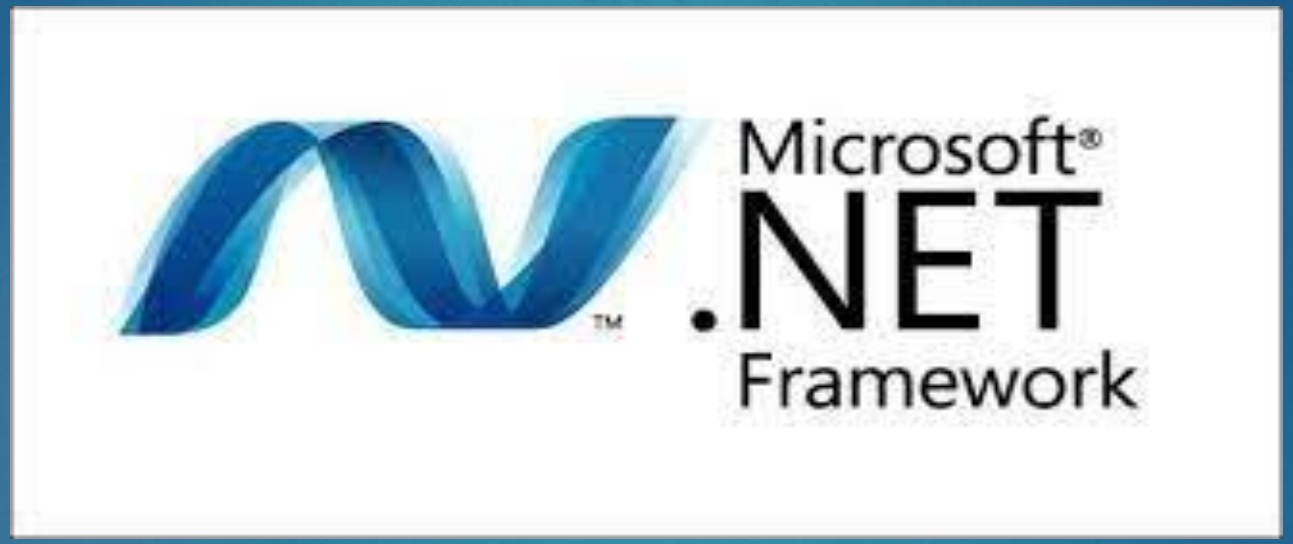
C# and Dot Net Framework

Part-1



Dr. K ADISESHA

C# and Dot Net Framework

- Introduction
- HTML
- HTML Tags
- Web Script
- .NET Framework

Prof. K. Adisesha

Introduction

Web Technology:

Web Technology refers to the various tools and techniques that are utilized in the process of communication between different types of devices over the internet.

- A web browser is used to access web pages. Web browsers can be defined as programs that display text, data, pictures, animation, and video on the Internet.
- Web Technology can be classified into the following sections:
  - ***World Wide Web (WWW)***
  - ***Web Browser***
  - ***Web Server***
  - ***Web Pages***
  - ***Web Development***

Introduction

Web Technology:

Web Technology refers to the various tools and techniques that are utilized in the process of communication between different types of devices over the internet.

- ***World Wide Web (WWW):*** *The World Wide Web is based on several different technologies : Web browsers, Hypertext Markup Language (HTML) and Hypertext Transfer Protocol (HTTP).*
- ***Web Browser:*** *The web browser is an application software to explore www (World Wide Web). It provides an interface between the server and the client and requests to the server for web documents and services.*
- ***Web Server:*** *Web server is a program which processes the network requests of the users and serves them with files that create web pages. This exchange takes place using Hypertext Transfer Protocol (HTTP).*

Introduction

Web Technology:

Web Technology refers to the various tools and techniques that are utilized in the process of communication between different types of devices over the internet.

- ***Web Pages:*** *A webpage is a digital document that is linked to the World Wide Web and viewable by anyone connected to the internet has a web browser.*

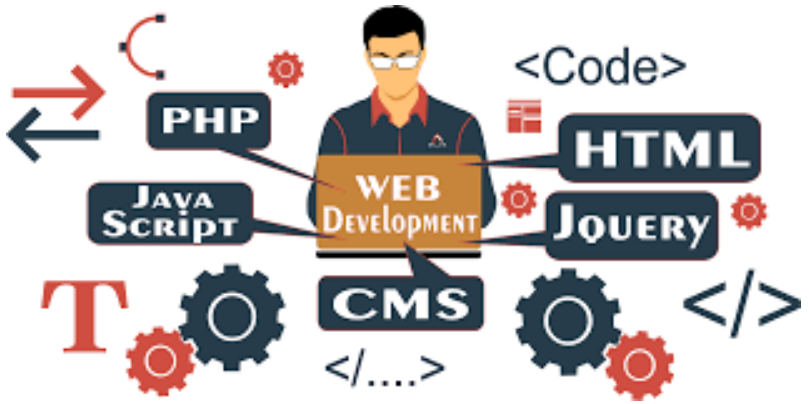
- ***Web Development:*** *Web development refers to the building, creating, and maintaining of websites.*
  - *It includes aspects such as web design, web publishing, web programming, and database management.*
  - *It is the creation of an application that works over the internet i.e. websites.*
  - *Web designing is of three kinds, to be specific static, dynamic and eCommerce.*

Introduction

Web Technology:

Web Development can be classified into two ways:

- ***Frontend Development:*** *The part of a website that the user interacts directly is termed as front end. It is also referred to as the 'client side' of the application.*
- ***Backend Development:*** *Backend is the server side of a website. It is the part of the website that users cannot see and interact. It is the portion of software that does not come in direct contact with the users. It is used to store and arrange data.*



HTML

HTML :

HTML stands for "Hyper Text Markup Language".

- HTML developed by Tim Berners-Lee in 1990, which was officially published in 1995 as HTML 2.0, The latest version of HTML is HTML5.
- HTML is a simple markup language used to create or develop static web pages. A web page is a document that provides specific information to the user via a web browser (eg: Chrome, IE, etc.,).
- Web pages can either be static or dynamic. Static pages display the same content every time we visit. These pages are written in HTML.
- In Dynamic Every time a page is accessed, its content is altered. Typically, these pages are created in scripting languages like PHP, ASP, JSP, etc,.

Introduction

HTML :

Features of HTML

- HTML is used to create or develop static web documents or web pages.
- HTML documents are plain text files, these are created by using text editor like notepad.
- HTML is HyperText language because it supports font styled text, pictures, audio, video, graphics and animations.
- HTML is Markup language, which provides a set of tags suitable for making up webpages.
- HTML is not a case-sensitive and it does not generate errors.
- HTML is a tag-based system. A tag is a special instruction for browser. A tag is made up of left operator(<), right operator(>) and a tag name between these two operators.

HTML

HTML :

Structure of HTML Document

- All HTML documents does follow some basic structure. It has two blocks
- The Head block contains control information and title of document.
- The Body block contains content that displayed on screen as well as tags which control how that content is formatted by browser.
- The Basic HTML document is:
- format :

```
<html>
 <head>
  <title>A HTML Document</title>
 </head>
 <body>
    Welcome to HTML
 </body >
</html>
```

HTML

Basic HTML Tags :

TAG	DESCRIPTION
<html>	Describes an HTML document.
<head>	Represents head section of the HTML document.
<title>	Describes the title of HTML document.
<body>	Describes content of the document.
<!--...-->	This tag is used to insert comment inside document code.
<H1> ……<H6>	Describes the heading of HTML document.
<P>	Describes the Paragraph of HTML document.
<table>	To define a Table in HTML.

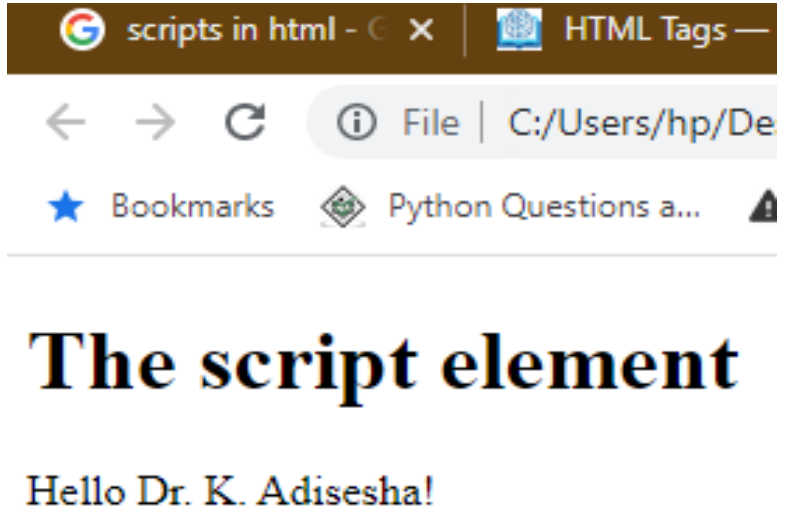
Script

HTML Script:

The HTML <script> tag is used to define a client-side script (JavaScript).

- The <script> element either contains script statements, or it points to an external script file through the src attribute.
- Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

```
<html>
<body>
<h1>The script element</h1>
<p id="demo"></p>
<script>
document.getElementById("demo").innerHTML = "Hello Dr. K. Adisesha!";
</script>
</body>
</html>
```



Script

Client-side and Server-side Scripts:

On a dynamic website there are client -side and server -side scripts. Client-side and server-side are sometimes referred to as front-end and back-end.

- The client-side of a website refers to the web browser and the server-side is where the data and source code is stored.
- The client-side programming languages are HTML, CSS, and JavaScript.
- The server-side scripting programming languages, including PHP, ColdFusion, Python, ASP.net, Java, C++, Ruby, C#
- A server-side script communicates with the server when it is executed.

Script

Client-side and Server-side Scripts:

On a dynamic website there are client -side and server -side scripts. Client-side and server-side are sometimes referred to as front-end and back-end.

- The client-side of a website refers to the web browser and the server-side is where the data and source code is stored.
- ***When to use Client-side and Server-side Scripting.***
- A site such as Google, Amazon, and Facebook will use both types of scripting:
  - Server-side handles logging in, personal information and preferences and provides the specific data which the user wants (and allows new data to be stored).
  - Client-side makes the page interactive, sorting data in different ways if the user asks for that by clicking on elements with event triggers.

Script

Client-side scripts:

Client-side means that the processing takes place on the user's computer.

- *It requires browsers to run the scripts on the client machine without involving any processing on the server.*
- ***Advantages of Client-side Scrips:***

- *Allow for more interactivity by immediately responding to users' actions.*
- *Execute quickly because they do not require a trip to the server.*
- *May improve the usability of Web sites for users whose browsers support scripts.*
- *Can give developers more control over the look and behaviour of their Web widgets.*
- *Can be substituted with alternatives (for example, HTML) if users' browsers do not support scripts*
- *Are reusable and obtainable from many free resources.*

Script

Advantages and disadvantages of client-side scripts:

- ***Disadvantages of Client-side Scripts:***

- *Not all browsers support scripts, therefore, users might experience errors if no alternatives have been provided.*
- *Different browsers and browser versions support scripts differently, thus more quality assurance testing is required.*
- *More development time and effort might be required (if the scripts are not already available through other resources).*
- *Developers have more control over the look and behaviour of their Web widgets; however, usability problems can arise if a Web widget looks like a standard control but behaves differently or vice-versa.*

Script

Server-side scripts:

Server-side means that the processing takes place on a web server.

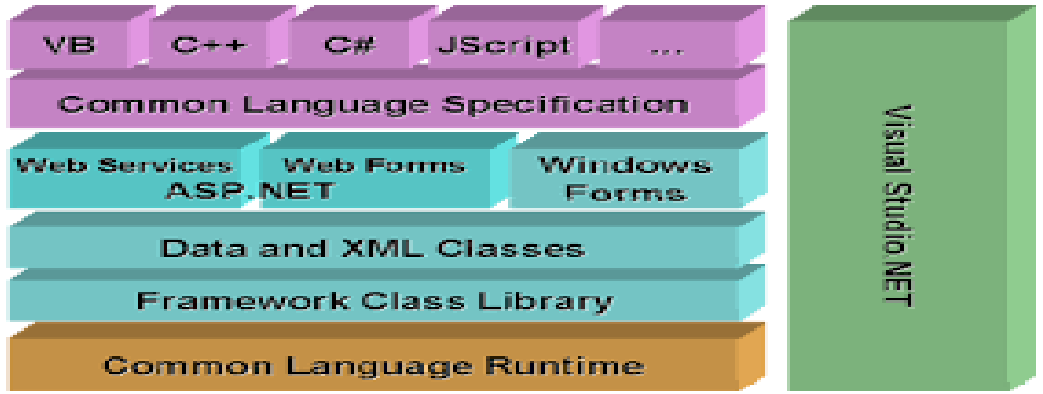
- ***Advantages of Server-side Script:***
  - *User can create one template for the entire website*
  - *The site can use a content management system which makes editing simpler.*
  - *Generally quicker to load than client-side scripting*
  - *User is able to include external files to save coding.*
  - *Scripts are hidden from view so it is more secure. Users only see the HTML output.*
- ***Disadvantages of Server-side Script:***
  - *Many scripts and content management systems tools require databases in order to store dynamic data.*
  - *It requires the scripting software to be installed on the server.*
  - *The nature of dynamic scripts creates new security concerns, in some cases making it easier for hackers to gain access to servers exploiting code flaws.*

.NET framework

.NET framework :

.NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000.

- It is used to develop applications for web, Windows, phone. Moreover, it provides a broad range of functionalities and support.
- This framework provides various services like memory management, networking, security, memory management, and type-safety.



.NET framework

.NET framework :

The .NET Framework is composed of four main components:

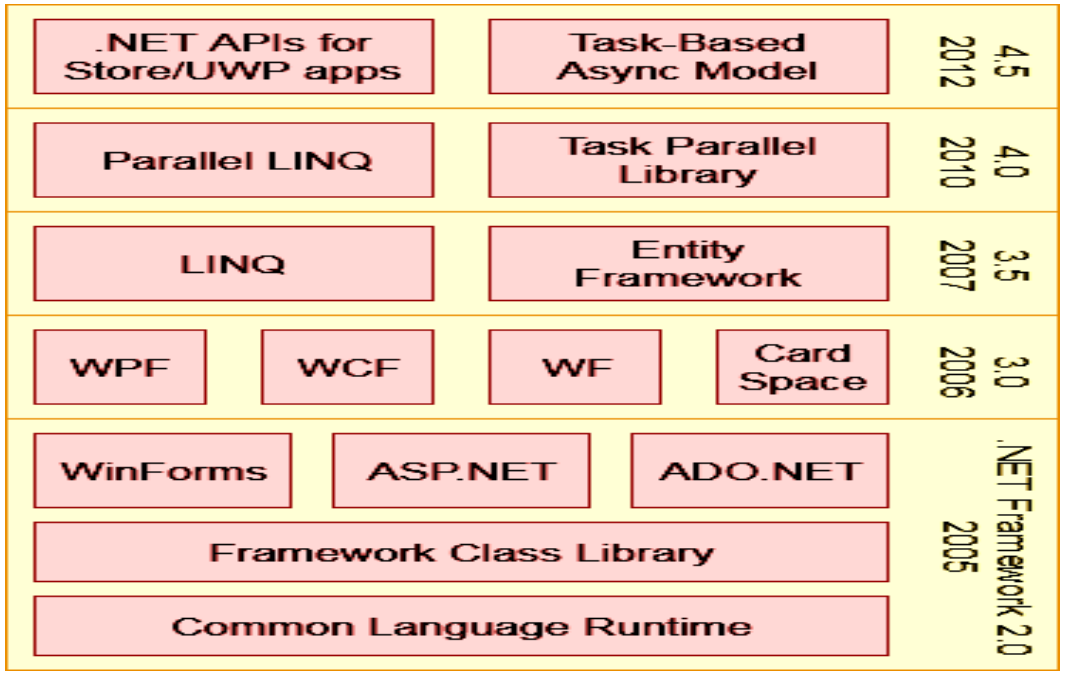
- *Common Language Runtime (CLR)*
- *Framework Class Library (FCL),*
- *Core Languages (WinForms, ASP.NET, and ADO.NET)*
- *Other Modules (WCF, WPF, WF, Card Space, LINQ, Entity Framework, Parallel LINQ, Task Parallel Library, etc.)*

.NET framework

.NET framework :

.NET is a framework to develop software applications. It is designed and developed by Microsoft and the first beta version released in 2000.

- Various components that are present in .NET Framework based on version are:



.NET framework

.NET framework :

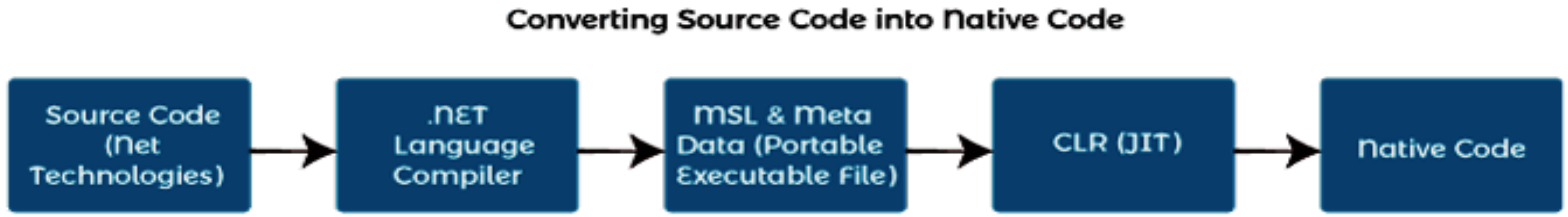
The .NET Framework is composed of four main components:

- ***Common Language Runtime (CLR)***

❖ *It is a program execution engine that loads and executes the program. It converts the program into native code.*

❖ *It acts as an interface between the framework and operating system. It does exception handling, memory management, and garbage collection. Moreover, it provides security, type-safety, interoperability, and portability.*

❖ *A list of CLR components are given below:*

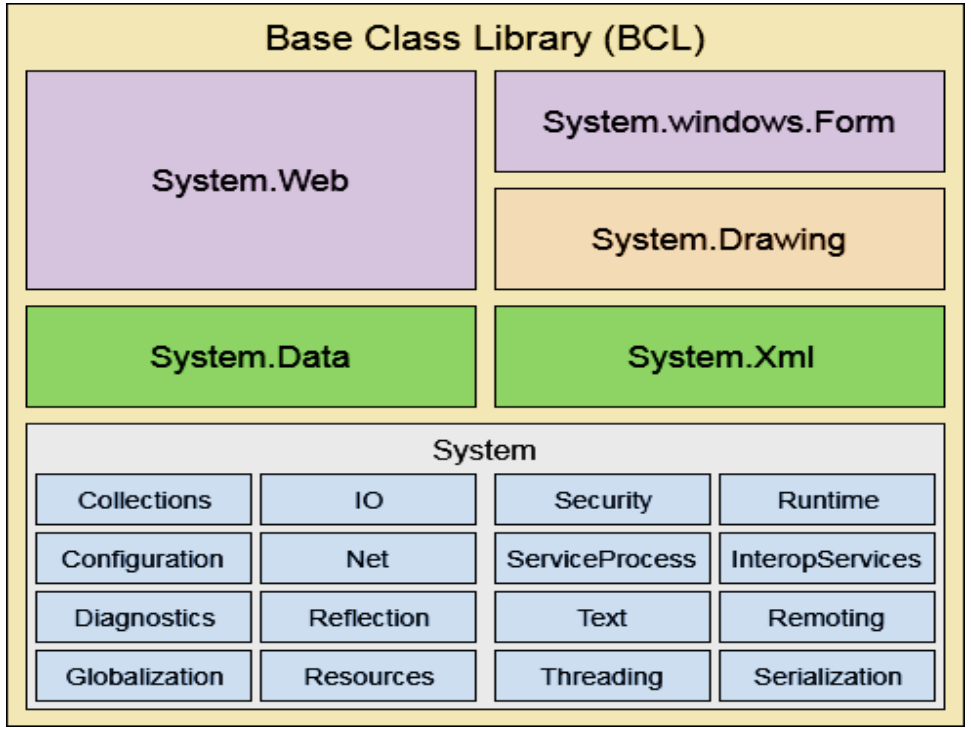


.NET framework

.NET framework :

The .NET Framework is composed of four main components:

- ***Framework Class Library (FCL)***
  - *It is a standard library that is a collection of thousands of classes and used to build an application.*
  - *The BCL (Base Class Library) is the core of the FCL and provides basic functionalities.*



.NET framework

.NET framework :

The .NET Framework is composed of four main components:

- Core Languages (WinForms, ASP.NET, and ADO.NET)
- **WinForms:** *Windows Forms is a smart client technology for the .NET Framework, a set of managed libraries that simplify common application tasks such as reading and writing to the file system.*
- **ASP.NET:** *ASP.NET is a web framework designed and developed by Microsoft. It is used to develop websites, web applications, and web services. It provides a fantastic integration of HTML, CSS, and JavaScript. It was first released in January 2002.*
- **ADO.NET:** *ADO.NET is a module of .Net Framework, which is used to establish a connection between application and data sources. Data sources can be such as SQL Server and XML. ADO .NET consists of classes that can be used to connect, retrieve, insert, and delete data.*

.NET framework

.NET framework :

Discussion

Queries ?

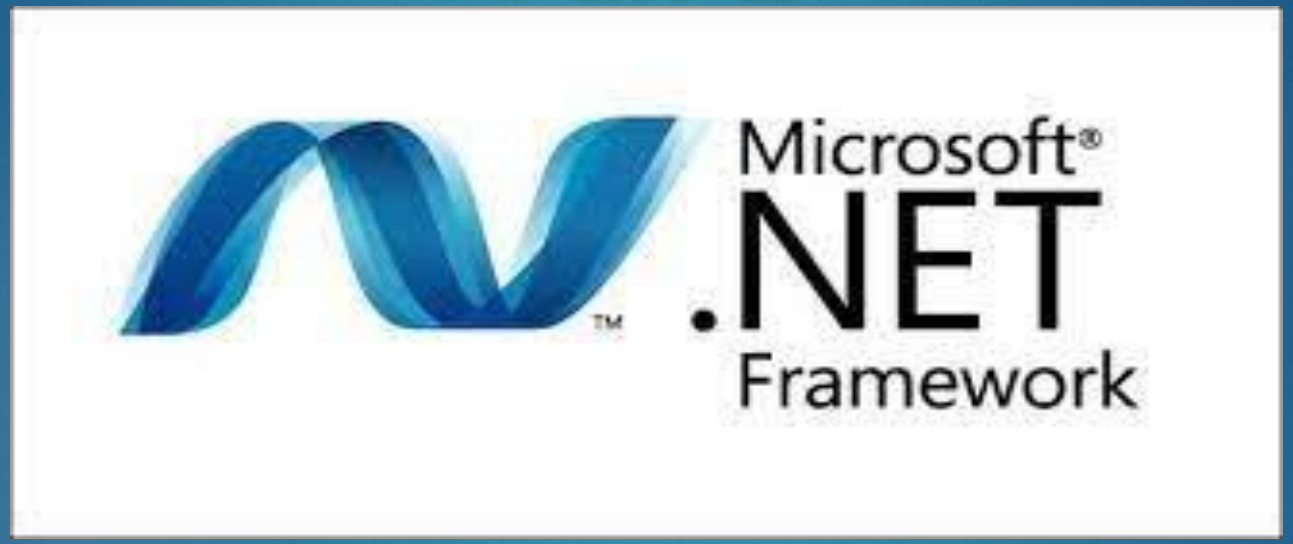
Prof. K. Adisesha

9449081542

Thank you

C# and Dot Net Framework

Part-2



Dr. K ADISESHA

Introduction to C#

- Introduction
- Overview of C#
- C# Program Structure
- Arrays
- OOPS with C#

Prof. K. Adisesha

Introduction

C# (C-Sharp) :

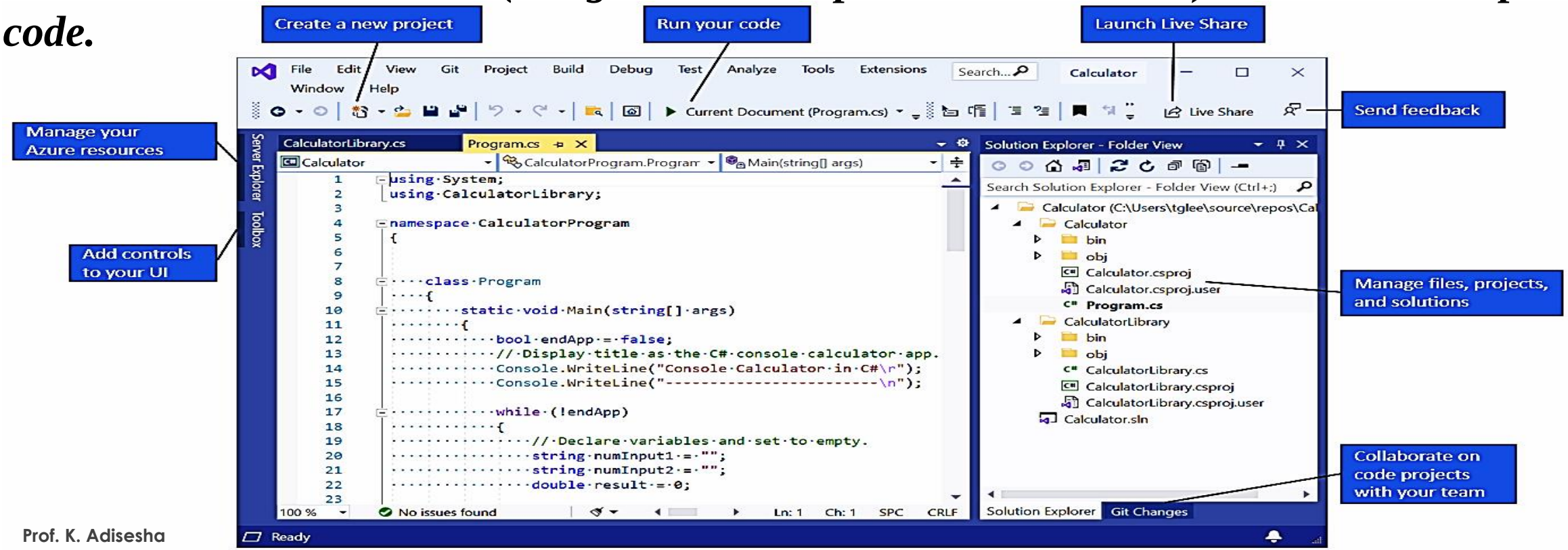
C# (C-Sharp) is object-oriented programming language created by Microsoft that runs on the .NET Framework.

- C# has roots from the C family, and the language is close to other popular languages like C++ and Java.
- C# is used for developing:
  - ***Mobile applications***
  - ***Desktop applications***
  - ***Web applications***
  - ***Web services***
  - ***Games***
  - ***Database applications***

Introduction

C# IDE:

C# uses Visual Studio IDE (Integrated Development Environment) to edit and compile code.



Introduction

C# IDE:

Using Visual Studio.Net IDE 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.

Introduction

C# program structure:

A C# program consists of the following parts:

- *Namespace declaration*
- *A class*
- *Class methods*
- *Class attributes*
- *A Main method*
- *Statements and Expressions*
- *Comments*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MyFirstProject
{
    internal class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Welcome to C#.NET");
            Console.ReadKey();
        }

    }

}
```

Importing Namespaces Section

Namespace Declaration Section

Class Declaration Section

Main Method Section

C# program

C# program Structure:

- ***C# Comments:*** *Comments can be used to explain C# code, and to make it more readable. It can also be used to prevent execution when testing alternative code.*
  - *Single-line comments start with two forward slashes (//).*
  - *Multi-line comments start with /* and ends with */. (will not be executed).*
- ***C# Identifiers:*** *Identifiers are descriptive names in order to create understandable and maintainable code.*
- ***The general rules for naming Identifiers/variables are:***
  - *Names can contain letters, digits and the underscore character (_)*
  - *Names must begin with a letter*
  - *Names should start with a lowercase letter and it cannot contain whitespace*
  - *Names are case sensitive ("myVar" and "myvar" are different variables)*
  - *Reserved words (like C# keywords, such as int or double) cannot be used as names*

C# program

C# program Structure:

- ***C# Variables:*** *Variables are containers for storing data values. All C# variables must be identified with unique names.*
  - ***Syntax: type variableName = value;***
  - ***Example: string name = "K. Adisesha";***
    int x = 5, y = 6, z = 50;
- ***C# Constants:*** *The* ***const*** *keyword is useful when you want a variable to always store the same value.* ***Example: const int Num = 15;***
  Num = 20; // error
- ***Display Variables:*** *The WriteLine() method is often used to display variable values to the console window. To combine both text and a variable, use the + character:*

Example: string name = "John";
Console.WriteLine("Hello " + name);

C# program

C# program Structure:

***C# Data Types:* A data type specifies the size and type of variable values. It is important to use the correct data type for the corresponding variable to save time and memory.**

➢ *The most common data types are:*

Data Type	*Size*	*Description*
Int	**4 bytes**	***Stores whole numbers from -2,147,483,648 to 2,147,483,647***
long	**8 bytes**	***Stores whole numbers from -/+9,223,372,036,854,775,808***
float	**4 bytes**	***Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits***
double	**8 bytes**	**Stores fractional numbers. Sufficient for storing 15 decimal digits**
bool	**1 bytes**	**Stores true or false values**
char	**2 bytes**	**Stores a single character/letter, surrounded by single quotes**
string	***2 bytes per character***	***Stores a sequence of characters, surrounded by double quotes***

C# Programming

C# program Structure:

C# Type Casting: ***Type casting is when you assign a value of one data type to another type.***

- *In C#, there are two types of casting:*
- ***Implicit Casting (automatically)*** *- converting a smaller type to a larger type size*
  - ***char -> int -> long -> float -> double***
  - ***Example:*** ***int myInt = 9;***
    double myDouble = myInt; *// Automatic casting: int to double*
- ***Explicit Casting (manually)*** *- converting a larger type to a smaller size type*
- *It must be done manually by placing the type in parentheses in front of the value:*
  - ***double -> float -> long -> int -> char***
  - ***Example:*** ***double myDouble = 9.78;***
    int myInt = (int) myDouble; ***// Manual casting: double to int***

C# Programming

C# program Structure:

Type Conversion Methods: ***It is also possible to convert data types explicitly by using built-in methods,***

- *In C#, there are various built-in methods for types conversion:*
  - ***Convert.ToBoolean***
  - ***Convert.ToDouble***
  - ***Convert.ToString***
  - ***Convert.ToInt32 (int)***
  - ***Convert.ToInt64 (long)***

Example:

```
int myInt = 10;
double myDouble = 5.25;
bool myBool = true;
Console.WriteLine(Convert.ToString(myInt));    // convert int to string
Console.WriteLine(Convert.ToDouble(myInt));    // convert int to double
Console.WriteLine(Convert.ToInt32(myDouble));  // convert double to int
Console.WriteLine(Convert.ToString(myBool));   // convert bool to string
```

C# Programming

C# program Structure:

C# User Input/output: The user can input or display output his or hers data, which is stored in the variable using built-in methods.

User Input:*The Console.ReadLine() method returns a string.*

- *// Create a string variable and get user input from the keyboard and store it in the variable*

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

User Output:*Console.WriteLine() is used to output (print) values..*

- *// Print the value of the variable (userName), which will display the input value*

```
Console.WriteLine("Username is: " + userName);
```

- ***Example:***

```
Console.WriteLine("Enter your age:");
int age = Convert.ToInt32(Console.ReadLine());
Console.WriteLine("Your age is: " + age);
```

C# Programming

C# program Structure:

***C# - Operators:* An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.**

- *C# has rich set of built-in operators and provides the following type of operators.*
  - *Arithmetic Operators*
  - *Relational Operators*
  - *Logical Operators*
  - *Bitwise Operators*
  - *Assignment Operators*
  - *Misc Operators*

C# Programming

C# - Operators:

***Arithmetic Operators:* Following table shows all the arithmetic operators supported by C#.**

➢ *Assume variable A holds 10 and variable B holds 20 then −.*

Operator	Description	Example
+	Adds two operands	A + B = 30
-	Subtracts second operand from the first	A - B = -10
*	Multiplies both operands	A * B = 200
/	Divides numerator by de-numerator	B / A = 2
%	Modulus Operator and remainder of after an integer division	B % A = 0
++	Increment operator increases integer value by one	A++ = 11
--	Decrement operator decreases integer value by one	A-- = 9

C# Programming

C# - Operators:

***Relational Operators:* Following table shows all the relational operators supported by C#.**

- *Assume variable A holds 10 and variable B holds 20 then −.*

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

C# Programming

C# - Operators:

***Relational Operators:* Following table shows all the relational operators supported by C#.**

➢ *Assume variable A holds 10 and variable B holds 20 then −.*

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

C# Programming

C# - Operators:

***Logical Operators:* Following table shows all the logical operators supported by C#.**

➢ *Assume variable A holds Boolean value true and variable B holds value false, then:*

Operator	Description	Example
&&	*Called Logical AND operator. If both the operands are non zero then condition becomes true.*	**(A && B) is false.**
||	*Called Logical OR Operator. If any of the two operands is non zero then condition becomes true.*	**(A || B) is true.**
!	*Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.*	**!(A && B) is true.**

***Bitwise Operators:* Bitwise operator works on bits and perform bit by bit operation.**

➢ *The Bitwise operator supported by C# are: " &, \|, ~ , ^ , >> , << "*

C# Programming

C# - Operators:

Miscellaneous Operators: ***There are few other important operators including sizeof, typeof and ? : supported by C#.***

Operator	Description	Example
sizeof()	*Returns the size of a data type.*	*sizeof(int), returns 4.*
typeof()	*Returns the type of a class.*	*typeof(StreamReader);*
&	*Returns the address of an variable.*	*&a; returns actual address of the variable.*
*****	*Pointer to a variable.*	**a; creates pointer named 'a' to a variable.*
? :	*Conditional Expression*	*If Condition is true ? Then value X : Otherwise value Y*
is	*Determines whether an object is of a certain type.*	*If(Ford is Car) // checks if Ford is an object of the Car class.*
as	*Cast without raising an exception if the cast fails.*	*Object obj = new StringReader("Hello");StringReader r = obj as StringReader;*

C# Programming

C# - Control Structures:

Conditional statements: ***There are few conditions statements to perform different actions for different decisions that are supported by C#.***

- *C# has the following conditional statements:*
  - ***if*** *:Use if to specify a block of code to be executed, if a specified condition is true*
  - ***else*** *: Use else to specify a block of code to be executed, if the same condition is false*
  - ***else if*** *:Use else if to specify a new condition to test, if the first condition is false*
  - ***Switch*** *: Use switch to specify many alternative blocks of code to be executed*

Example

```
if (30 > 18)
{
  Console.WriteLine("30 is greater than 18");
}
```

C# Programming

C# - Control Structures:

Short Hand If...Else (Ternary Operator): ***short-hand if else, which is known as the ternary operator because it consists of three operands.***

- *It can be used to replace multiple lines of code with a single line.*
- *It is often used to replace simple if else statements:*

Syntax: ***variable = (condition) ? expressionTrue : expressionFalse;***

Example

```
int Percentage = 60;
if (time < 35)
{   Console.WriteLine("First class"); }
else
{   Console.WriteLine("Fail."); }
```

Example :Ternary Operator

```
int Percentage = 60;
string result = (Percentage < 35) ? "First Class." : "Fail";
Console.WriteLine(result);
```

C# Programming

C# - Control Structures:

***Switch Statements :* Use the switch statement to select one of many code blocks to be executed.**

- *The switch expression is evaluated once*
- *The value of the expression is compared with the values of each case*
- *If there is a match, the associated block of code is executed:*

Example

```
switch(expression)
{
  case x:
    // code block
    break;
case y:
    // code block
    break;
 default:
    // code block
    break;     }
```

C# Programming

C# - Control Structures:

***C# - Loops:* Loops can execute a block of code as long as a specified condition is reached.**

- *C# provides following types of loop to handle looping requirements.*

Loop Type	Description
while loop	***It repeats a statement or a group of statements while a given condition is true. It tests the condition before executing the loop body.***
for loop	***It executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.***
do...while loop	***It is similar to a while statement, except that it tests the condition at the end of the loop body***
nested loops	***You can use one or more loop inside any another while, for or do..while loop.***

C# Programming

C# - Loops:

- ***C# For Loop:*** ***When you know exactly how many times you want to loop through a block of code, use the for loop instead of a while loop.***
  - *Syntax : for (statement 1; statement 2; statement 3)*
    { // code block to be executed }

Example

```
for (int i = 0; i < 5; i++)
{   Console.WriteLine(i);
}
```

- ***foreach Loop:*** ***There is also a foreach loop, which is used exclusively to loop through elements in an array:***
  - *Syntax : foreach (type variableName in arrayName)*
    {
    // code block to be executed
    }

Example

```
string[] Course = {"BCA", "BCom",
"BBA", "BHM"};
foreach (string i in Course)
{   Console.WriteLine(i);
}
```

C# Programming

C# - Arrays:

An array stores a fixed-size sequential collection of elements of the same type.

- *To declare an array in C#, you can use the following syntax −*
  - ***Syntax :*** *datatype[] arrayName;*
- *Array is a reference type, so you need to use the new keyword to create an instance of the array.*
  - ***Example :*** *double[] balance = new double[10];*
- *You can also create and initialize an array, as shown −*
  - ***Example :*** *int [] marks = new int[5] { 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 −*
  - ***Example:*** *int [] marks = new int[] { 99, 98, 92, 97, 95};*
    int[] score = marks;

C# Programming

Multidimensional Arrays:

A multidimensional array is basically an array of arrays which can have any number of dimensions. The most common are two-dimensional arrays (2D).

- *The single comma [,] specifies that the array is two-dimensional. A three-dimensional array would have two commas: int[,,].*
- *To create a 2D array, add each array within its own set of curly braces, and insert a comma (,) inside the square brackets.*
  - ***Syntax :*** *int[,] numbers = { {1, 2, 3}, {10, 20, 30} };*
- *To access an element of a two-dimensional array, you must specify two indexes.*
  - ***Example :*** *int[,] numbers = { {1, 2, 3}, {10, 20, 30} };*

```
Console.WriteLine(numbers[0, 2]);  // Outputs 3
```

C# Programming

C# Methods / Functions:

A method is a block of code which only runs when it is called.You can pass data, known as parameters, into a method.

- *Methods are used to perform certain actions, and they are also known as functions.*
- *A method is defined with the name of the method, followed by parentheses ().*
- *C# provides some pre-defined methods, such as Main().*
  - ***Create a method inside the Program class:***

```
class Program
{   static void MyMethod()
  {     // code to be executed     }
}
```

```
static void MyMethod()
{   Console.WriteLine("Hello Sunny!");
}
static void Main(string[] args)
{   MyMethod();
}
```

Outputs : Hello Sunny!

C# Programming

C# Methods / Functions:

Parameters and Arguments: ***Information can be passed to methods as parameter. Parameters act as variables inside the method.***

- *They are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.*
- ***Example:***

```
static void MyMethod(string fname, int age)
{   Console.WriteLine(fname + " is " + age);
}
static void Main(string[] args)
{    MyMethod("Adi", 48);
     MyMethod("Prajwal", 18);
     MyMethod("Sunny", 16); }
```

Outputs : Adi is 48
Prajwal is 18
Sunny is 16

C# Programming

C# OOP:

OOP stands for Object-Oriented Programming, which is about creating objects that contain both data and methods.

- *Object-oriented programming has several advantages over procedural programming:*
  - *OOP is faster and easier to execute*
  - *OOP provides a clear structure for the programs*
  - *OOP helps to keep the C# code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug*
  - *OOP makes it possible to create full reusable applications with less code and shorter development time*

C# Programming

C# Object and Class:

C# is an object-oriented language, program is designed using objects and classes in C#.

- ***C# Object:*** *Object is a runtime entity, it is created at runtime. Object is an entity that has state and behavior. Here, state means data and behavior means functionality.*
  - *Example:* ***Student s1 = new Student();//creating an object of Student***
- ***C# Class :*** *In C#, class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.*
  - *Let's see an example of C# class that has two fields only.*

```
public class Student
{      int id;//field or data member
       String name;//field or data member
}
```

C# Programming

C# - Inheritance:

Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.

- *This also provides an opportunity to reuse the code functionality and speeds up implementation time.*
- ***Base and Derived Classes:*** *A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.*
  - *Example:* ***<acess-specifier> class <base_class>*** *//Base class*

```
{   ...     }
class <derived_class> : <base_class>    //Derived class
{   ...     }
```

C# Programming

C# - Polymorphism:

In object-oriented programming paradigm, polymorphism is often expressed as 'one interface, multiple functions'.

- *Polymorphism can be static or dynamic.*
  - *In **static polymorphism**, the response to a function is determined at the compile time.*
  - *In **dynamic polymorphism**, it is decided at run-time..*
- ***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:*
  - ***Function overloading***
  - ***Operator overloading***

C# Programming

C# - Polymorphism:

***Function overloading:* Also called Method overloading, having two or more methods with same name but different in parameters, is known as method overloading in C#.**

- *You can perform method overloading in C# by two ways:*
  - *By changing number of arguments*
  - *By changing data type of the arguments.*
- ***Example:***

```
using System;
public class Cal
{
    public static int add(int a, int b)
        {  return a + b;     }
    public static int add(int a, int b, int c)
        {  return a + b + c;      }
}
```

```
public class TestMemberOverloading
{       public static void Main( )
        {
        Console.WriteLine(Cal.add(15, 20));
        Console.WriteLine(Cal.add(15, 20, 25));
        }
}
```

C# Programming

C# - Polymorphism:

C# - Operator Overloading: ***You can redefine or overload most of the built-in operators available in C#. Thus a programmer can use operators with user-defined types as well.***

- *Overloaded operators are functions with special names the keyword operator followed by the symbol for the operator being defined. similar to any other function, an overloaded operator has a return type and a parameter list.*
- *Example,*
  - *function implements the addition operator (+) for a user-defined class Box. It adds the attributes of two Box objects and returns the resultant Box object.*

```
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;
}
```

C# Programming

C# - Interfaces:

An interface is defined as a syntactical contract that all the classes inheriting the interface should follow. The interface defines the 'what' part of the syntactical contract and the deriving classes define the 'how' part of the syntactical contract.

- *Interfaces define properties, methods, and events, which are the members of the interface. Interfaces contain only the declaration of the members.*
- *Interfaces are declared using the interface keyword. It is similar to class declaration. Interface statements are public by default.*
- *Syntax:*

```
public interface ITransactions
{     // interface members
      void showTransaction();
       double getAmount();
}
```

Example: // Interface

```
interface Student
{void StudentCourse( ); // interface method (does not have a body) }
class BCA : Student
{   public void StudentCourse( )
  {     // The body of StudentCourse( ) is provided here
        Console.WriteLine("BCA Students Study C#");      }    }
```

C# Programming

C# - Interfaces:

An interface is defined as a syntactical contract that all the classes inheriting the interface should follow. The interface defines the 'what' part of the syntactical contract and the deriving classes define the 'how' part of the syntactical contract.

- *Like abstract classes, interfaces cannot be used to create objects*
- *Interface methods do not have a body - the body is provided by the "implement" class*
- *On implementation of an interface, you must override all of its methods*
- *Interfaces can contain properties and methods, but not fields/variables*
- *Interface members are by default abstract and public*
- *An interface cannot contain a constructor (as it cannot be used to create objects)*

C# Programming

C# Enumerations:

An enum is a special "class" that represents a group of constants (unchangeable/read-only variables).

- *Enum is short for "enumerations", which means "specifically listed".*
- *To create an enum, use the enum keyword (instead of class or interface), and separate the enum items with a comma.*
- *By default, the first item of an enum has the value 0. The second has the value 1, and so on.*
- *Example:*

```
class Program
{
 enum Level
 {   Low,
     Medium,
     High
 }
```

```
//You can access enum items with the dot syntax:
static void Main(string[] args)
 {    Level myVar = Level.Medium;
      Console.WriteLine(myVar);
 }
}
```

C# Programming

C# - Exception Handling:

An exception is a problem that arises during the execution of a program.

A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

- *Exceptions provide a way to transfer control from one part of a program to another.*
- *C# exception handling is built upon four keywords:*
  - *try*
  - *catch*
  - *finally*
  - *throw*

```
try {        // statements causing exception  }
catch( ExceptionName e1 ) {   // error handling code }
catch( ExceptionName e2 ) {   // error handling code }
catch( ExceptionName eN ) {   // error handling code }
finally {   // statements to be executed }
```

C# Programming

C# - Exception Handling:

A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

- ***try*** *– A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.*
- ***catch*** *– A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.*
- ***finally*** *– The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.*
- ***throw*** *– A program throws an exception when a problem shows up. This is done using a throw keyword.*

C# Programming

C# - File I/O:

A file is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a stream.

- *The stream is basically the sequence of bytes passing through the communication path.*
- *There are two main streams: the **input stream** and the **output stream**.*
  - *The **input stream** is used for reading data from file (read operation).*
  - *The **output stream** is used for writing into the file (write operation).*
- *The File class from the System.IO namespace, allows us to work with files.*
- ***Example:***

```
using System.IO;  // include the System.IO namespace
File.SomeFileMethod();  // use the file class with methods
```

Discussion

Queries ?

Prof. K. Adisesha

9449081542

Thank You

ARRAYS

1.Arrays in c# are the collection of a variable of the same type ,storedtogether in the continues block of memory .

2.Arrays are objects in c#.They are the instances of the System.Array class and inherit its properties and methods.

Declaring an Array

Datatype[]arrayname ;

Example :

Int [] numbers;

Student[] studentArray;

Creating Arrays

int[] intarray; intArray=new int[5];

Initializing Arrays

```
int[] myIntArray =new int[5];
        myIntArray[0]=10;
        myIntArray[1]10;
        myIntArray[2]=10;
        myIntArray[3]=10;
        myIntArray[4]=10;
```

Or

```
Int[] numbers;
numbers = {10,20,30,40,50};
```

Accessing and modifying Array elements :

```
int [] numbers=new int[10];
 int firstnumber = number[0];
```

Modifying the Array element

```
number[0]=50;
```

```
using System;

namespace ArrayExample
{
    class Program
    {
        static void Main(string[] args)
        {
            // Creating an array of 5 integers and initializing it with values
            int[] numbers = new int[5] { 1, 2, 3, 4, 5 };
            Console.WriteLine("Array before modification:");
            // Accessing elements of the array and printing them
            for (int i = 0; i < numbers.Length; i++)
            {
                Console.Write(numbers[i] + " ");
            }
            Console.WriteLine();

            // Modifying the values of the second and fourth elements
            numbers[1] = 10;
            numbers[3] = 20;
            Console.WriteLine("Array after modification:");
            // Accessing elements of the array and printing them
            for (int i = 0; i < numbers.Length; i++)
            {
                Console.Write(numbers[i] + " ");
            }
            Console.WriteLine();
        }
    }
}
```

Output:

```
Array before modification:
1 2 3 4 5
Array after modification:
1 10 3 20 5
```

3.21.5 Multidimensional Arrays

In C#, a multi-dimensional array is an array that has more than one dimension. You can create a multi-dimensional array in C# by specifying multiple sets of square brackets.

Here's the syntax for declaring a multi-dimensional array:

```
data-type[,] arrayName = new data-type[size1, size2, ... size n];
```

Where data-type is the type of the elements in the array, arrayName is the name of the array, size1, size2, ..., size n are the dimensions of the array, and n is the number of dimensions.

Example : Here's an example of how to create and initialize a two-dimensional array:

```
int[,] numbers = new int[3, 3]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

We can access an element in a multi-dimensional array using the following syntax:

```
arrayName[index1, index2, ..., index n]
```

Where index1, index2, ..., index n are the indices of the elements in the array, and n is the number of dimensions.

Example : Here's an example that demonstrates how to access elements in a two-dimensional array:

```
int[,] numbers = new int[3, 3]
{
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};

Console.WriteLine(numbers[0, 0]);  // Output: 1
Console.WriteLine(numbers[1, 1]);  // Output: 5
Console.WriteLine(numbers[2, 2]);  // Output: 9
```

We can modify the values of elements in a multi-dimensional array in the same way as a one-dimensional array. For example:

```
numbers[1, 1] = 10;
Console.WriteLine(numbers[1, 1]);  // Output: 10
```

		C# Example Code and Output

`Sort`	Sorts the elements in an array in ascending order.	**Example :**
		`int[] numbers = {3,1,4,1,5,9,2,6,5,3};`
		`Array.Sort(numbers);`
		`Console.WriteLine(string.Join(", ", numbers));`
		Output:
		`1,1,2,3,3,4,5,5,6,9`

`Reverse`	Reverses the elements in an array.	**Example :**
		`int[] numbers = { 1, 2, 3, 4, 5 };`
		`Array.Reverse(numbers);`
		`Console.WriteLine(string.Join(", ", numbers));`
		Output:
		`5, 4, 3, 2, 1`

`Clear`	Sets a range of elements in an array to zero, to false, or to a null reference, depending on the element type.	**Example :**
		`int[] numbers = { 1, 2, 3, 4, 5 };`
		`Array.Clear(numbers, 0, 3);`
		`Console.WriteLine(string.Join(", ", numbers));`
		Output:
		`0, 0, 0, 4, 5`

`IndexOf`	Searches for the specified object and returns the index of the first occurrence within the entire array.	**Example :**
		`int[] numbers = { 1, 2, 3, 4, 5 };`
		`int index = Array.IndexOf(numbers, 3);`
		`Console.WriteLine(index);`
		Output:
		`2`

`Copy`	Copies a specified number of elements from a source array starting at a particular source index to a destination array starting at a specified destination index.	**Example :**
		`int[] source = { 1, 2, 3, 4, 5 };`
		`int[] destination = new int[3];`
		`Array.Copy(source, 1, destination, 0, 3);`
		`Console.WriteLine(string.Join(", ", destination));`
		Output:
		`2, 3, 4`

`BinarySearch`	Searches an array for a specified value and returns the index of the first occurrence, using a binary search algorithm.
`Find`	Searches for an element that matches the conditions defined by the specified predicate and returns the first occurrence within the entire array.
`FindAll`	Searches for elements that match the conditions defined by the specified predicate and returns all the occurrences as a new array.
`Resize`	Changes the number of elements of an array to the specified new size.
`ConvertAll`	Converts the elements of a specified array to the specified type and returns a new array.

Example :

```
int[] numbers = { 1, 2, 3, 4, 5 };
int index = Array.BinarySearch(numbers, 4);
Console.WriteLine(index);
```

Output:

```
3
```

Example :

```
int[] numbers = { 1, 2, 3, 4, 5 };
int firstEven = Array.Find(numbers, x => x % 2 == 0);
Console.WriteLine(firstEven);
```

Output:

```
2
```

Example :

```
int[] numbers = { 1, 2, 3, 4, 5 }; int[] evenNumbers = Array.
FindAll(numbers, x => x % 2 == 0);
Console.WriteLine(string.Join(", ", evenNumbers));
```

Output:

```
2, 4
```

Example :

```
int[] numbers = { 1, 2, 3, 4, 5 };
Array.Resize(ref numbers, 3);
Console.WriteLine(string.Join(", ", numbers));
```

Output:

```
1, 2, 3
```

Example :

```
string[] words = { "apple", "banana", "cherry" };
int[] lengths = Array.ConvertAll(words, x => x.Length);
Console.WriteLine(string.Join(", ", lengths));
```

Output:

```
5, 6, 6
```

3.22 Strings in C#

Stings are sequence of characters. A `string` variable contains a collection of characters surrou
by double quotes: string is an object of **System.String** class that represent sequence of charac
A string in C# is actually an object, which contain properties and methods that can perform cer
operations on strings.

Strings are immutable, which means that once we create a string, its value cannot be chan
However, we can create a new string by concatenating or manipulating the original string.

- **Declare and initialize a string in C# :**

```
string firstName = "Srikanth";
string lastName = "S";
```

- We can concatenate two or more strings using the + operator:

```
string fullName = firstName + " " + lastName;
Console.WriteLine(fullName);   // Output: Srikanth
```

- We can also use the string.Format method to create a formatted string:

```
string message = string.Format("Hello, my name is {0} {1}.", firstName, lastName);
Console.WriteLine(message);   // Output: Hello, my name is Srikanth S.
```

- We can access individual characters in a string using the square bracket notation:

```
char firstLetter = firstName[0];
Console.WriteLine(firstLetter);   // Output: S
```

- We can also compare two strings to see if they are equal or not:

```
bool areEqual = firstName == "Srikanth";
Console.WriteLine(areEqual);   // Output: True
```

Note that the == operator compares the values of two strings, not the references.

- We can also use the string.Length property to get the length of a string:

```
int length = firstName.Length;
Console.WriteLine(length);   // Output: 8
```

Stings are Immutable

In C#, strings are immutable, which means that once you create a string, its value cannot be changed. When you try to modify a string, a new string is created with the modified value, and the original string remains unchanged. Here's an example that demonstrates this concept.

Example:

```
using System;

namespace StringImmutabilityExample
{
    class Program
    {
        static void Main(string[] args)
        {
            string str1= "Hello";
            Console.WriteLine("Original string : " +str1);
            string str2= str1 + ", World!";
            Console.WriteLine("Modified string: " + str2);
        }
    }
}
```

Output:

```
Original string : Hello
Modified string: Hello, World!
```

Mutable Strings

Mutable Strings are those strings whose content can be changed without creating a new object. Mutable string uses StringBuilder class. C# mutable string in a memory can be changed or modified and no new memory will be created on appending a string.

Example:

```
using System;
using System.Text;

namespace StringBuilderExample
{
    class Program
    {
        static void Main(string[] args)
        {
            StringBuilder sb = new StringBuilder("Hello");
            Console.WriteLine("Original string: " + sb.ToString());

            sb.Append(", World!");
            Console.WriteLine("Modified string: " + sb.ToString());
        }
    }
}
```

Output:

```
Original string : Hello
Modified string: Hello, World!
```

Methods of C# String

There are various string methods in C#. Some of them are as follows:

Method	Description	C# Example Code and Output

`Length`	Gets the number of characters in the current string.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.Length);`
		Output:
		`5`

`ToUpper`	Converts the current string to uppercase.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.ToUpper());`
		Output:
		`HELLO`

`ToLower`	Converts the current string to lowercase.	**Example :**
		`string str = "HELLO";`
		`Console.WriteLine(str.ToLower());`
		Output:
		`hello`

`Substring`	Gets a substring of the current string.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.Substring(2, 3));`
		Output:
		`llo`

`Split`	Splits a string into an array of substrings based on a specified separator.	**Example :**
		`string str = "apple,banana,cherry";`
		`string[] words = str.Split(',');`
		`Console.WriteLine(string.Join(", ", words));`
		Output:
		`apple, banana, cherry`

`Replace`	Replaces all occurrences of a specified string with another specified string.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.Replace("l", "x"));`
		Output:
		`hexxo`

`IndexOf`	Reports the zero-based index of the first occurrence of a specified string within the current string.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.IndexOf("l"));`
		Output:
		`2`

`Trim`	Removes all leading and trailing white-space characters from a string.	**Example :**
		`string str = " hello ";`
		`Console.WriteLine("[" + str.Trim() + "]");`
		Output:
		`[hello]`

`TrimStart`	Removes all leading white-space characters from a string.	**Example :**
		`string str = " hello ";`
		`Console.WriteLine("[" + str.TrimStart() + "]");`
		Output:
		`[hello ]`

`TrimEnd`	Removes all trailing white-space characters from a string.	**Example :**
		`string str = " hello ";`
		`Console.WriteLine("[" + str.TrimEnd() + "]");`
		Output:
		`[ hello]`

`StartsWith`	Determines whether a string starts with the specified string.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.StartsWith("he"));`
		Output:
		`True`

`EndsWith`	Determines whether a string ends with the specified string.	**Example :**
		`string str = "hello";`
		`Console.WriteLine(str.EndsWith("lo"));`
		Output:
		`True`

3.23 Structures

A structure in C# is a value type that represents a collection of data fields with related data. Structures are similar to classes in that it can contain fields, methods, properties, and events, but it is different in that it is stored on the stack and passed by value.

he keyword **struct** declares **Student** as a new data type that can hold three variables of different ta types. These variables are known as members of fields or elements. The identifier **Student** can ow be used to crease variables of time **Student.**

Example: Creating Structure Object and Assigning values to members

```
// Create a structure object
Student s1;
// Assign values to the structure
s1.Name = "Snigdha";
s1.RollNumber = 1;
s1.TotalMarks = 90.0;
```

Program: Structures Demo

```
using System;

struct Student
{
    public string Name;
    public int RollNumber;
    public double TotalMarks;

    // Parameterized constructor
    public Student(string name, int rollNumber, double totalMarks)
    {
        Name = name;
        RollNumber = rollNumber;
        TotalMarks = totalMarks;
    }

    // Method to display student details
    public void DisplayDetails()
    {
        Console.WriteLine("Name: " + Name);
        Console.WriteLine("Roll Number: " + RollNumber);
        Console.WriteLine("Total Marks: " + TotalMarks);
    }

    // Property to get and set the Name field
    public string NameProp
    {
        get { return Name; }
        set { Name = value; }
    }
}
```

```
enum EnumName
{
    EnumValue1,
    EnumValue2,
    ...
    EnumValueN
}
```

- enum is the keyword used to declare an enumeration.
- EnumName is the name of the enumeration.
- EnumValue1, EnumValue2, ... EnumValueN are the named constants or values in the enumeration.

In the below example, a new enumeration named DaysOfWeek is declared with 7 named constants or values Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday.
A variable today is declared with the type DaysOfWeek and is assigned the value DaysOfWeek.Monday.
Finally, the value of the today variable is printed to the console, which will output the string Today is Monday.

Example:

```
enum DaysOfWeek
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}

DaysOfWeek today = DaysOfWeek.Monday;
Console.WriteLine("Today is " + today); // Output : Today is Monday
```

Program: Enumeration Demo

```
using System;

namespace EnumExample
{
    enum DaysOfWeek {
        Sunday,
        Monday,
        Tuesday,
        Wednesday,
        Thursday,
        Friday,
        Saturday
    }
```

```
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Enter the day of the week (Sunday to Saturday):");
        string input = Console.ReadLine();

        DaysOfWeek selectedDay;
        if (Enum.TryParse(input, out selectedDay))
        {
            switch (selectedDay)
            {
                case DaysOfWeek.Sunday:
                    Console.WriteLine("Today is Sunday. Time for some rest.");
                    break;
                case DaysOfWeek.Monday:
                    Console.WriteLine("Today is Monday.Time to start the work");
                    break;
                case DaysOfWeek.Tuesday:
                    Console.WriteLine("Today is Tuesday. Keep working.");
                    break;
                case DaysOfWeek.Wednesday:
                    Console.WriteLine("Today is Wednesday. Halfway through the w
                    break;
                case DaysOfWeek.Thursday:
                    Console.WriteLine("Today is Thursday. Keep pushing.");
                    break;
                case DaysOfWeek.Friday:
                    Console.WriteLine("Today is Friday. Almost the weekend.");
                    break;
                case DaysOfWeek.Saturday:
                    Console.WriteLine("Today is Saturday. Time for fun.");
                    break;
            }
        }
        else
        {
            Console.WriteLine("Invalid input. Please enter a valid day of the
        }

        Console.ReadKey();
    }
```

Output:

```
Enter the day of the week (Sunday to Saturday)
3
Today is Wednesday. Halfway through the week.
```

ation:

new enumeration named DaysOfWeek is declared with 7 named constants
unday, Monday, Tuesday, Wednesday, Thursday, Friday, and Saturday.

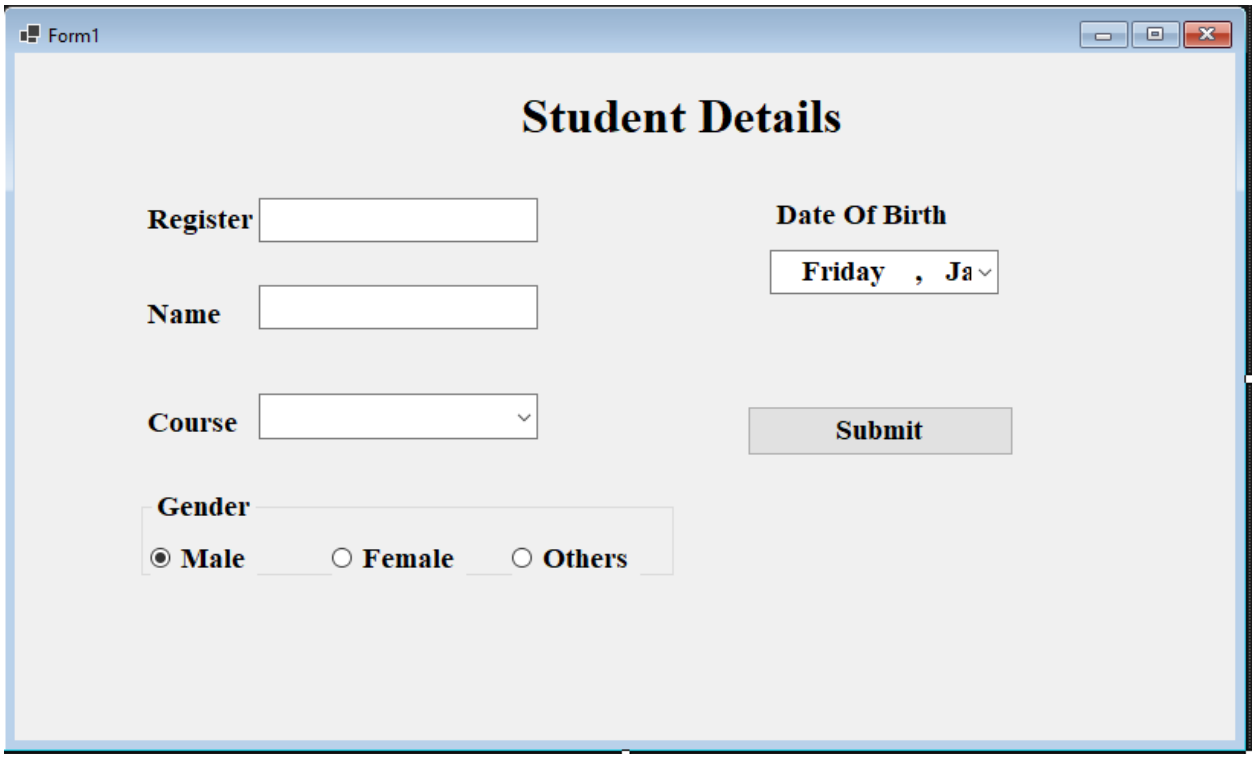
C# Lab Program

```
// program1 conditional statements
using System;
public class program1
{
   public static void Main()
   {
      int n1, n2, n3,ch, max;
      Console.WriteLine("Conditional Statements using switch case: ");
      Console.WriteLine("Enter first values: ");
      n1 = Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("Enter second values: ");
      n2 = Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("Enter third values: ");

      n3 = Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("1: Simple If: ");
      Console.WriteLine("2: If Else: ");
      Console.WriteLine("3: Nested If: ");
      Console.WriteLine("Enter your choice: ");
      ch = Convert.ToInt32(Console.ReadLine());
      switch (ch)
      {
         case 1:
            max = n1;

            if (n2 > max)
               max = n2;
            if (n3 > max)
               max = n3;
            Console.WriteLine("Maximum: (Using if) " + max);
            break;
         case 2:

            if (n1 > n2)
               max = n1;
            else
               max = n2;
            if (n3 > max)
               max = n3;
            Console.WriteLine("Maximum: (Using else if) " + max);
            break;
         case 3:

            if (n1 > n2)
               if (n1 > n3)
                  max = n1;
               else
                  max = n3;
            else
            if (n2 > n3)
               max = n2;
```

```
            else
                max = n3;
            Console.WriteLine("Maximum: (using nested if) " + max);
            break;
        }
    }
}


// program2 control statements
using System;
public class program1
{
    public static void Main()
    {
        int n, i = 1,ch,f=1 ;
        Console.WriteLine("Control Statements using switch case: ");
        Console.WriteLine("Enter a number to find factorial: ");
        n = Convert.ToInt32(Console.ReadLine());
        Console.WriteLine("1: while loop: ");
        Console.WriteLine("2: do while loop: ");
        Console.WriteLine("3: for loop: ");
        Console.WriteLine("Enter your choice: ");
        ch = Convert.ToInt32(Console.ReadLine());
        switch (ch)
        {
            case 1:
                while (i <= n)
                {
                    f *= i;
                    i++;
                }
                Console.WriteLine("Factorial: (Using while loop) " + f);
                break;
            case 2:
                do
                {
                    f *= i;
                    i++;
                }
                while (i <= n);
                Console.WriteLine("Factorial: (Using do-while loop) " + f);
                break;
            case 3:

                for (i=1; i<= n;i++)
                {
                    f  *= i;

                }
                Console.WriteLine("Factorial: (Using for loop) " + f);
                break;
        }
```

```
    }
}
```

//Program 3: windows controls in C#



//Program 4: multi threading in C#

```
using System;
using System.Threading;
public class Multi
{
   public static void method1()
   {
      for (int I = 0; I <= 10; I++)
      {

         Console.WriteLine("Method1 is : {0}", I);

         if (I == 5)
         {
            Thread.Sleep(6000);
         }
      }
   }
   public static void method2()
   {
      for (int J = 0; J <= 10; J++)
      {

         Console.WriteLine("Method2 is : {0}", J);
      }
   }

   static public void Main()
   {

      Thread thr1 = new Thread(method1);
      Thread thr2 = new Thread(method2);
      thr1.Start();
      thr2.Start();
   }
}
```

```
//Program 5: functions in C#

using System;

   class Sum
   {
      // User defined function
      public int add(int  a, int b)
      {
         return (a+b);
      }
      // Main function, execution entry point of the program
      static void Main(string[] args)
      {
      Sum s = new Sum();
      int x, y, z;
      Console.WriteLine("Enter first number");
      x = Convert.ToInt32(Console.ReadLine());
      Console.WriteLine("Enter second number");
      y = Convert.ToInt32(Console.ReadLine());
      z= s.add(x,y);
         Console.WriteLine("Sum= " + z);
      }
   }
```