```

        {
            document.write("<b>History Book</b>");
        }
        else if( book == "maths" )
        {
            document.write("<b>Maths Book</b>");
        }
        else if( book == "economics" )
        {
            document.write("<b>Economics Book</b>");
        }
        else
        {
            document.write("<b>Unknown Book</b>");
        }
    }
</script>
</body>
</html>

```

SWITCH statement: It is used to select one of many blocks of code to be executed. The objective of a switch statement is to give an expression to evaluate and several different statements to execute based on the value of the expression. The interpreter checks each case against the value of the expression until a match is found. If nothing matches, a default condition will be used.

Syntax:

```

switch (expression)
{
    case condition 1: statement(s)
                    break;
    case condition 2: statement(s)
                    break;
    ...
    case condition n: statement(s)
                    break;
    default: statement(s)
}

```

The break statements indicate the end of a particular case. If they were omitted, the interpreter would continue executing each statement in each of the following cases.

Example

```

<html>
<body>
<script type="text/javascript">
    var grade='A';
    document.write("Entering switch block<br />");
    switch (grade)
    {
        case 'A': document.write("Good job<br />");
                break;
        case 'B': document.write("Pretty good<br />");
    }
}

```

```

        break;
    case 'C': document.write("Passed<br />");
        break;
    case 'D': document.write("Not so good<br />");
        break;
    case 'F': document.write("Failed<br />");
        break;
    default: document.write("Unknown grade<br />")
    }
    document.write("Exiting switch block");
</script>
</body>
</html>

```

Output: Entering switch block
Good job

Example:

```

<html>
<body>
    <script type="text/javascript">
        var grade='A';
        document.write("Entering switch block<br />");
        switch (grade)
        {
            case 'A': document.write("Good job<br />");
            case 'B': document.write("Pretty good<br />");
            case 'C': document.write("Passed<br />");
            case 'D': document.write("Not so good<br />");
            case 'F': document.write("Failed<br />");
            default: document.write("Unknown grade<br />")
        }
        document.write("Exiting switch block");
    </script>
    <p>Set the variable to different value and then try...</p>
</body>
</html>

```

Output: Entering switch block
Good job
Pretty good
Passed
Not so good
Failed
Unknown grade

CONDITIONAL LOOPS:

WHILE LOOP: The purpose of a while loop is to execute a statement or code block repeatedly as long as an expression is true. Once the expression becomes false, the loop terminates. The while loop first check for the condition being tested, and if it is satisfied, only then executes the code. It is entry control loop.

Syntax

```

while (expression)
{
    code to be executed if expression is true
}

```

Example

```

<html>
  <body>
    <script type="text/javascript">
      var count = 0;
      document.write("Starting Loop ");
      while (count < 10)
      {
          document.write("Current Count : " + count + "<br />");
          count++;
      }
      document.write("Loop stopped!");
    </script>
  </body>
</html>

```

Output: Starting Loop Current Count : 0
 Current Count : 1
 Current Count : 2
 Current Count : 3
 Current Count : 4
 Current Count : 5
 Current Count : 6
 Current Count : 7
 Current Count : 8
 Current Count : 9
 Loop stopped!

Do-WhileLoop: The do...while loop is similar to the while loop except that the condition check happens at the end of the loop. This means that the loop will always be executed at least once, even if the condition is false. It is an exit control loop.

Syntax

```

do
{
    code to be executed;
} while (expression);

```

Example:

```

<html>
  <body>
    <script type="text/javascript">
      var count = 0;
      document.write("Starting Loop" + "<br />");
      do
      {
          document.write("Current Count : " + count + "<br />");

```

```

        count++;
    }while (count < 5);
    document.write ("Loop stopped!");
</script>
</body>
</html>

```

Output: Starting Loop
 Current Count : 0
 Current Count : 1
 Current Count : 2
 Current Count : 3
 Current Count : 4
 Loop Stopped!

FOR LOOP: It includes the following three important parts:

- The loop initialization where we initialize our counter to a starting value. The initialization statement is executed before the loop begins.
- The test statement which will test if a given condition is true or not. If the condition is true, then the code given inside the loop will be executed, otherwise the control will come out of the loop.
- The iteration statement where you can increase or decrease your counter.

Syntax

```

for (initialization; test condition; iteration statement)
{
    code to be executed if test condition is true
}

```

Example

```

<html>
<body>
    <script type="text/javascript">
        var count;
        document.write("Starting Loop" + "<br />");
        for(count = 0; count < 10; count++)
        {
            document.write("Current Count : " + count );
            document.write("<br />");
        }
        document.write("Loop stopped!");
    </script>
    <p>Set the variable to different value and then try...</p>
</body>
</html>

```

Output: Starting Loop
 Current Count : 0
 Current Count : 1
 Current Count : 2
 Current Count : 3
 Current Count : 4

```

Current Count : 5
Current Count : 6
Current Count : 7
Current Count : 8
Current Count : 9
Loop stopped!

```

LOOP CONTROL: JavaScript provides full control to handle loops and switch statements. There may be a situation when you need to come out of a loop without reaching at its bottom. There may also be a situation when you want to skip a part of your code block and start the next iteration of the loop. To handle all such situations, JavaScript provides break and continue statements. These statements are used to immediately come out of any loop or to start the next iteration of any loop respectively.

break Statement: It is used to exit a loop early, breaking out of the enclosing curly braces.

Example

```

<html>
  <body>
    <script type="text/javascript">
      var x = 1;
      document.write("Entering the loop<br /> ");
      while (x < 20)
      {
        if (x == 5)
        {
          break; // breaks out of loop completely
        }
        x = x + 1;
        document.write( x + "<br />");
      }
      document.write("Exiting the loop!<br /> ");
    </script>
  </body>
</html>

```

Output: **Entering the loop**
2
3
4
5
Exiting the loop!

Continue Statement: The continue statement tells the interpreter to immediately start the next iteration of the loop and skip the remaining code block. When a continue statement is encountered, the program flow moves to the loop check expression immediately and if the condition remains true, then it starts the next iteration, otherwise the control comes out of the loop.

Example

```

<html>

```

```

<body>
  <script type="text/javascript">
    var x = 1;
    document.write("Entering the loop<br /> ");
    while (x < 10)
    {
      x = x + 1;
      if (x == 5)
      {
        continue; // skip rest of the loop body
      }
      document.write( x + "<br />");
    }
    document.write("Exiting the loop!<br /> ");
  </script>
  <p>Set the variable to different value and then try...</p>
</body>
</html>

```

Output: **Entering the loop**
 2
 3
 4
 6
 7
 8
 9
 10
Exiting the loop!

FUNCTIONS: A function is a group of reusable code which can be called anywhere in your program. This eliminates the need of writing the same code again and again. Functions allow a programmer to divide a big program into a number of small and manageable functions.

Function Definition: Before we use a function, we need to define it. The most common way to define a function in JavaScript is by using the function keyword, followed by a unique function name, a list of parameters (that might be empty), and a statement block surrounded by curly braces.

Syntax

```

function functionname(parameter-list)
{
  statements
}

```

Example

```

function sayHello()
{
  alert("Hello there");
}

```

Calling a Function: To invoke a function somewhere later in the script, you would simply need to write the name of that function as shown in the following code.

```

<html>
  <head>
    <script type="text/javascript">
      function sayHello()
      {
        document.write ("Hello there!");
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="sayHello()" value="Say Hello">
    </form>
  </body>
</html>

```

Function Parameters : Till now, we have seen functions without parameters. But there is a facility to pass different parameters while calling a function. These passed parameters can be captured inside the function and any manipulation can be done over those parameters. A function can take multiple parameters separated by comma.

Example

```

<html>
  <head>
    <script type="text/javascript">
      function sayHello(name, age)
      {
        document.write (name + " is " + age + " years old.");
      }
    </script>
  </head>
  <body>
    <p>Click the following button to call the function</p>
    <form>
      <input type="button" onclick="sayHello('Zara', 7)" value="Say Hello">
    </form>
  </body>
</html>

```

return Statement: A JavaScript function can have an optional return statement. This is required if you want to return a value from a function. This statement should be the last statement in a function.

Example

```

<html>
  <head>
    <script type="text/javascript">
      function concatenate(first, last)
      {
        var full;

```

```

        full = first + last;
        return full;
    }
    function secondFunction()
    {
        var result;
        result = concatenate('Zara', 'Ali');
        document.write (result );
    }
</script>
</head>
<body>
    <p>Click the following button to call the function</p>
    <form>
    <input type="button" onclick="secondFunction()" value="Call Function">
    </form>
</body>
</html>

```

Arrays in JavaScript: Array is a group of similar data elements that all have same name and normally are of the same type. To refer a particular element in the array, we specify the name of the array and the position of particular element in the array. **Arrays in JavaScript are represented by the array object.**

BASIC OPERATIONS PERFORMED ON ARRAYS ARE

Creating array: Here arrays are constructed in three different ways.

1. Var days=["mon","tue","wed"];
2. Var days=new Array("mon","tue","wed");
3. Var days=new Array(4);

Note: unlike other programming language javascript arrays can hold mixed data types .

```
Var days=["mon","tue","wed"];
```

```
Var days=new Array("mon",24,57.5,"wed");
```

Accessing array: The elements in the array are accessed through their index. The same method is used to find elements and to change their values.

length: It is used to find the length of the array.

Example:

```

<html>
    <head>
        <title>array element</title>
    </head>
    <body bgcolor="red">
    </body>
    <script type="text/javascript">
        var a=new Array("mon",23.56,"tue","wed",5);
        var len=a.length;
        for(var i=0;i<len;i++)
        {
            document.write ("the array elements are"+a[i] );
        }
    </script>

```

```
</html>
```

Adding array: Adding an element uses the square brackets.

Syntax: `var arrayname[index]=value;`

Example:

```
<html>
  <head>
    <title>array element</title>
  </head>
  <body bgcolor="red">
  </body>
  <script type="text/javascript">
    var a=new Array("mon",23.56,"tue","wed",5);
    var len=a.length;
    for(var i=0;i<len;i++)
    {
      document.write ("the array elements are"+a[i] );
    }
    a[10]="Thursday";
    document.write ("the added element is"+a[10] );
  </script>
</html>
```

Searching array: To search an element, simply read each element in turn and compare it with the value you are looking for.

Example:

```
<html>
  <head>
    <title>array element</title>
  </head>
  <body bgcolor="red">
  </body>
  <script type="text/javascript">
    var a=new Array("mon",23.56,"tue","wed",5);
    var len=a.length;
    for(var i=0;i<len;i++)
    {
      if(a[i]=="wed")
      {
        document.write ("the searched element is"+a[i] );
        break;
      }
    }
  </script>
</html>
```

Removing array: To remove an element from an array uses the following procedure

1. Read element in array
2. If the element is not the one you want to delete, copy it into a temporary array.
3. If you want to delete the element then do nothing
4. Just increment the loop counter.
5. Repeat the process.

Example:

```
<html>
  <head>
```

```

        <title>array element</title>
    </head>
    <body bgcolor="red">
    </body>
    <script type="text/javascript">
        var a=new Array("mon",23.56,"tue","wed",5);
        var len=a.length;
        for(var i=0;i<len;i++)
        {
            if(a[i]=="wed")
            {
                a[i]++;
            }
            document.write ("the searched element is"+a[i] );
        }
    </script>
</html>

```

Objects in java script: JavaScript uses objects **to perform many tasks and therefore it is referred to as object-based programming language.** JavaScript objects are

Date object:

1. The date object are used **for obtaining the date and time.**
2. The date objects are created with new Date().

Methods	Description
Date()	Returns date object
getDate()	Returns the date of Date object(from 1-31)
getDay()	Returns the day of Date object(from 0-6, where 0=Sunday, 1=Monday etc)
getMonth()	Returns the month of Date object(from 1-11)
getFullYear()	Returns the year of Date object(four digits)
getYear()	Returns the year of Date object(0-99)
getHours()	Returns the hours of Date object(0-23)
getMinutes()	Returns the minutes of Date object(0-59)
getSeconds()	Returns the seconds of Date object(0-59)

Example:

```

<html>
    <head>
        <title>array element</title>
    </head>
    <body bgcolor="red" onload="Dater()">
    <script type="text/javascript">
        function Dater()
        {
            var today=new Date();
            var yesterday=new Date();
            var diff=today.getDate()-1;
            yesterday.setDate(diff);
            document.write ("<h3>the date today is"+today+"</h3>");
            document.write ("<h3>the date yesterday was</h3>");
            document.write (yesterday+"");
        }
    </script>

```

```
</body>
</html>
```

Math object: The math object allows the programmer to perform many common mathematical calculations. The object methods are called by writing the name of the object followed by dot and the name of the method. The argument or argument list are written in the method parenthesis.

Example: math.sqrt(64);

Methods	Description
abs(x)	Absolute value of x
ceil(x)	Rounds x to the smallest integer not less than x
cos(x)	Trigonometric cosine of x
exp(x)	Exponential method e to the power of x
floor(x)	Rounds x to the largest integer not greater than x
log(x)	Natural logarithm of x (base e)
max(x,y)	Larger value of x and y
min(x,y)	smaller value of x and y
pow(x,y)	X raised to the power of y
round(x)	Round x to the closest integer
sin(x)	Trigonometric sine of x
sqrt(x)	Square root of x
tan(x)	Trigonometric tangent of x

Example:

```
<html>
  <head>
    <title>Random numbers generation</title>
    <script type="text/javascript">
      var value;
      document.writeln("<table border='1' width='50%'>");
      document.write("<caption>random numbers</caption><tr>");
      for(var i=1;i<=20;i++)
      {
        value=Math.floor(1+Math.random()*6);
        document.writeln("<td>" + value + "</td>");
        if(i%5==0&&i!=20)
          document.writeln("</tr><tr>");
      }
      document.writeln("</tr></table>");
    </script>
  </head>
  <body><p>click here to refresh page</p></body>
</html>
```

Boolean object: JavaScript provides the Boolean object wrappers for Boolean values (true or false). when a JavaScript program requires a Boolean value, JavaScript automatically creates a Boolean object to store the value. JavaScript programmers can create Boolean objects explicitly with the statement.

Syntax: var b=new Boolean(Boolean value);

Method	Description
toString()	Returns the string true if the value of Boolean object is true; otherwise returns the string false.
valueOf()	Returns the value true if the Boolean object is

	true ; otherwise returns false.
--	--

Document object: It is used for **manipulating the document that is currently visible in the browser window.**

Method	Description
write(string)	Write the string to the html document as html code
writeln(string)	Write the string to the html document as html code and adds a new character at the end.
properties	Description
document.cookie	This property is a string containing the values of all the cookies stored on the user's computer for the current document.
document.lastmodified	This property gives the date and time that this document was modified.

String object: A string is a series of characters. A string is a object of string. A string is written in a double quotations marks as example "write your own string here".

Method	Description
charAt()	Returns the character at the specified index (position).
concat()	Joins two or more strings, and returns a new string
indexOf()	returns the index of the first occurrence of a specified text in a string
lastIndexOf()	returns the index of the last occurrence of a specified text in a string
slice()	Extracts a part of a string and returns a new string
split()	Splits a string into an array of substrings
substr()	Extracts the string from a string, beginning at a specified start position, and through the specified number of character
substring()	Extracts the string from a string, between two specified indices
toLowerCase()	Converts a string to lowercase letters
toString()	Returns the value of a String object
valueOf()	Returns the primitive value of a String object
toUpperCase()	Converts a string to uppercase letters

Window Object: It is used to create a new window

Method	Description
<u>alert()</u>	Displays an alert box with a message and an OK button
<u>close()</u>	Closes the current window
<u>confirm()</u>	Displays a dialog box with a message and an OK and a Cancel button
<u>focus()</u>	Sets focus to the current window
<u>open()</u>	Opens a new browser window
<u>prompt()</u>	Displays a dialog box that prompts the visitor for input
<u>resizeBy()</u>	Resizes the window by the specified pixels
<u>resizeTo()</u>	Resizes the window to the specified width and height
<u>stop()</u>	Stops the window from loading

Program: login page validation using JavaScript

```

<html>
  <head>
    <title>Login Form Validation</title>
    <script type="text/javascript">
      function validate()
      {
        if( document.myForm.username.value == "" )
        {
          alert( "Please provide your name!" );
          document.myForm.Name.focus() ;
          return false;
        }
        if( document.myForm.password.value == "" )
        {
          alert( "Please provide your password" );
          document.myForm.password.focus() ;
          return false;
        }
        return true;
      }
    </script>
  </head>
  <body>
    <form name="myForm"onsubmit="return(validate());">
      <table cellspacing="2" cellpadding="2" border="1">
        <tr>
          <td align="right">username</td>
          <td><input type="text" name="username" /></td>
        </tr>
        <tr>
          <td align="right">password</td>
          <td><input type="password" name="password" /> </td>
        </tr>
        <tr>
          <td align="right"></td>
          <td><input type="submit" value="Submit" /></td>
        </tr>
      </table>
    </form>
  </body>
</html>

```

Program: registration page validation using JavaScript

```

<html>
  <head><title>Validation form</title>
  <script type="text/javascript">
    function validateForm()
    {
      var uidlen=document.form1.uid.value.length;
      if(uidlen==0 || uidlen <6 || uidlen>12)

```

```

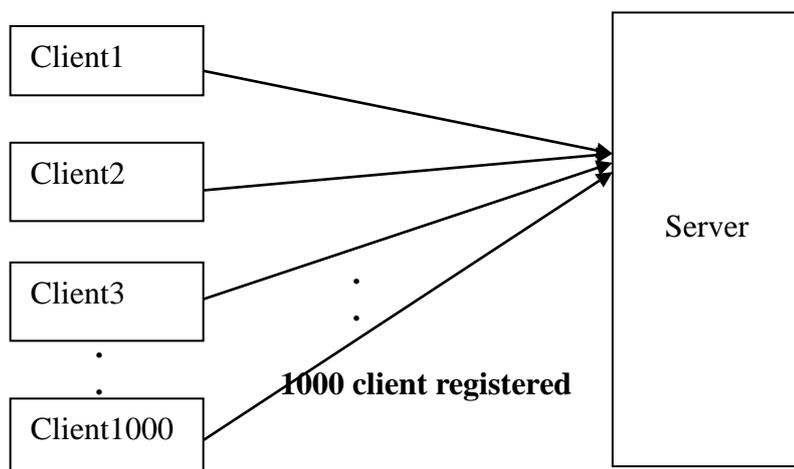
        {
            alert("user id must be between 6 to 12 characters");
            document.form1.uid.focus();
            return false;
        }
        var uname_re=/^[a-z A-Z]+$/;
        if(!document.form1.uname.value.match(uname_re))
        {
            alert("enter username chars only");
            document.form1.uname.focus();
            return false;
        }
        var passlen=document.form1.pwd.value.length;
        if(passlen == 0 || passlen < 6 || passlen > 12)
        {
            alert("password length between 6 to 12 only");
            document.form1.pwd.focus();
            return false;
        }
        var add_re=/^[0-9 a-z A-Z]+$/;
        if(!document.form1.add.value.match(add_re))
        {
            alert("enter alphanumeric only");
            document.form1.add.focus();
            return false;
        }
        var zip_re=/^[0-9]+$/;
        if(!document.form1.zip.value.match(zip_re))
        {
            alert("please enter the numerics only");
            document.form1.zip.focus();
            return false;
        }
        var email_re=/^\w+([\.,_]?w+)*@\w+([\.,_]?w+)*$/;
        if(!document.form1.email.value.match(email_re))
        {
            alert("please enter correct mail only");
            document.form1.email.focus();
            return false;
        }
        return true;
    }
</script>
</head>
<body>
    <form name="form1" onsubmit="return validateForm()">
        <table bgcolor="cyan" cellpadding="10" cellspacing="10">
            <tr>
                <td>userid</td>

```

```

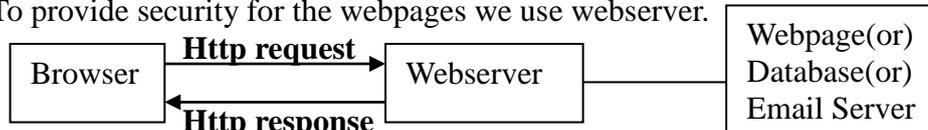
        <td><input type="text" id="uid"></td>
    </tr>
    <tr>
        <td>username</td>
        <td><input type="text" id="uname"></td>
    </tr>
    <tr>
        <td>password</td>
        <td><input type="password" id="pwd"></td>
    </tr>
    <tr><td>address</td>
        <td><textarea id="add"rows="5"cols="5">
            </textarea></td>
    </tr>
    <tr>
        <td>zipcode</td>
        <td><input type="text" id="zip"></td>
    </tr>
    <tr><td>email</td>
        <td><input type="text" id="email"></td>
    </tr>
    <tr>
        <td><p align="center"><input type="submit"
            value="submit"></p></td>
    </tr>
</table>
</body>
</html>
    
```

Webserver:



➤ Webpage of a company can be accessed by all the employees, then those requires the physical path of webpages, for this case there is no security for the webpages.

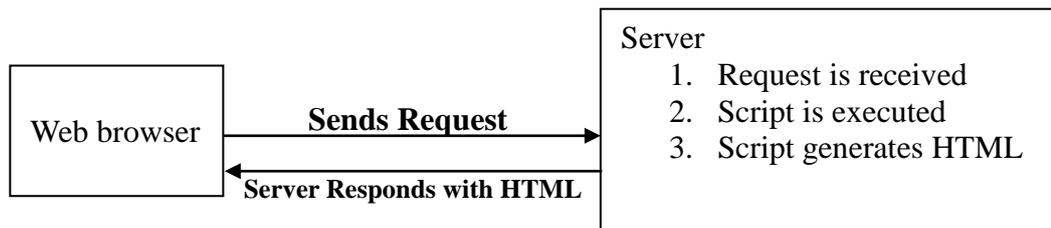
➤ To provide security for the webpages we use webserver.



➤ Webserver is software to manage webpages, provides security for web pages and supports

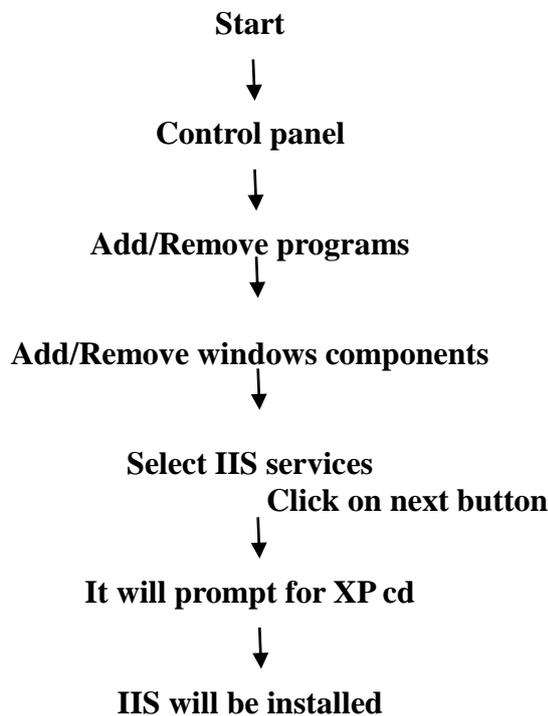
server side code execution.

- Webserver accepts http request and provides http response.
- Http stands Hypertext transfer protocol (protocol is set of rules).
- The rules obeyed by the browser and webserver towards communication is called HTTP.

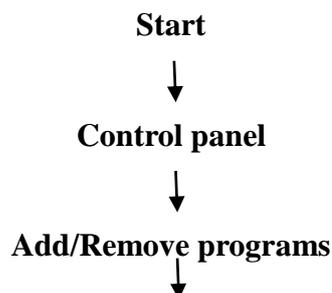


- There are several web services available in the market. They are
 1. Apache webserver
 2. Java webserver
 3. IIS server(internet information services)
 4. Tomcat webserver
- ASP.NET towards windows requires IIS5.0 or later versions.
- ASP.NET towards Linux requires apache webserver.

INSTALLING IIS TOWARDS WINDOWS XP:



INSTALLING IIS TOWARDS WINDOWS 7:



Turn on or off windows features



Select IIS and ASP.NET options

CHECKING IIS FUNCTIONALITY:

go to browser



<http://localhost> (use this for personal)

or

<http://servername> (It can be used for personal pc or network pc)

or

IP Address

WEBSITE:

- Website is an application managed by webserver to maintain physical path of web pages.
- IIS installation comes with default website that will map to physical path that is **c:\inetpub\wwwroot**.
- Microsoft is providing **internet services manager tool** (inetmgr.exe) to view the IIS information.

Start



Run



inetmgr

Click on ok



Internet Information Services

-syst(servername)

-websites

Default websites



Right click and select properties



Select home directory path tab

- Default webpages should be placed into **wwwroot** folder (example: **c:\inetpub\wwwroot\program.html**).

CREATING WEBPAGE TOWARDS DEFAULT WEBSITE:

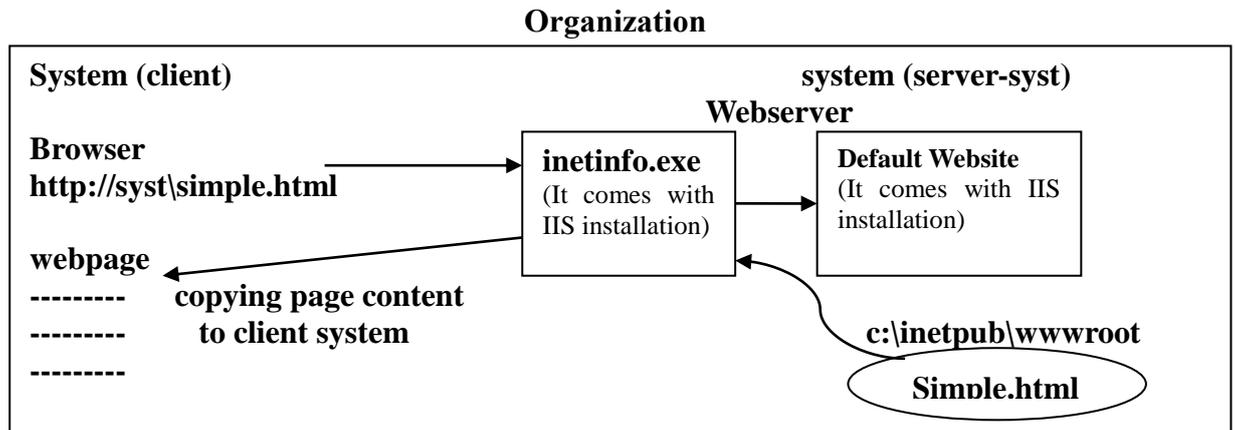
```
<html>
```

```
  <head><title> simple web page</title></head>
```

```
  <body><h1>webpage created in default website</h1></body>
```

```
</html>
```

- The above file will save with simple.html extension in **wwwroot** folder.
- Consider the organization is placing set of pages within website to provide information to employees.
- Here Client system requires only browser to access the webpages of company through webserver.



- Client request to webserver and the request will be received by inetinfo.exe.
- inetinfo.exe is looking for requested webpage within the physical path of default website, then the webpage content will be copied into client system.
- This webserver approach does not allow client to access physical path of webpage, this provides security for webpages.

SERVER SIDE SCRIPTING:

- The script which is executed by webserver is called server side scripting.
- Server side scripts will written in languages such as php,perl,ASP.NET, they are executed by the webserver when the user request a document and they produce output in a format which is understandable by web browsers , which is sent to the user's computer.
- Server side scripts have greater access to the information and functions available on the server.
- Server side scripts require that their language's interpreter to be installed on the server and produce the same output regardless of the clients browser.
- The code execution on webserver system is called server side code execution.
- There are different technologies available for server side scripting. They are
Example: JSP, PHP, ASP3.0, ASP.NET
- Server side scripting is used to provide interactive websites that interface to databases or other data stores.
- The advantage of server side scripting is the ability tightly customizes the response based on the user's requirements, access rights or queries into data sores.
- Server side scripts are never visible to the browser and these scripts are executed on the server.

HISTORY OF SERVER SIDE PROGRAMMING TECHNOLOGIES:

- The web began in the year 1990's.
- In beginning it was only possible to serve static webpages (or) pages that could not accept

user input and change according to user input.

- Now the webpages are dynamic and interactive to the user.

CGI (COMMON GATEWAY INTERFACE):

- CGI is built-in into almost all webservers.
- CGI is a method of obtaining and returning information to and from the webserver.
- CGI is not a language, but it an interface to the server.
- CGI application can be programmed with variety of languages.
- The most common language used in CGI programming is PERL (practical extraction and report language).
- Another common language for CGI programming is c which offers several performance advantages over PERL.
- CGI is platform independent, so CGI scripts will run with little change over multiple platforms such as UNIX and WINDOWS.
- CGI applications hard to maintain and very difficult to debug.
- A disadvantage of a CGI application (or "executable file," as it is sometimes called) is that each client request for a CGI requires a new process to be created in its own address space, resulting in extra instructions that have to be performed, especially if many client requests are running.

ISAPI (INTERNET SERVER APPLICATION PROGRAMMING INTERFACE):

- To overcome the disadvantage in CGI Microsoft has introduced ISAPI.
- ISAPI applications are compiled as windows dynamic link libraries (dll).
- ISAPI applications run in the IIS process and address space which allows faster execution.
- There are two types of ISAPI applications
 - 1. ISAPI Filters**
 - 2. ISAPI Extensions**
- ISAPI filters can run in the background and can monitor web server requests and performs a variety of tasks accordingly.
- The example of ISAPI filter is asp.dll filter, which monitor requests for files with an extension “.asp”, which process the file using the variables it has access to the webserver and returns the resulting page to IIS and IIS will sent to the client.
- ISAPLI extensions are equivalent of CGI executables and run upon client request.
- ISAPI extensions have access to important variables into the request and are used to perform database interaction, business logic

IDC (INTERNET DATABASE CONNECTOR):

- IDC is used to provide easy database connectivity for webpages and it is included in IIS.
- Each webpage requires two files

Query file: It contains database connection and sql query information.

Template file: It contains HTML and Tags which denotes the specific data from the query file must be inserted.

- The disadvantage of IDC is there is no place to business logic.

ASP (ACTIVE SERVER PAGE):

- Microsoft provides ASP which has best features of IDC and ISAPI.
- ASP page offers the usage of IDC by allowing database and business logic code to be inserted directly into pages using special tags and ASP offers performance advantages of ISAPI.
- ASP offers completely new paradigm for dynamic webpage development when it was introduced.
- ASP depends on interpreted programming model, which affects its performance.
- If ASP code is heavily used in webpages, this can create a number of complications.
- Most of the design tools do not recognize ASP code.
- If the design teams hand code their html, they may Un wittingly remove the code on the page this is the major problem in ASP.

ASP.NET:

- ASP.NET is a technology which **provides set of specifications for building web based applications towards server side execution.**
- ASP.NET takes **object oriented approach to webpage creation and every element in ASP.NET page is treated as an object and run on the server.**
- ASP.NET page gets **compiled into an intermediate language by a .NET CLR-complaint compiler, then a JIT compiler converts the intermediate code into native code** and that machine will run on processor.
- ASP.NET is a part of Microsoft .NET vision.
- ASP.NET provides two things
 1. **.NET languages programming towards server side coding:** It supports VB.NET/C# programming.
 2. **Collection of objects called ASP.NET intrinsic objects** (response, request, session, application, server and cache).

ASP.NET=.NET LANGUAGES PROGRAMMING+COLLECTION OBJECTS

IMPORTANT FACTS ABOUT ASP.NET:

1. **ASP.NET is integrated within the .NET framework:** The .NET framework allows us to develop console, windows and web application. This means ASP.NET is a part of entire framework of .NET
2. **ASP.NET is compiled, not interpreted:** ASP.NET has two stages of compilation
 - C# code is compiled into MSIL code.
 - MSIL code is compiled into low-level machine code.
3. **ASP.NET is Multilanguage:** to use any .NET supported programming language to create ASP.NET web pages, but the default languages are C# and VB.NET.
4. **ASP.NET is object oriented:** ASP.NET has full access to all objects in the .NET framework and you can also use all oops concepts.
5. ASP.NET applications are easy to deploy and configure.
6. **ASP.NET is multidevice and multibrowser:** ASP.NET supports different browser brands, versions and configurations.

NEW FEATURES IN ASP.NET:

1. Better language support.
2. Programmable controls.
3. Event-driven programming.
4. XML based components.
5. User authentication with accounts
6. High scalability.
7. Increased performance.
8. Easy to deploy and configure.

LANGUAGE SUPPORT:

1. ASP.NET supports ADO.NET
2. ASP.NET supports full VB
3. ASP.NET supports C# and C++
4. ASP.NET supports JavaScript.

ASP.NET SUPPORTS DIFFERENT TYPES OF WEB BASED APPLICATION DEVELOPMENT:

1. **Normal web application:** Developing dynamic webpages like login page.
2. **Web services:** developing business logic to integrate different vendor applications (Example: J2EE, Oracle).
3. **AJAX (Asynchronous JavaScript and XML) Web application:** Developing web pages by implementing Ajax specifications, the importance of AJAX is avoiding complete page submission.
4. **Silverlight Integration:** It supports rich internet application development. It means

implementing shapes, animations, video streaming into webpage development.

5. **MVC application (Model View Controller):** Developing web pages with the concept of MVC, the advantage is testing will become easier.

NOTE: ASP.NET 4.0 is the current version which provides the following features

1. **url routing**
2. **chart control**
3. **dynamic data**
4. **Silverlight 3.0 integration**
5. **MVC integration**

ADVANTAGES USING ASP.NET:

- **ASP.NET reduces the amount of code required for building large and complex applications** which can increase the development speed and reduce the development cost.
- **ASP.NET provides cross platform migration.**
- **ASP.NET provides security for the applications with built-in windows authentication and per-application configuration.**
- **ASP.NET provides easy way to perform common tasks like site configuration, client authentication to deployment.**
- **ASP.NET provides better performance by taking the advantage of early binding, just-in-time compilation, native optimization.**
- **It is purely server side technology**, so ASP.NET code executes on the server before it is sent to the browser.
- **ASP.NET provides easy deployment .There is no need to register components because the configuration information is built-in.**
- **ASP.NET easily works with ADO.NET using data binding and page formatting features.**
- **ASP.NET application will run faster and it allows large number of users without having performance problems.**

ASP.NET VERSIONS: The version release of ASP.NET is closely tied with the release of .NET Framework. The different versions of ASP.NET are

ASP.NET1.0:

- It was released in 2002 along with visual studio.NET
- It supports object oriented features such as polymorphism, inheritance, early binding etc.

ASP.NET1.1:

- It was released in 2003 along with visual studio.NET 2003
- It supports features like automatic input validation, mobile controls.

ASP.NET2.0:

- It was released in 2005 along with visual studio 2005 .
- It supports features like GridView data control, FormView data control, navigation controls, login controls etc

ASP.NET3X:

- It was released in 2008 along with visual studio 2008 and visual web developer express
- It supports features like ListView data control,AJAX,WCF,LINQ

ASP.NET4.0:

- It was released in 2010 along with visual studio 2010.
- It supports Parallel extensions and other features of .NET framework4

ASP.NET NAMESPACES:

- Microsoft defines namespaces as a **logical naming scheme for grouping related types** that is all objects used in ASP.NET are grouped by type, making them easy to find and use.
- Namespaces are **logical collection objects**.
- To use a namespace in an **ASP.NET page ,you must use the Import directive**

Example: `<%@import Namespace="System.Data"%>`

This tells the CLR to reference this namespace when it compiles your ASP.NET application and objects in the namespaces are dynamically loaded when they are called in webpages.

- **The most commonly used namespaces in ASP.NET are**
 1. **System:** It is the **root namespace** for the entire .NET framework and it contained all **basic and generic classes used in ASP.NET**.
 2. **System.Collections:** It contains the **functionality for grouping objects and data types into collections**.
 3. **System.Data:** It contains various **database connectivity methods**.
 4. **System.Web:** It defines types that **enable browser and web server communication**.
 5. **System.Web.SessionState:** It defines types that allow you to **maintain state full data across a session**.
 6. **System.Web.UI, System.Web.UI.WebControls, System.Web.UI.HtmlControls:** These namespaces define a number of types that allow you to **build GUI front end for your web application**.
 7. **System.Web.Caching:** It contains a type that **provides caching support for a web application**.
 8. **System.Web.Security:** It provides **security support for web application**.
 9. **System.Web.Configuration:** It provide types that allow you to **configure your web application in conjunction with projects web.config file**
 10. **System.Web.Services, System.Web.Services.Description, System.Web.Services.Discovery, System.Web.Services.Protocols:** These namespaces provide the types that allow you to **build web services**.

ASP.NET WEB PAGE STRUCTURE:

1. HTML page
↓
HTML+client side scripting

2. ASP3.0 page
↓
HTML+client side scripting
+
Server side scripting (VB script/javascript)

3. ASP.NET page
↓
HTML+client side scripting
+
Server side scripting (VB.NET/C#)

CREATING ASP.NET WEBPAGE USING NOTEPAD: First open the notepad there write the following code

```

<html>
  <body>
    <h1>
      <%
        response.write("ASP.NET page with server side script");
      %>
    </h1>
  </body>
</html>
    
```

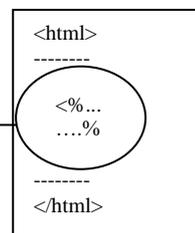
Save this file as first.aspx(c:\inetpub\wwwroot)

Browser

http://localhost/first.aspx

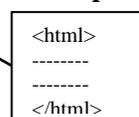
webserver

first.aspx



Execute on the server(server side execution)

Output



➤ When client request comes to asp.net page, webserver will execute only server side logic and

output will be produced to client browser. The output will contain only HTML content.

- ASP.NET extension is .aspx(active server page execution).
- response.write method can be used for displaying message to the user.
- ASP.NET page execution takes place at 2 places
 1. Webserver will execute server side present in page.
 2. Browser will execute html tags and client side script given by the user.

CREATING ASP.NET WEBPAGE USING C#:

First open the notepad there write the following code

```
<% @ page language="c#"%>
<html>
  <body>
    <h1>
      <%
        response.write("ASP.NET page with server side script");
      %>
    </h1>
  </body>
</html>
```

Save this file as C#page.aspx(c:\inetpub\wwwroot)

Page Directive:

- Directive is an instruction to the server, it should be the first statement within page.
- Page Directive can be used to provide different instructions to server, which can changing language, attaching theme, etc.

Syntax: <% @ directive name options %>

IDE FOR ASP.NET(or) DESIGNER FOR ASP.NET:

- Microsoft is providing two IDE'S for ASP.NET
 1. **VISUAL STUDIO:** It supports different types of .NET application developments
Example: GUI,CUI,WEB
 2. **VISUAL WEB EXPRESS:** It supports only web application development

CREATING WEB PAGE USING VISUAL STUDIO:

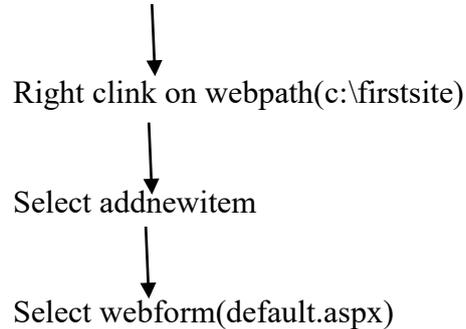
- Visual studio is provided in the form of devenv.exe file.
- Visual studio supports asp.net web page creation in the form of website development.
- Start==>Run==>devenv
- It will display main window of the visual studio.
- Goto file menu==>new==>website

Select ASP.NET Empty Website
↓
Language C#

- Choose web location ==> file system
- Select path where we can store and click ok , this will create a website(c:\firstsite).

Placing asp.net page into website:

- Go to view menu==> select solution explorer



- Go to source view, place server side code below body tags.

Example: <body>

<h1>

<%

Response.Write("ASP.NET webpage using visual studio");

%>

</h1>

</body>

- Go to debug==>select run without debugging.
- Visual studio will create a folder towards website; in this all the website related files will be stored.
- Visual studio supports website development towards three locations.
 1. **File System:** website will be managed by ASP.NET development server. This server comes with visual studio. It is a part of visual studio.

Advantages:

- Webpages will be accessible with in the network
- It supports all advanced options(webservices, security)

Disadvantages:

- IIS installation is required.
- ASP.NET to be configured properly with IIS.

2. **HTTP:** website will be managed by IIS webserver.

Advantages:

- IIS installation is not required.
- ASP.NET configuration is not required explicitly.

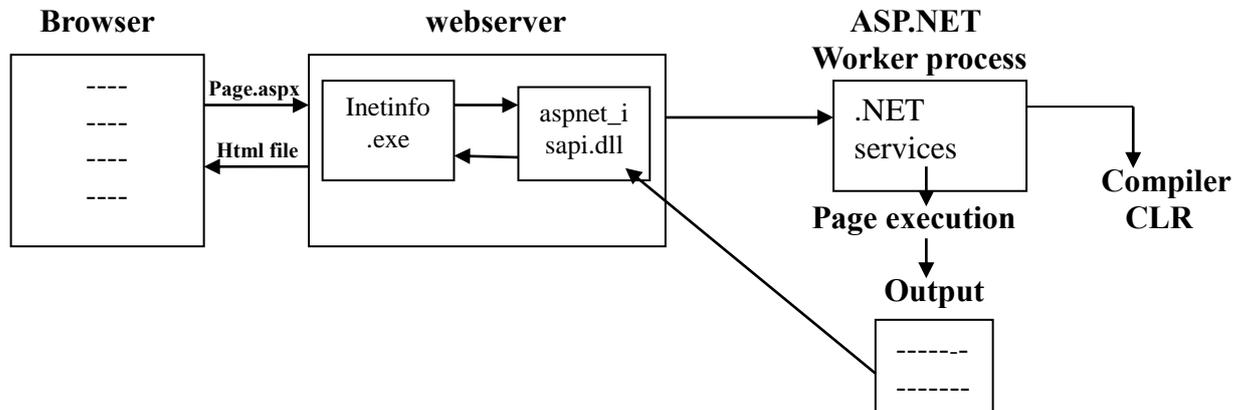
Disadvantages:

- Webpages are not accessible with in the network
- It doesn't support advanced options(web services,security)

3. **FTP:**It allows developing websites in one system and stored in another system.

ASP.NET WEBPAGE EXECUTION ARCHITECTURE:

- Server system **requires IIS installation and .NET installation.**
- **.NET installation will configure IIS with aspnet_isapi.dll.**
- **Aspnet_isapi.dll acts like mediator between webserver and .NET**



- Client sends request to webserver then that request is received by the inetinfo.exe, it will verify the requested file extension.
- inetinfo.exe will send that request to asp.net worker process with the help of aspnet_isapi.dll.
- ASP.NET worker process can be called as **asp.net runtime or asp.net engine**, it provides all the **.NET services required for asp.net page execution.**
- **.NET services includes language compiler and CLR.**
- After asp.net is executed,it will give output in the form of html filethat will sent back to browser.
- ASP.NET worker process will perform the following steps towards first request to webpage.

1. It will convert asp.net webpage into a class

Example: first.aspx → class first.aspx

```
{
    -----
    -----
}
```

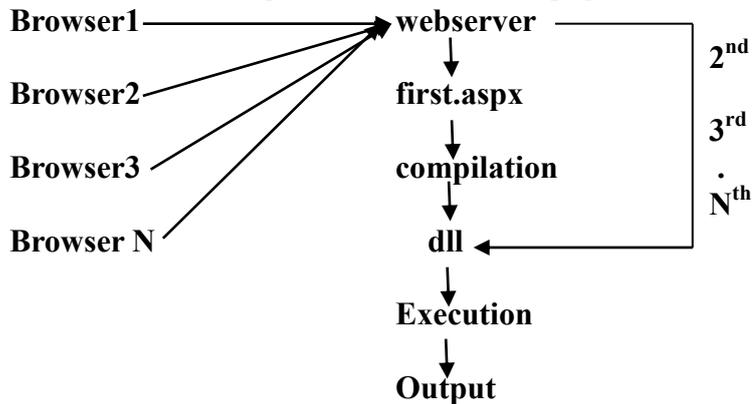
2. It will compile webpage class into a dll

Example: class first.aspx

```
{
    ---
    --- ==>C# Compiler ==> dll(IL code)
}
```

3. It will execute dll(IL code) to produce output to the browser.

- ASP.NET worker process will perform compile code execution that is dll(IL code) execution for 2nd, 3rd ...nth request to ASP.NET webpage.



TRACING AND DEBUGGING ASP.NET PAGE:

Debugging ASP.NET page:

- Debugging is the process of finding errors.
- An ASP.NET debug page is displayed when an unhandled error is encountered. Debugging can be applied in two levels.

1. Page level debugging:

- To enable debugging at the page level we use @page directive.

Example: `<%@ page language="C#" debug="True">`

2. Application level debugging:

- To enable debugging at the application level we can set the debug attribute in the compilation section at web.config.

Example: `<compilation default language="C#" debug="true">`

- The debug object is the member of System.Diagnostics.Debug.
- The method for debug is

debug.Write("application initializing");

Tracing ASP.NET page:

- Tracing is the concept of getting complete information towards request execution.
- Tracing will provide request details, trace information.
- Tracing can be applied at two levels

1. Page level tracing:

- If the tracing is applied to webpage is called page level tracing.
- It requires trace option in page directive

Syntax: `<%@ page.....trace="true">`

- It can be used to **display order of events execution and display the controls with memory size towards specific webpage.**

2. Application level tracing:

- If the tracing is applied to a website is called application level tracing.
- It requires setting in web.config

Syntax: `<trace enabled="true"`

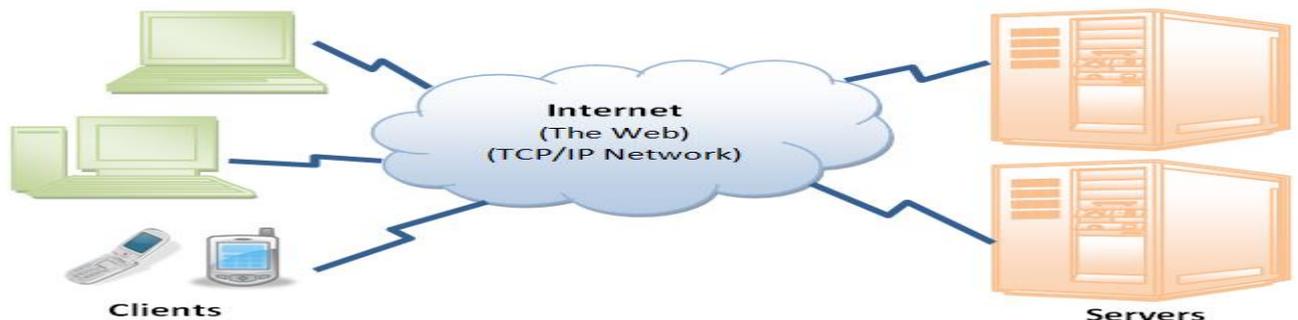
`localonly="true/false"`

`pageoutput="true/false"`

`requestlimit="number" />`

- Page level tracing **does not provide options for live websites.**

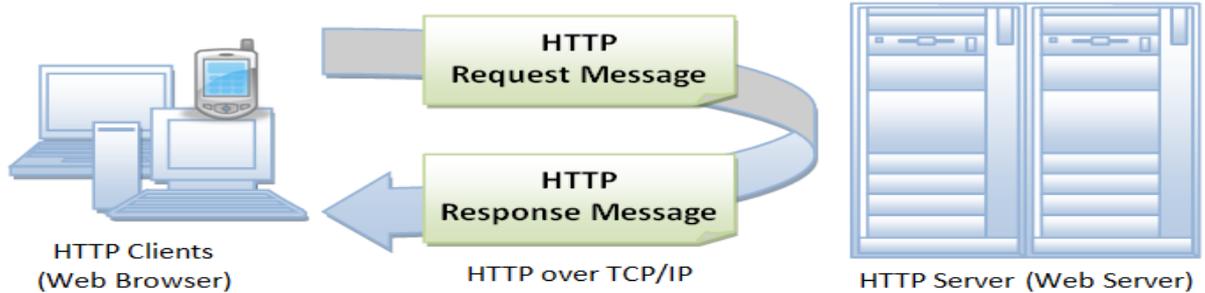
WEB: Internet (or The Web) is a massive distributed client/server information system.



Many applications are running concurrently over the Web, such as web browsing/surfing, e-mail, file transfer, audio & video streaming, and so on. In order to provide proper communication between the client and the server, these applications must agree on a specific application-level protocol such as HTTP, FTP, SMTP, POP, and etc.

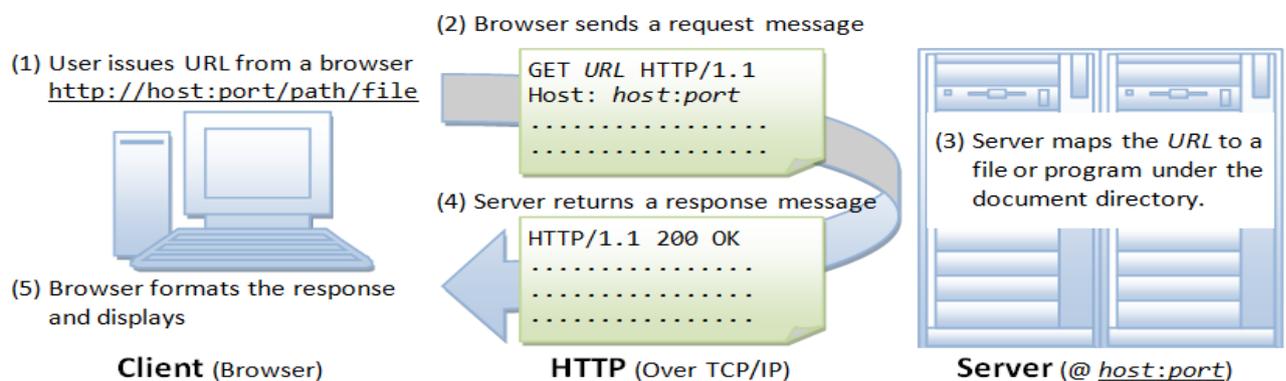
HyperText Transfer Protocol (HTTP):

- HTTP (Hypertext Transfer Protocol) is most popular application protocol used in the Internet (or The WEB).
- HTTP is an *asymmetric request-response client-server* protocol. An HTTP client sends a request message to an HTTP server. The server, in turn, returns a response message. In other words, HTTP is a *pull protocol*, the client *pulls* information from the server (instead of server *pushes* information down to the client).



HTTP is a stateless protocol. In other words, the current request does not know what has been done in the previous requests.

Browser: Whenever you issue a URL from your browser to get a web resource using HTTP, e.g. `http://www.nowhere123.com/index.html`, the browser turns the URL into a *request message* and sends it to the HTTP server. The HTTP server interprets the request message, and returns you an appropriate response message, which is either the resource you requested or an error message. This process is illustrated below:



Uniform Resource Locator (URL): A URL (Uniform Resource Locator) is used to uniquely identify a resource over the web.

Syntax: *protocol://hostname:port/path-and-file-name*

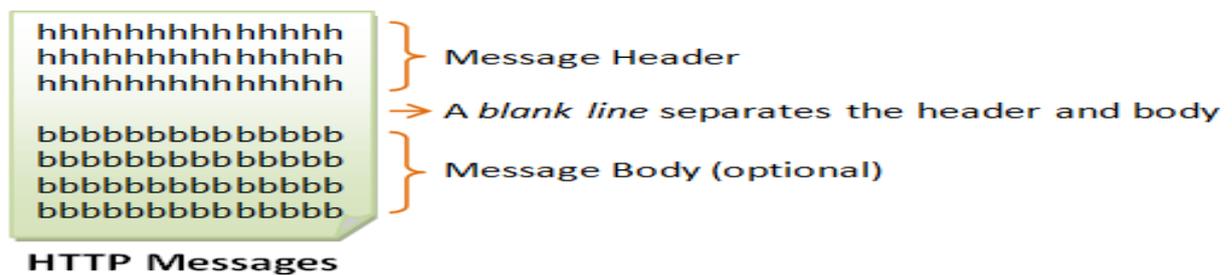
There are 4 parts in a URL:

- **Protocol:** The application-level protocol used by the client and server, e.g., HTTP, FTP, and telnet.
- **Hostname:** The DNS domain name (e.g., `www.nowhere123.com`) or IP address (e.g., `192.128.1.2`) of the server.
- **Port:** The TCP port number that the server is listening for incoming requests from the clients.
- **Path-and-file-name:** The name and location of the requested resource, under the server document base directory.

For example, in the URL `http://www.nowhere123.com/docs/index.html`, the communication protocol is HTTP; the hostname is `www.nowhere123.com`. The port number was not specified in the

URL, and takes on the default number, which is TCP port 80 for HTTP. The path and file name for the resource to be located is `"/docs/index.html"`.

HTTP Request and Response Messages: HTTP client and server communicate by sending text messages. The client sends a *request message* to the server. The server, in turn, returns a *response message*. An HTTP message consists of a *message header* and an optional *message body*, separated by a *blank line*, as illustrated below:



HTTP Request Message: The format of an HTTP request message is as follow:



Request Line: The first line of the header is called the request line, followed by optional request headers.

syntax: *request-method-name request-URI HTTP-version*

- **request-method-name:** HTTP protocol defines a set of request methods, e.g., GET, POST, HEAD, and OPTIONS. The client can use one of these methods to send a request to the server.
- **request-URI:** specifies the resource requested.
- **HTTP-version:** Two versions are currently in use: HTTP/1.0 and HTTP/1.1.

Examples of request line are:

```
GET /test.html HTTP/1.1
```

```
HEAD /query.html HTTP/1.0
```

```
POST /index.html HTTP/1.1
```

Request Headers: The request headers are in the form of name:value pairs. Multiple values, separated by commas, can be specified.

Syntax: *request-header-name: request-header-value1, request-header-value2, ...*

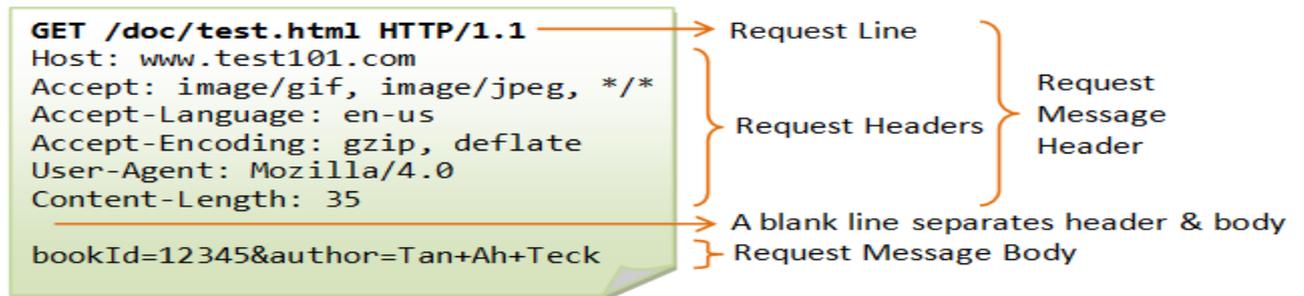
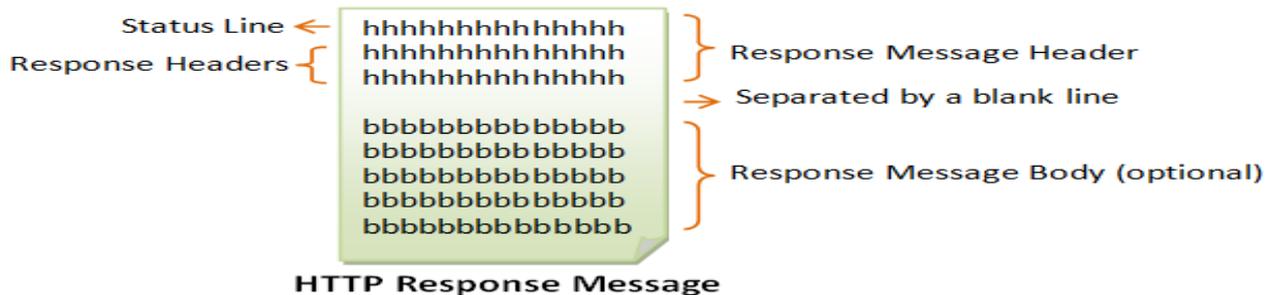
Examples of request headers are:

Host: www.xyz.com

Connection: Keep-Alive

Accept: image/gif, image/jpeg, */*

Accept-Language: us-en, fr, cn

Example: The following shows a sample HTTP request message:**HTTP Response Message:** The format of the HTTP response message is as follows:**Status Line:** The first line is called the status line, followed by optional response header(s).**syntax:** *HTTP-version status-code reason-phrase*

- **HTTP-version:** The HTTP version used in this session. Either HTTP/1.0 and HTTP/1.1.
- **status-code:** A 3-digit number generated by the server to reflect the outcome of the request.
- **reason-phrase:** Gives a short explanation to the status code.

Common status code and reason phrase are "200 OK", "404 Not Found", "403 Forbidden", "500 Internal Server Error".

Examples of status line are:

HTTP/1.1 200 OK

HTTP/1.0 404 Not Found

HTTP/1.1 403 Forbidden

Response Headers: The response headers are in the form name:value pairs:**Syntax:** *response-header-name: response-header-value1, response-header-value2, ...*

Examples of response headers are:

Content-Type: text/html

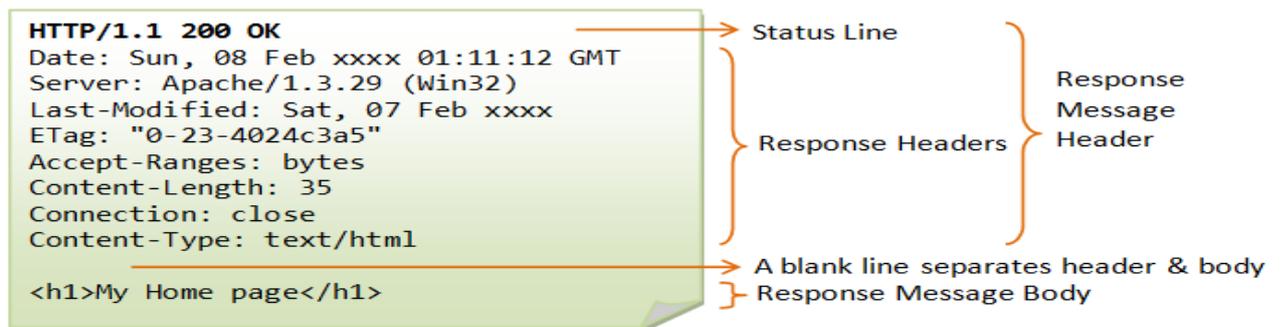
Content-Length: 35

Connection: Keep-Alive

Keep-Alive: timeout=15, max=100

The response message body contains the resource data requested.

Example: The following shows a sample response message:



UNIT-5

WEB FORM STRUCTURES:

- Visual studio provides **web Form** template to create ASP.NET webpage.
- The advantage of **web Form** template is what you see is what you get this means, the page appearance what you see at the time creation is what you will get when you run the web page with the browser.
- Web Form template provides three views.
 1. **Design view:** this view allows the developer to arrange controls.
 2. **Source view:** this view display all the tags generated by visual studio.
 3. **Code view:** this view allows the developer to place logic part within CS file.

SERVER SIDE CONTROLS:

- Control can be classified into two types
 - Client side control
 - Server side control
- The control which is created by the browser is called client side control.

```
<input type="text" name="t1">
```

It is an instruction to the browser to create textbox control or object.
- The client side control is not accessible within server side coding.
- ASP.NET provides server side controls. The control which is created by server side processing is called server side control, this makes developer job easier,more clarity in coding and better performance.
- Server side controls can be classified into two types
 - HTML server Controls
 - Web server controls

HTML SERVER CONTROLS:

- HTML server controls have similar syntax like client side controls to make migration easier.

```
<input type="text" RunAt="server">
```

- Providing RunAt="server" we can convert client side control into server side control.

WEB SERVER CONTROLS:

- ASP.NET is providing a set of controls towards rich presentation with ASP.NET webpage are called web server controls.
- Web server controls are called smart controls, because these controls will produce output based on requested browser type and browser settings
- Web server controls can be classified into following categories
 - **Standard controls**
Example: label, button, textbox...
 - **Validation controls**
Example: RequiredFieldValidator, RangeValidator...
 - **Data Controls**
Example: GridView, ListView...
 - **Navigation controls**
Example: Menu, treeview..
 - **WebPart controls**
Example: Webpart zone..
 - **Login controls**
Example: Create user wizard..
 - **AJAX Extension controls**
Example: Script manager..
- Web server controls supports extensibility, which means Microsoft can intraduce new controls from one version to another and developer can create user defined web server controls.
- Web server control syntax is


```
<asp: Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" / >
```
- The namespace for basic controls is System.Web.UI.WebControls.

ASP.NET WEB FORM STANDARD CONTROLS:

Button Controls: ASP.NET provides three types of button control

- **Button:** It displays text within a rectangular area.
- **Link Button:** It displays text that looks like a hyperlink.
- **Image Button:** It displays an image.

When a user clicks a button, two events are raised: Click and Command.

Basic syntax of button control:

```
<asp: Button ID="Button1" runat="server" onclick="Button1_Click" Text="Click" / >
```

Common properties of the button control:

Property	Description
Text	The text displayed on the button. This is for button and link button controls only.
ImageUrl	For image button control only. The image to be displayed for the button.
AlternateText	For image button control only. The text to be displayed if the browser cannot display the image.
CausesValidation	Determines whether page validation occurs when a user clicks the button. The default is true.
CommandName	A string value that is passed to the command event when a user clicks the button.
CommandArgument	A string value that is passed to the command event when a user clicks the button.
PostBackUrl	The URL of the page that is requested when the user clicks the button.

Text Boxes and Labels: Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

Basic syntax of text control:

```
<asp:TextBox ID="txtstate" runat="server" ></asp:TextBox>
```

Common Properties of the Text Box and Labels:

Property	Description
TextMode	Specifies the type of text box. SingleLine creates a standard text box, MultiLine creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine.
Text	The text content of the text box.

MaxLength	The maximum number of characters that can be entered into the text box.
ReadOnly	Determines whether the user can change the text in the box; default is false, i.e., the user can change the text.
Columns	The width of the text box in characters. The actual width is determined based on the font that is used for the text entry.
Rows	The height of a multi-line text box in lines. The default value is 0, means a single line text box.

The mostly used attribute for a label control is 'Text', which implies the text displayed on the label.

Check Boxes and Radio Buttons: A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group. If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

Basic syntax of check box:

```
<asp:CheckBox ID= "chkoption" runat= "Server">
</asp:CheckBox>
```

Basic syntax of radio button:

```
<asp:RadioButton ID= "rdboption" runat= "Server">
</asp: RadioButton>
```

Common properties of check boxes and radio buttons:

Property	Description
Text	The text displayed next to the check box or radio button.
Checked	Specifies whether it is selected or not, default is false.
GroupName	Name of the group the control belongs to.

List Controls: ASP.NET provides the following controls

- Drop-down list,
- List box,

- Radio button list,
- Check box list,
- Bulleted list.

These control let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the ListItemCollection editor.

Basic syntax of list box control:

```
<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
</asp:ListBox>
```

Basic syntax of drop-down list control:

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
</asp:DropDownList>
```

Common properties of list box and drop-down Lists:

Property	Description
Items	The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection.
Rows	Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added.
SelectedIndex	The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1.
SelectedValue	The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string ("").
SelectionMode	Indicates whether a list box allows single selections or multiple selections.

Common properties of each list item objects:

Property	Description
Text	The text displayed for the item.
Selected	Indicates whether the item is selected.
Value	A string value associated with the item.

It is important to notes that:

- To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.
- The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

Example: create a website to work with DropDownList and ListBox controls.

Procedure:

1. Select ASP.NET empty website .
2. Add web form
3. Place one dropdownlist, two listboxes and four button on default.aspx design view.

Properties:

DropDownlist1 properties:

- autopostback=true
- items= Microsoft
Microsoft1
Microsoft1

ListBox1 properties:

- selectMode=multiple

ListBox2 properties:

- selectMode=multiple

Button1 properties:

- Text =one(>)

Button2 properties:

- Text=multiple(>>)

Button3 properties:

- Text=removeone

Button4 properties:

- Text=removeall
- ```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace WebApplication2
{
```

```
 public partial class WebForm1 : System.Web.UI.Page
```

```

 {
 protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
 {
 ListBox1.Items.Clear();
 switch (DropDownList1.SelectedValue)
 {
 case "microsoft": ListBox1.Items.Add("10");
 ListBox1.Items.Add("20");
 break;
 case " microsoft1": ListBox1.Items.Add("100");
 ListBox1.Items.Add("200");
 break;
 case " microsoft2": ListBox1.Items.Add("1000");
 ListBox1.Items.Add("2000");
 break;
 }
 }
 protected void Button1_Click(object sender, EventArgs e)
 {
 ListBox2.Items.Add(ListBox1.SelectedItem);
 }
 protected void Button2_Click(object sender, EventArgs e)
 {
 for (byte i = 0; i < ListBox1.Items.Count; i++)
 ListBox2.Items.Add(ListBox1.Items[i]);
 }
 protected void Button3_Click(object sender, EventArgs e)
 {
 ListBox2.Items.RemoveAt(ListBox2.SelectedIndex);
 }
 protected void Button4_Click(object sender, EventArgs e)
 {
 ListBox2.Items.Clear();
 }
 }
}

```

**PAGE SUBMISSION:** Page submission is the concept of client submitting the data to webserver.

The page submission can be classified into two types.

- **Crosspage submission**
- **PostBack submission**

**CROSSPAGE SUBMISSION:**

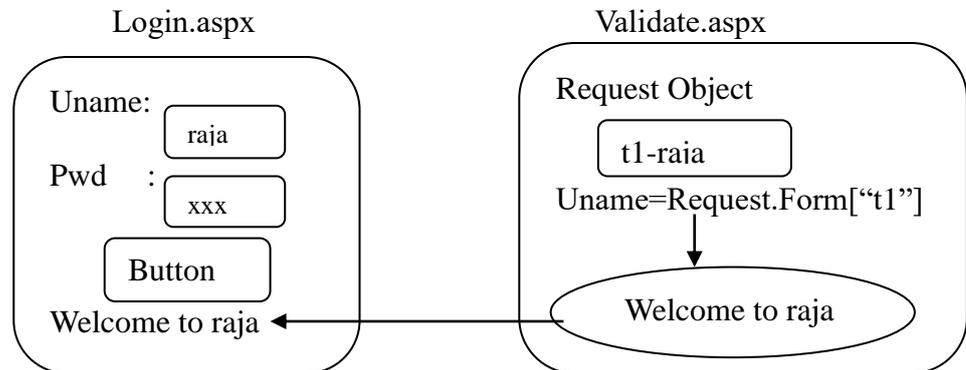
- One webpage is submitting the data to another webpage is called crosspage submission.
- Cross page submission implementation requires two webpages
  - Source webpage
  - Destination webpage
- Source webpage will accept data from user, this will submit data to destination webpage.

- Destination webpage will read data, reading data is possible using request object.

**Example:** Request.Form["controlid"]=> this can be used when the method is post

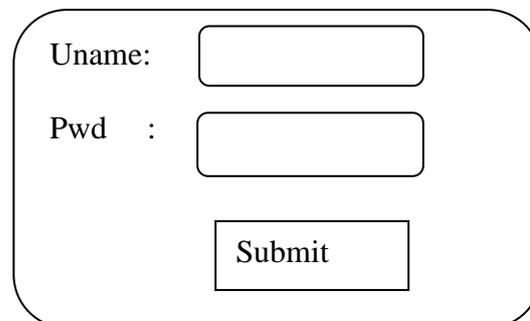
Request.Form["controlid"]=> this can be used when the method is get

- The default method in ASP.NET is post.



### Example: Creating a website to work with crosspage submission.

- Select ASP.NET empty website.
- Add web form and name it as login.aspx
- Goto design view of login.aspx



- Place two labels, two textboxes and one submit button
- **Label1 properties:**
  - id=l1
  - Text=username
- **Label2 properties:**
  - id=l2
  - Text=password
- **textBox1 properties:**
  - id=t1
- **textBox2 properties:**
  - id=t2
  - textmode=password
- **button1 properties:**
  - id=b1
  - text=submit
  - postbackurl=validate.aspx
- add webform(place one more web form into asp.net website) and name it as validate.aspx
- goto design view of validate.aspx

- place one label
- lebel1 properties
  - id=lb
- code for page load event
 

```
{
 String uname=Request.Form["t1"];
 String pwd=Request.Form["t2"];
 lb.Text="wel come to"+uname;
 }
```
- to set login page as starting page  
goto solution explorer=>right click on login.aspx=>set as start page
- ctrl+f5

**ANALYSING PAGE SUBMISSION:** page submission is depends on two things

- Arrangement of data for transmission
- Reading submitted data on server

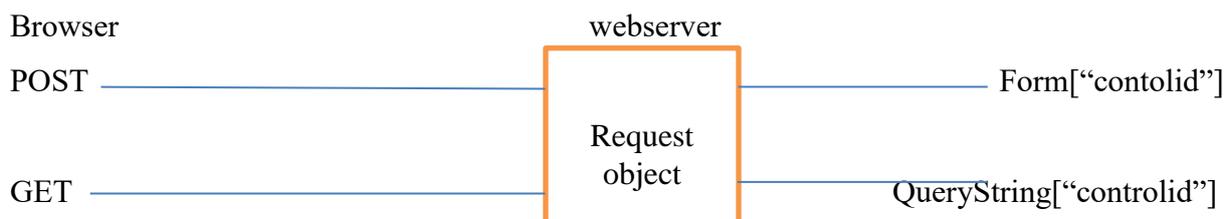
**Arrangement of data for transmission:** We have two methods for arranging data

- POST method
- GET method

| POST                                           | GET                                                                     |
|------------------------------------------------|-------------------------------------------------------------------------|
| Data will be arranged within HTTP message body | Data will be arranged within HTTP header(here data is appending to URL) |
| It supports large amount of data arrangement.  | It supports small amount of data arrangement(maximum of 2kb).           |
| It provides security for sensitive data.       | It does not provide security for sensitive data.                        |
| Transmission will be slow                      | Transmission will be faster                                             |

**Reading submitted data on server:**

- The reading procedure will be based on arrangement type.
  - POST arrangement==> Request.Form["controlid"]
  - GET arrangement==>Request.Form["controlid"]



- The default method in ASP.NET is post.

**TO CHANGE CROSSPAGE SITE FROM POST TO GET:**

- Open cross page website
- Goto login.aspx source view

- ```
<form method="GET">
```
- Goto validate.aspx
- ```
<%
 String username=Request.Form["t1"];
 String password=Request.QueryString["t2"];
%>
```
- Press ctrl+f5

**POST BACK SUBMISSION:** The page submitting to itself is called postback submission.

### BROWSER

Customer.aspx

ACCNO:

Display

postback submission

### WEBSERVER

Customer.aspx


**Example: Creating a website to work with postback submission**

- Select ASP.NET empty website.
- Add web form and name it as customer.aspx
- Go to design view of customer.aspx

|        |                      |         |
|--------|----------------------|---------|
| ACCNO: | <input type="text"/> | Display |
| NAME : | <input type="text"/> |         |
| BAL :  | <input type="text"/> |         |

- textBox2 properties:
  - visible=false
- textBox3 properties:
  - visible=false
- button1\_click event logic
 

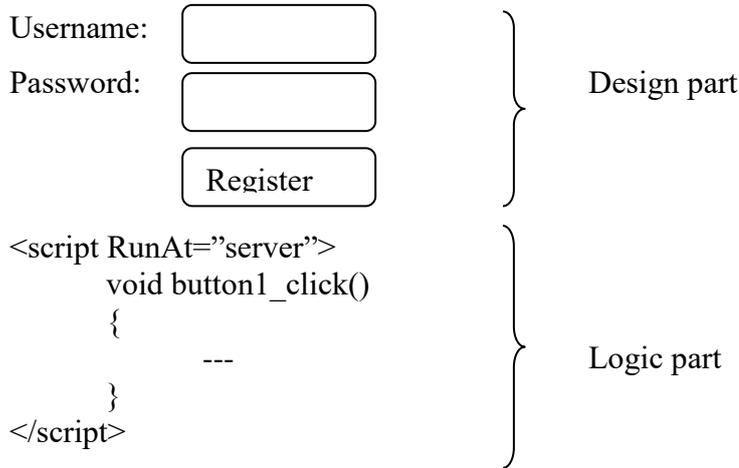
```
{
 String[,]={{"60","raja","5000"},"{"50","raja1","10000"}};
 bool b=false;
 if(s[I,0]==textBox1.Text)
 {
 textBox2.Text=s[i,1];
 textBox3.Text=s[i,2];
 textBox2.Visible=true;
 textBox3.Visible=true;
 b=true;
 break;
 }
}
if(b==false)
 Response.Write("accno not existing");
```
- ctrl+f5

### WEBPAGE CREATION TECHNIQUES:

- Web page development goes with design part and logic part.

- Web designer will handle design part and web developer will handle logic part.
- ASP.NET supports two techniques for webpage creation
  - In page Technique
  - Code file Technique or Code behind Technique

**Inpage Technique:** placing design part and logic part into aspx file is called inpage technique.



**Advantages:**

- Migration is easy that means converting ASP page to ASP.NET page will be easier.
- Deploying webpage into server will be easier.

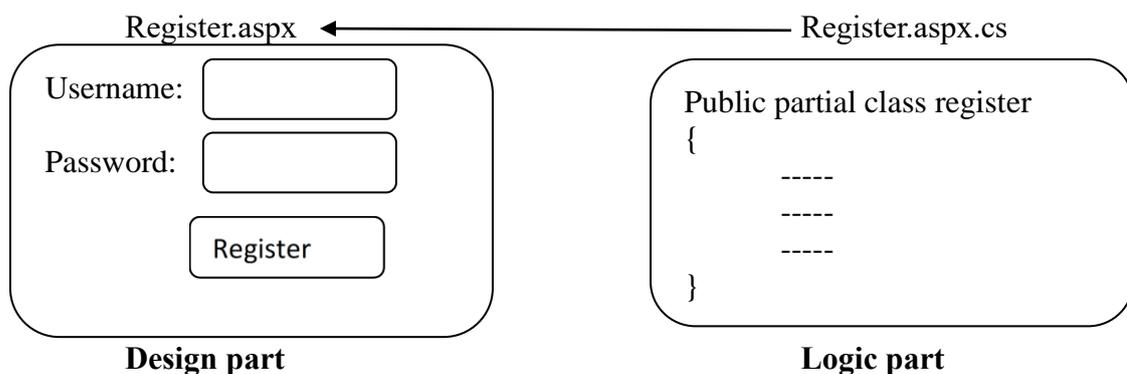
**Disadvantages:**

- Parallel development is not possible which makes development slow
- No security for logic part because web designer can view the logic part present under aspx file.

**Code file Technique or Code behind Technique:** placing design part code into aspx file and logic part code into cs file is called Code file Technique or Code behind Technique. The cs file can be linked with aspx file using page directive

**Example:** `<%@ page.....codefile="filename.cs"...%>`

page directive



**Advantages:**

- Parallel development is possible which makes development fast
- logic part will be secure because web designer unable to view the logic part present under aspx file.

#### DisAdvantages:

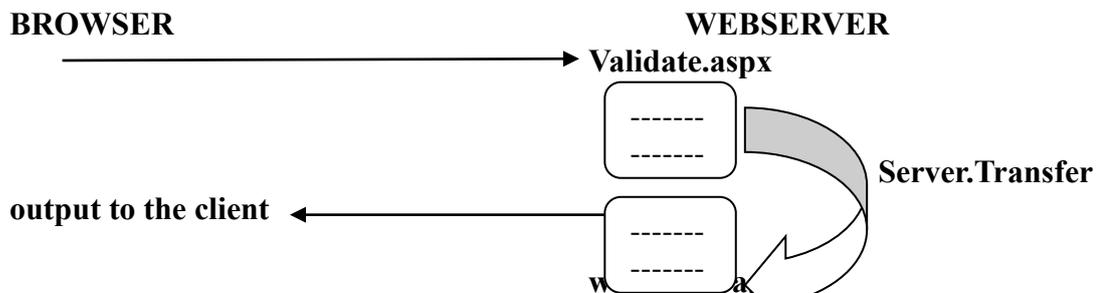
- Migration is not possible that means converting ASP page to ASP.NET page will be difficult.
- Deploying webpage into server will be difficult when compare to inpage technique.

**REDIRECTION BETWEEN WEBPAGES:** Redirection between webpages can be done in two ways

- Server.Transfer method
- Response.Redirect method

#### Server.Transfer method:

- Server.Transfer method will perform redirection from one page to another page within server side processing without informing to web browser.
- Server.Transfer method will give better performance over response.redirect method.
- Server.Transfer method requires one communication.
- Server.Transfer method supports redirection only within website.



**Example: Develop a website to work with textbox, button and Server.Transfer method**

1. Open new website==>select ASP.NET empty website and name it as response.
2. Right click on project path==>select add new item==>select webform and name it as login.aspx
3. Go to design view of login.aspx
4. Place one label,one textbox and one button

Enter login type:

#### textBox1 properties:

id=t1

5. Code for button1\_click event
 

```
{
 if(t1.text=="admin)
```

```

 Server.Transfer("admin.aspx");
 else if(t1.text=="normal")
 Server.Transfer("normal.aspx");
 else
 Response.Write("invalid login");
}

```

6. Add web form and name it as admin.aspx

**Label1 properties:**

Text=welcome to admin

7. Add web form and name it as normal.aspx

**Label1 properties:**

Text=welcome to normal user

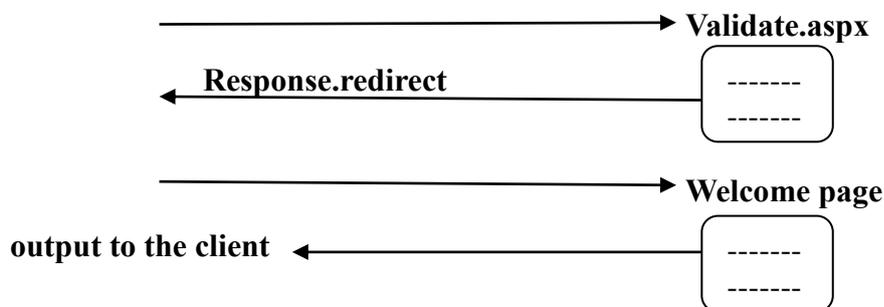
8. Ctrl+f5(login.aspx)

**Response.Redirect method:**

- Response.Redirect method will instruct browser to call a particular page by sending URL
- Response.Redirect method will give less performance over response.transfer method
- Response.Redirect method requires two communications
- Response.Redirect method supports redirection within website and between websites.

**BROWSER**

**WEBSERVER**



**Example: Develop a website to work with dropdown list and Response.Redirect method**

**Procedure:**

9. Open new website==>select ASP.NET empty website and name it as response.
  10. Right click on project path==>select add new item==>select webform and name it as default.aspx
  11. Place one dropdown list
  12. Properties of dropdown list is
    - autopostback: true
    - Items: members
      - 0-gmail
      - 1-google
      - 2-yahoo
- ```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;

```

```
using System.Web.UI.WebControls;
public partial class _Default : System.Web.UI.Page
{
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        switch (DropDownList1.SelectedIndex)
        {
            case 0: Response.Redirect("http://www.gmail.com");
                break;
            case 1: Response.Redirect("http://www.google.com");
                break;
            case 2: Response.Redirect("http://www.yahoo.com");
                break;
        }
    }
}
```

VALIDATION CONTROLS:

- Ensuring proper input from user is called validation.
Example: username cannot be blank, email id requires valid format
- Validation controls are provided with built-in logic to perform different types of validations within ASP.NET page.
- Validation controls validate the user input at runtime.
- Validation controls support client side validation or server side validation(default is client side validation).
- By default validation controls performs their validations automatically on post back in server side.
- We can attach more than one validation control to an input control.
- ASP.NET provides the following validation controls:
 - RequiredFieldValidator
 - RangeValidator
 - CompareValidator
 - RegularExpressionValidator
 - CustomValidator
 - ValidationSummary

BaseValidator Class: The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. The common properties for all the validation controls are:

- **ControlToValidate:** It requires control id to be validated with user input.
- **Display:** Indicates how the error message is shown like static, dynamic and none.

- **EnableClientScript:** This property have two values that is true/false (by default true) .
 - True:** It will perform client side validation and again on the server side when the form is submitted.
 - False:** It will perform server side validation only when the form is submitted.
- **Enabled:** It Enables or disables the validator.
- **ErrorMessage:** Indicates error string.
- **Text:** Error text to be shown if validation fails.
- **SetFocusError:** This property has two values that are true/false (by default false).
 - **True:** cursor will place within the textbox if the user input is invalid.
 - **False:** cursor will place within the next textbox irrespective of user input is valid or invalid.

RequiredFieldValidator Control: The RequiredFieldValidator control ensures that the required field is not empty. In **RequiredFieldValidator Control** the display attribute must be set to dynamic, static or none.

Static (default): Space for the validation message is allocated in the page layout.

Dynamic: Space for the validation message is dynamically added to the page if validation fails.

None: The validation message is never displayed inline.

It contains EnableClientScript attribute which can be set either true or false

- **True:** It will perform client side validation and again on the server side when the form is submitted.
- **False:** It will perform server side validation only when the form is submitted.

RangeValidator Control: The RangeValidator control verifies that the input value within a predetermined range. RangeValidator control will not perform validation if the text box is blank. It has three specific properties:

- **Type:** It defines the type of the data. The available values are: **Currency, Date, Double, Integer, and String.**
- **MinimumValue:** It specifies the minimum value of the range.
- **MaximumValue:** It specifies the maximum value of the range.

Example: Creating a website to work with RangeValidator and RequiredFieldValidator Controls.

Age:	<input type="text"/>	RangeValidator1, RequiredFieldValidator1
Mobile:	<input type="text"/>	RequiredFieldValidator2
<input type="button" value="SUBMIT"/>		

- Create an ASP.NET empty website (name it as validation site).
- Add web form(default.aspx)

RangeValidator properties:

- ControlToValidate-TextBox1
- Type-integer
- MinimumValue-25
- MaimumValue-40
- SetFocusError-True

RequiredFieldValidator1 properties:

- ControlToValidate-TextBox1
- Text-cannot be blank
- SetFocusError-True
- Display-Dynamic

RequiredFieldValidator2 properties:

- ControlToValidate-TextBox2
- Text-cannot be blank
- SetFocusError-True
- Display-Dynamic

Submit Button click logic:

```
Response.Write("<h2>"Data Submitted "</h2>");
```

CompareValidator Control:The CompareValidator control compares a value in one control with a fixed value or a value in another control or with a constant using relational operator.It has the following specific properties:

- **ControlToValidate:** It specifies the control id which acts like source
- **Type:** It specifies the data type to accept input into source.
- **ControlToCompare:** It specifies the control id to be used for comparison with source.
- **ValueToCompare:** It specifies the constant value to be used for comparison with source.
- **Operator:** It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan,LessThanEqual, and DataTypeCheck.

RegularExpressionValidator or RequiredExpressionValidator Control: The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. ASP.NET provides a set of special characters to frame an expression.

- `\d{5}`: It is an expression which accepts 5 digit number.

- `\{1,5}`: It is an expression which accepts 1 to 5 digit number.
- `\d{1,}`: It is an expression which accepts any digit number
- `\D{5}`: It is an expression which accepts 5 characters string
- `\D{1,}`: It is an expression which accepts any number of character strings.
- `^`: It can be used to specify starting characters

Example: `^(91)\d(3)\-\d{8}`
91-040-23456789

- `$`: It can be used to specify ending characters

Example: `^(http://)\D{1,}$(.com/.net)$`
http://www.gmail.com

CustomValidator: It can be used to validate user input with custom logic. Developer can attach custom logic to custom validator control in two ways. The logic which is provided by the developers is called custom logic.

- **ClientValidationFunction Property:** It requires javascript function to perform Client side Validation.
- **ServerValidateEvent Handler:** It requires logic to perform server side validation.

Validation summary: The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation. The following two mutually inclusive properties list out the error message:

- **ShowSummary:** shows the error messages in specified format.
- **ShowMessageBox:** shows the error messages in a separate window.

Example: Create a registration webpage which will perform different validations using validation controls.

username	<input type="text"/>	RequiredFieldValidator1
password	<input type="text"/>	CompareValidator1
confirmpass	<input type="text"/>	
age	<input type="text"/>	RangeValidator1
Emailid	<input type="text"/>	RegularExnressionValidator1

Procedure:

1. Select ASP.NET empty website and name it as registration
2. Add webform(default.aspx)
3. Place 5 lables,5 textboxes and 1 button
4. Place one RequiredFieldValidator,one RangeValidator,one CompareValidator,one RegularExpressionValidator and one ValidationSummary control on default.aspx design view.

Properties:

textBox2:

- text=password

textBox3:

- text=password

RequiredFieldValidator1:

- ControlToValidate=textBox1
- ErrorMessage=Enter something

RangeValidator1:

- ControlToValidate=textBox4
- ErrorMessage=should be greater than 18
- Type=integer

CompareValidator1:

- ControlToCompare=textBox2
- ControlToValidate=textBox3
- ErrorMessage=doesnot match password

RegularExpressionValidator1:

- ControlToValidate=textBox5
- ErrorMessage=invalid email id
- validationExpression=select internet email address

validationSummary:

- HeaderText=you have received the following errors.

UNIT VI

INTRODUCTION TO WEB SERVICES

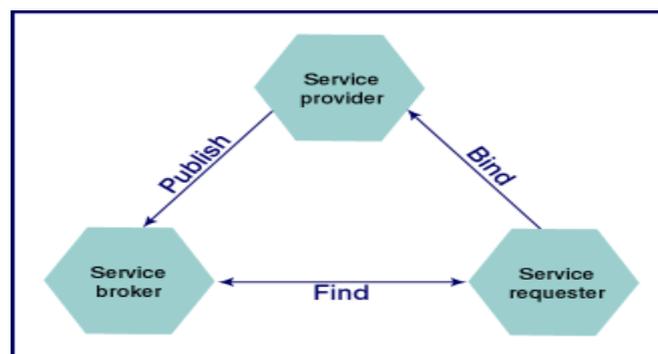
WEB SERVICES:

- Web service is an application logic managed by web server and it provides access to different client applications based on open industry standards in a platform independent way.
- Web services provide communication between different software's across geographical platforms which leads to heterogeneous application communication under heterogeneous platforms.
- Web service standards are open industry standards provided by W3C ,it is designed completely based on XML.
- web services is based on the following things
 1. WSDL (Web Services Description Language)
 2. SOAP (Simple Object Access Protocol)
 3. UDDI (Universal Description ,Discovery and Intigration)
 4. HTTP (Hypertext Transfer Protocol)
- Web services are self-describing and modular business applications that expose the business

logic as services over the Internet through programmable interfaces

- A web service is a web application which is basically a class consisting of methods that could be used by other applications.
- Two separate documents describe web services
 1. A well defined service document describes non-operational service information such as service category, service description and expiration dates.
 2. A Network-Accessible service specification language (NASSL) document describes operational information about the service such as service interface, implementation details and access protocols.

WEB SERVICE ARCHITECTURE:



The web service architecture describes three roles

Service provider: The service provider implements the service and makes it available on the Internet.

Service Requestor: This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

Service Registry(or) Service Broker: This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones

Basic operations:

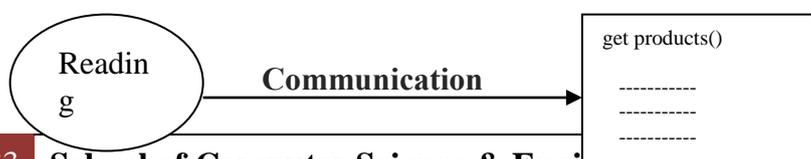
1. Publish
2. Find
3. Bind

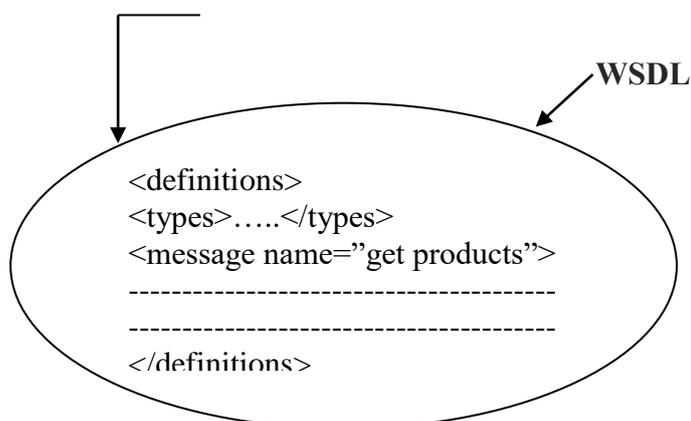
Web Services Description Language (WSDL):

- It is an XML based language to provide description of webservices.

APP (J2EE)

.NET WEBSERVICE





- A WSDL definition document consists of the following four key structural elements
 1. **<definitions>**: It is the root element in a WSDL document.
 2. **<types>**: This element defines data types which are supported by Web service for exchanging the data.
 3. **<message>**: This element contain methods supported by web services.
 4. **<portType>**: This element contains the address of web service
 5. **<binding>** This element contain message encoding format and protocol supported by web services.
- .NET provides one assembly towards web services that is Sustum.Web.Services.dll
- Web method of web services will be given to client application.

CREATING WEB SERVICE USING NOTEPAD:

```

<%@WebService Language="C#" class="mathc"%>
Using System.Web.Services
public class math
{
    [WebMethod]
    public int sum(int a,int b);
    {
        int result;
        result=a+b;
        return result;
    }
}

```

- Save this file as myservice.asmx

Note:

- Web service file extension is .asmx(active service method extension).
- Webservice file can have any number of classes, one class should be declared as webservice using **Class Attribute** of the web service directive.

Example: <%@ WebServiceClass="mathc"%>

- Web service should be placed into virtual directory path.

TESTING WEB SERVICE USING BROWSER:

- Go to browser and write `http://localhost\virdirectorypath\myservice.asmx`
Sum [click]
a-10
b-20
invoke===== <int>30</int>
- The target for webpage is end user.
- The target for webservice is developer.

CREATING WEB SERVICE USING VISUAL STUDIO:

Procedure:

1. Go to visual studio
File====>new====>website
2. In solution explorer Right click on project path and select webservice and click on add.
3. Define web methods

```
[WebMethod]
public int add(int a, int b)
{
    return a+b;
}
[WebMethod]
public int sub(int a, int b)
{
    return a-b;
}
[WebMethod]
public int mul(int a, int b)
{
    return a*b;
}
[WebMethod]
public int div(int a, int b)
{
    return a/b;
}
[WebMethod]
public int moddiv(int a, int b)
{
    return a%b;
}
```

- After writing the logic then go to debug select start without debugging option
- The output window will be displayed in WSDL XML document

Consuming web service:

Select solution explorer of client project

↓
Right click on project path

Add new item==>select web service==> click on add

WIREFORMATS SUPPORTED BY WEBSERVICE:

- Wire format is the concept of arranging data for transmission on network wire.
- Web services supports the following wireformats are
 1. http post
 2. http get
 3. soap
- http post and http get supports only basic types of data arrangement
- soap supports different types(basic, complex and user defined) of data arrangements in a structure manner

MULTITHREADING:

- Multithreading is a concept of executing more than one process simultaneously.
- Multithreading is supported by language level.
- A thread is a unit of execution responsible for in executing a program. By default every application have one thread for execution that is main thread.
- Sharing CPU cycles efficiently is also called as multithreading.
- In a single threaded model, the execution of a program will be one after another that is after completing one task then only it goes to another task.
- In multi-threaded model, all the tasks will be given equal importance, which works on two principles
 1. **Time sharing:** In this case the OS transfers the control between each thread in execution.
 2. **Maximum utilization of CPU time:** It comes into the picture when the first principle violates that is thread could not execute in the time allotted to it without waiting there the control jumps to other threads in execution.
- To work with multi-threading .NET introduces **System.Threading namespace** which contains three important classes.
 1. **Thread:** which is responsible for maintaining thread life cycle.
 2. **Thread start:** which is a predefined delegate and responsible for creating the thread.
 3. **Mutex:** This is required in thread synchronization.

Methods under thread class:

- **Start():** It is used to start the execution of thread.
- **Sleep(int milliseconds):** It is a static method which suspends the execution of a thread until the specified time period was elapsed.
- **Suspend():** suspends the execution of a thread until resume method was called.

- **Resume():** It used for resuming a suspended thread.
- **Abort():** It terminates the execution of a thread.
- **Join():** The join() method is used to hold the execution of currently running thread until the specified thread is dead(finished execution).

Property under thread class:

- **CurrentContext:** Gets the current context in which the thread is executing.
- **CurrentCulture:** Gets or sets the culture for the current thread.
- **CurrentPrinciple :** Gets or sets the thread's current principal (for role-based security).
- **CurrentThread:** Gets the currently running thread.
- **CurrentUICulture:** Gets or sets the current culture used by the Resource Manager to look up culture-specific resources at run-time.
- **ExecutionContext:** Gets an ExecutionContext object that contains information about the various contexts of the current thread.
- **IsAlive:** Gets a value indicating the execution status of the current thread.
- **IsBackground:** Gets or sets a value indicating whether or not a thread is a background thread.
- **IsThreadPoolThread:** Gets a value indicating whether or not a thread belongs to the managed thread pool.
- **ManagedThreadId:** Gets a unique identifier for the current managed thread.
- **Name:** Gets or sets the name of the thread.
- **Priority:** Gets or sets a value indicating the scheduling priority of a thread.
- **ThreadState:** Gets a value containing the states of the current thread.

Steps for creating a thread:

Step1: writing a class with set of methods

```
class name
{
    public static void print()
    {
        // logic
    }
}
```

Step2: create an object for thread start delegate with the address of method.

```
ThreadStart ts=new ThreadStart(name.print);
```

Step3: create an object for thread class with the help of ThreadStart object.

```
Thread t=new Thread(ts);
```

Step4: t.start()

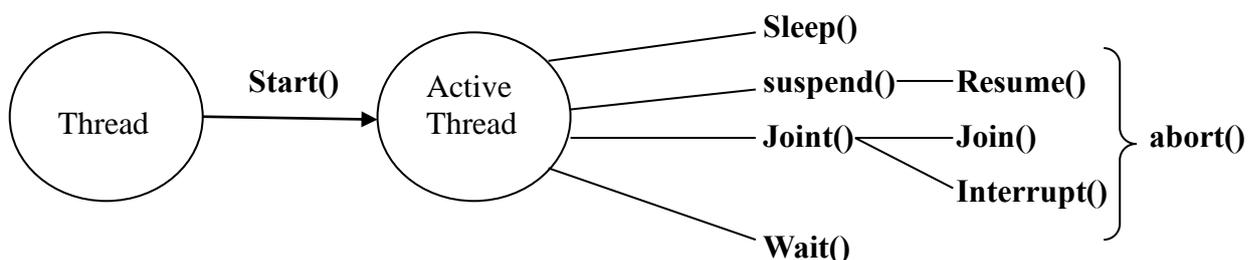
Example: program for creating threads.

```

using System;
using System.Threading;
namespace MultithreadingApplication
{
    class ThreadCreationProgram
    {
        public static void CallToChildThread()
        {
            Console.WriteLine("Child thread starts");
        }
        static void Main(string[] args)
        {
            ThreadStart childref = new ThreadStart(CallToChildThread);
            Console.WriteLine("In Main: Creating the Child thread");
            Thread childThread = new Thread(childref);
            childThread.Start();
            Console.ReadKey();
        }
    }
}

```

Output: In Main: Creating the Child thread
Child thread starts
Managing Threads

THREAD LIFE CYCLE:**Example: a multi-threaded application for executing two methods simultaneously.**

```

using System;
using System.Threading;
namespace ConsoleApplication11
{
    class Program
    {
        public static void m1()
        {
            for (int i = 0; i <= 10; i++)
            {
                Console.WriteLine(i + "");
            }
        }
    }
}

```

```

        Thread.Sleep(500);
    }
}
public static void m2()
{
    for (int k = 11; k <= 20; k++)
    {
        Console.WriteLine(k + "");
        Thread.Sleep(500);
    }
}
}
class test
{
    static void Main(string[] args)
    {
        ThreadStart ts1 = new ThreadStart(Program.m1);
        ThreadStart ts2 = new ThreadStart(Program.m2);
        Thread t1 = new Thread(ts1);
        Thread t2 = new Thread(ts2);
        t1.Start();
        t2.Start();
    }
}
}

```

Example2: program for multithreading

```

using System;
using System.Threading;
namespace ConsoleApplication11
{
    class Program
    {
        Thread t1, t2, t3;
        public Program()
        {
            t1 = new Thread(test1);
            t2 = new Thread(test2);
            t3 = new Thread(test3);
            t1.Start(); t2.Start(); t3.Start();
        }
        public static void test1()
        {
            for (int i = 1; i <= 100; i++)
            {
                Console.WriteLine("test1"+i);
                if (i == 10)
                    Thread.Sleep(20000);
            }
        }
    }
}

```

```

public static void test2()
{
    for (int i = 1; i <= 100; i++)
    {
        Console.WriteLine("test2"+i);
    }
    Console.WriteLine("thread2 exiting");
}
public static void test3()
{
    for (int i = 1; i <= 100; i++)
    {
        Console.WriteLine("test3" + i);
    }
    Console.WriteLine("thread3 exiting");
}
static void Main(string[] args)
{
    Program p = new Program();
    p.t1.Join();
    p.t2.Join();
    p.t3.Join();
    Console.WriteLine("main exiting");
}
}
}
}

```

Thread Locking: When there is multiple thread execution where each thread calling a different method we never have a problem. But in some cases multiple threads may call the same method in that situation we get unexpected results.

Example:

```

using System;
using System.Threading;
namespace ConsoleApplication14
{
    class test
    {
        Thread t1, t2;
        public test()
        {
            t1 = new Thread(display);
            t2 = new Thread(display);
            t1.Start();
            t2.Start();
        }
        public void display()
        {
            Console.Write("[C# is");
            Thread.Sleep(5000);
        }
    }
}

```

```

        Console.WriteLine("object oriented");
    }
    static void Main(string[] args)
    {
        test t = new test();
        t.t1.Join();
        t.t2.Join();
    }
}

```

The previous code gives unexpected results. To overcome this problem we use the thread locking.

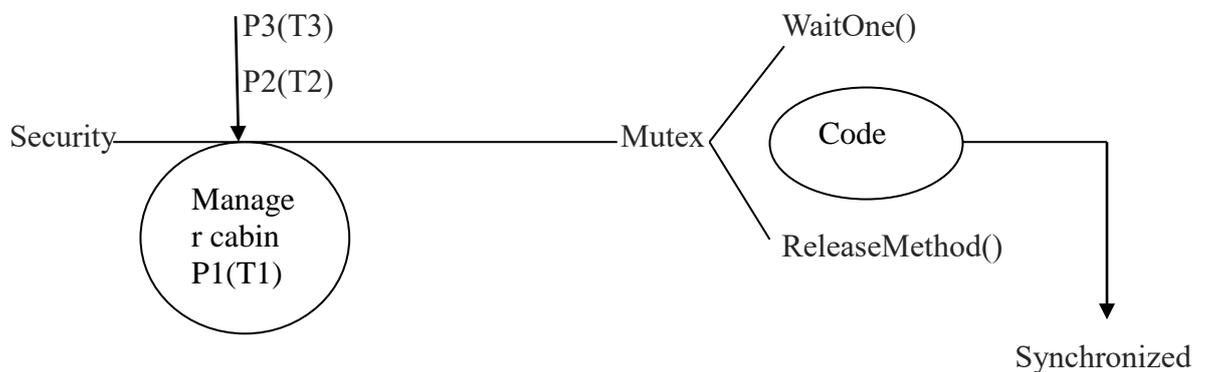
```

public void display()
{
    lock (this)
    {
        Console.WriteLine("[C# is");
        Thread.Sleep(5000);
        Console.WriteLine("object oriented");
    }
}

```

Thread synchronization:

- Whenever more than one thread is trying to access the same process. Then there is a possibility for runtime problem. To overcome this problem we required thread synchronization.
- Thread synchronization is a concept of allowing only one thread into one process at a time.
- In case if process is busy then the thread needs to wait in the queue. Waiting threads will be executed based on the priority.



- To work with thread synchronization we required a predefined class is called Mutex. Mutex class is having two methods.

WaitOne()

ReleaseMethod()

- The code which is written inside of these two methods is called synchronization code. Synchronization code is accessible only one thread at a time.

Example:

```

using System;
using System.Threading;
namespace ConsoleApplication13
{
    class Program
    {
        Mutex m = new Mutex();
        public void m1()
        {
            m.WaitOne();
            for (int i = 1; i <= 10; i++)
            {
                Console.WriteLine(i + "");
                Thread.Sleep(500);
            }
            m.ReleaseMutex();
        }
    }
    class test
    {
        static void Main(string[] args)
        {
            Program p = new Program();
            ThreadStart ts1 = new ThreadStart(p.m1);
            Thread t1 = new Thread(ts1);
            Thread t2 = new Thread(ts1);
            t1.Start(); t2.Start();
        }
    }
}

```

Example1: program for thread synchronization.

- For this first create a table that is create table GTB(accno number,bal number); insert into GTB values(587,10000)
- Open the console application template

```

using System;
using System.Threading;
using System.Data.OleDb;
namespace ConsoleApplication11
{
    class Program
    {
        static Mutex m = new Mutex();
        public static void ATMLogic()
        {
            m.WaitOne();
            OleDbConnection con = new OleDbConnection
                ("provider=MSDAORA;user id=system;password=raja;");
            con.Open();
        }
    }
}

```

```

OleDbCommand cmd=new OleDbCommand
                    ("select bal from GTB where Accno=587",con);
Object obj = cmd.ExecuteScalar();
int bal = int.Parse(obj.ToString());
Thread.Sleep(1000);
int wd = 10000;
Thread.Sleep(1000);
if (bal - wd >= 0)
{
    Thread.Sleep(1000);
    OleDbCommand cmd1 = new OleDbCommand
                    ("update GTB set bal=bal-wd where Accno=587", con);
    Thread.Sleep(1000);
    cmd1.ExecuteNonQuery();
    Thread.Sleep(1000);
    Console.WriteLine("take the money");
    Thread.Sleep(1000);
}
else
{
    Console.WriteLine("insufficient balance");
    Thread.Sleep(1000);
}
m.ReleaseMutex();
}
static void Main()
{
    ThreadStart ts = new ThreadStart(Program.ATMLogic);
    Thread t1 = new Thread(ts);
    Thread t2 = new Thread(ts);
    t1.Start();
    t2.Start();
}
}
}

```

Example: The following example demonstrates the use of the sleep() method for making a thread pause for a specific period of time.

```

using System;
using System.Threading;
namespace MultithreadingApplication
{
    class ThreadCreationProgram
    {
        public static void CallToChildThread()
        {
            Console.WriteLine("Child thread starts");
            int sleepfor = 5000;
            Console.WriteLine("Child Thread Paused for {0} seconds",sleepfor/ 1000);
            Thread.Sleep(sleepfor);
        }
    }
}

```

```

        Console.WriteLine("Child thread resumes");
    }
    static void Main(string[] args)
    {
        ThreadStart childref = new ThreadStart(CallToChildThread);
        Console.WriteLine("In Main: Creating the Child thread");
        Thread childThread = new Thread(childref);
        childThread.Start();
        Console.ReadKey();
    }
}

```

Output: In Main: Creating the Child thread
 Child thread starts
 Child Thread Paused for 5 seconds
 Child thread resumes

ASSEMBLIES:

- An assembly is a unit of file that contains IL code corresponding to your project when it was compiled.
- The name of an assembly is same as project name from which it was created.
- Assembly are known as units of deployment because once the application development is done what we install on client machines is assembly only.
- Assembly contains only reusable logic but not design , the reusable logic will be in the form of byte code(IL).
- Assembly's supports language interoperability that is the dll is developed in C#.NET works in any other .NET programming language.
- An assembly can have an extension of either .exe or .dll
- .exe assemblies are in-process components which are capable of running on their own as well as provides support for others to execute . when we use project templates like windows forms, WCF, console applications and windows services they will generate .exe assembly.
- .dll assemblies are out-process components which are not capable of running on their own they can provide support for others to execute. project templates like class and control library will generate an .dll assembly.
- To develop an assembly .NET introduced class library project template.
- Assemblies are not executed directly.
- To add an assembly into applications
 Right click on project path==>choose add reference==>select browse==>choose dll file
- Assemblies are divided into two types

Private assembly: By default every assembly is private ,if references of these assemblies was added to any project, a copy of the assembly is created and given to the project.so each project maintains a private copy of the assembly

Advantage: These are faster in execution.

Disadvantage: These are used by only one application

Example:

Procedure:

- Open a new class library template(name as raja)
 using System;
 namespace raja
 {

```

public class pa1
{
    public string m1()
    {
        return "pa1 test";
    }
}
namespace sekhar
{
    public class pa2
    {
        public string m2()
        {
            return "pa2 test";
        }
    }
}

```

- Now build the project, it will generate .dll file in their project folder.

CONSUMING THE PRIVATE ASSEMBLY:

- **Open winform app project(name as privateassembly)**

- **Place abutton**

- **To Add assembly to private assembly**

Right click on Project path ==>add new item==>add reference==>browse==>select raja.dll file

- Then write the code under button click event

using raja;

using raja.sekhar;

namespace WindowsFormsApplication32

```

{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            pa1 p1 = new pa1();
            MessageBox.Show(p1.m1());
            pa2 p2 = new pa2();
            MessageBox.Show(p2.m2());
        }
    }
}

```

Shared assembly:

- An assembly is registered with GAC(global assembly cache) is called as shared assembly or public assembly.
- If an assembly was shared multiple copies will not be created even if being consumed by

multiple projects , only a single copy under GAC server for all the projects.

- The location of GAC folder is c:\windows\microsoft.net\assembly\GAC_MSIL.
- To register a dll with GAC then the dll must contain a strong name.
- An assembly which has three attributes like name, version and public key token were known as strong named assemblies.
 1. **Name:** It was the name of an assembly used for identification. Every assembly by default has name.
 2. **Version:** To specify the version.
 3. **Public key token:** GAC contains multiple assemblies in it, to identify each assembly it will maintain a key value for the assembly known as public key token, which should be generated by us and associated with an assembly to make it strong named.
- **Strong name provides unique identifier for assembly.**
- To create a strong name go to visual studio command prompt and set the path upto debug folder.

sn -k key.snk

Where sn => strong name

key.snk=>contains the public key token after execution of this command.

STEPS TO LINK STRONG NAME TO AN ASSEMBLY:

- To link key.snk to raja
Right click on Project path ==> properties==>signing==>check checkbox==>select **browse** from combo box==>choose key.snk.
- After project is build then to register an assembly into GAC using
gacutil -I raja.dll

Example:

Procedure:

- Open class library template(name as raja)

```
using System;
namespace raja
{
    public class r
    {
        public string a()
        {
            return "this is shared assembly";
        }
    }
}
```

- To create a strong name go to visual studio command prompt and set the path upto debug folder(raja project path).
sn -k key.snk
- To link key.snk to raja
Right click on Project path ==> properties==> signing==>click checkbox==>select browse from combobox==>choose key.snk.
- Now build the project
build==>build raja
- Register raja.dll with GAC(global assembly cache)(the location of GAC folder is c:\windows\microsoft.net\assembly\GAC_MSIL)
gacutil -i raja.dll

CONSUMING SHARED ASSEMBLY:

- **Open winform project**
- **Place a button**
- **Right click on Project path ==>add reference==>browse==>raja.dll**

```
using raja;
namespace WindowsFormsApplication33
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            r b = new r();
            MessageBox.Show(b.a());
        }
    }
}
```

- An assembly internally contains three major things

Manifest Info: It contains information about the attributes that are associated with an assembly like title, description, company, version etc.

Type Metadata: It contains information about all the types under assembly like namespaces, classes, structures, interfaces and it describes about the contents of an assembly, so an assembly can be called as self-describing unit of execution because it describes about itself to the execution environment with the help of its metadata.

IL Code: These are the language instructions we have defined which can be understood by CLR

Shared(or)Public Assembly	Private Assembly
Public assembly can be used by multiple applications.	Private assembly can be used by only one application.
Public assembly is stored in GAC (Global Assembly Cache).	Private assembly will be stored in the specific application's directory or sub-directory.
Public assembly is also termed as shared assembly.	There is no other name for private assembly.
Strong name has to be created for public assembly.	Strong name is not required for private assembly.
Public assembly should strictly enforce version constraint.	Private assembly doesn't have any version constraint.
An example to public assembly is the actuate report classes which can be imported in the library and used by any application that prefers to implement actuate reports.	By default, all assemblies you create are examples of private assembly. Only when you associate a strong name to it and store it in GAC, it becomes a public assembly.

VERSIONING:

- Every assembly has a version number and versions cannot transcend the boundary of the assembly.
- A version can refer only to the contents of a single assembly.
- All types and resources within the assembly can change versions together.
- Every software has version to it, for discriminating changes that has been made from time to time.
- Versioning is used to create the assemblies with same name.
- A version contains four integer parts

Syntax: [major version.minor version.revised version.new version]

Example:10.23.9.8

STEPS TO CREATE VERSIONS OF AN ASSEMBLY:

- Project explorer==>properties==>assemblyinfo.cs==>change the assembly version and save project.
- To create strong name for an assembly
- To link the .snk file to project.
- Register the assembly into GAC
- Open raja.dll project which we have created previously and change the code under abc() method
 return "from raja";
- Now open assemblyinfo.cs file and change the assembly version attribute to 1.0.1.1
- Rebuild the project and add the new version of raja assembly also into GAC using gacutil tool.