C#

- **C#**
  - Server Side PROGRAMMING LANGUAGE

- **MVC** (MODEL-VIEW-CONTROLLER)
  - MVC is a design pattern by which you can develop web application

- **MSSSQL(**RDBMS**) -** Is a DBMS application which can store and manage the data by using **SQL**

- **ADO.NET –** .net tools that provide connection from frontend to backend. ADO.net i a data access technology

---

.NET FRAMEWORK: .net is a framework that is developed by Microsoft cooperation in 2000. **.net** framework provides platform to develop different types of application like web application, Mobile application, Desktop application windows forms, console application web services etc…

- Many programming languages are compatible with **.net** framework like **C#, Visual Basic, F#, J#, Python** etc.

CLR (Common Language Runtime**):** CLR provides a common runtime environment that is used to interpret HLL to the native code or byte code or LLL

Coding written in **.net** framework does not depends on the system's compiler, it has its own compiler that is **CLR**

CTS (Common Type System**): .net** support many programming language, **.net** has Created its own set of data type that is supported by every programming language used in **.net** framework that is **CTS.**

C#: C# is general purpose object oriented programming language.

C# is used to develop different types of application like web application, console application, windows forms etc... That's why C# is general purpose programming language.

In C#, coding is done in forms of class & object that's why C# is OOPs language. C# supports many other important features of OOPs like Constructor, Polymorphism, Abstraction, Encapsulation, Inheritance, Interface, Multi-threading, Exception handling etc...

VISUAL STUDIO : .NET IDE(Integrated Development System**) – IDE** is a software that is available with all possible components, services, libraries and tools required to develop different kind of application

NAMESPACE: Namespace is a collection of multiple namespace, classes and interfaces.

- **NameSpace**
  - ♦ **Interface and class**
    - ➢ **Method and function**
      - ▪ **Instruction and Statement**
        - **Keywords, identifier and operator (TOKEN)**

→**Namespace** are of **two** types

1. **Pre-defined**
    a. To use pre-defiend we use using keyword above all code

2. **User-defined**
    a. To declare our own namespace we use
        i. *Syntax:*

```
namespace name_space
{
    //statement of namespace we want to use another time
}
```

CLASS → Class is the collection of variable and methods. Variables are data member of class and method are member function of class.

Class is a concept which can contain some coding, but coding written in a class does not execute in real world untill, object of class is created.

To access members(variables,methods) of class in real world you need objects of class

Class does not have any memory sspace. Memory space for the member of class is creates when object of class is created.

→**Class** are of **two** types

1. **User defined**

Syntax:

```
class class_name //definition of class
    {
        //variable and methods
    }
```

2. **Pre-defined**

There are two ways to call the pre-defined class

i. create object of the class object_name.memeber of class
ii. class_name.member of class

Namespace: Namespace is the library, that is the collection of other namespaces, classes and interface.

Class: Class is the collection of variables and methods

Function: Function/ Method is the collection/ Block of statement that is used to perform a special task

Statement: Statements are the steps to achieve output from the Raw input. Statements are collection of keywords, operators, special, symbols, variables, constants values etc…

Basic Structure of C#

```
using System;
namespace firstApp
{
   internal class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Hello, World!");
      }
   }


}
```

Using: Using is a keyword that is used to call a namespace. If you want to use any pre-defined class or pre-defined function firstly you need to call its namespace in your code, only after you can call that class.

Syntax: using namespace_name;

System: System is a pre-defined namespaces of C#. System is the main namespace. All other namespace are declared within the system.

Namespace: Namespace is a keyword that is used to define a new namespace. Whenever you want to define your class in a namespace, namespace keyword is used.

Syntax:

```
namespace namespace_name
{
    //define your class
    //define your class
}
```

firstApp: firstApp is a user-defined namespace. It is an identifier; according to its use you can define any name to a user-defined namespace

internal: internal is a keyword. It is known as Access specifier/ Access modifier. Same as internal, we have many access modifiers like public, private, internal, protected internal.
Access modifiers are used to define scope of members (variables/methods) & types (namespace/class) that where at the outside this code is accessible or not

```
public int a;
internal class program;
private void Add ()
{
}
```

By default: By default all classes of **C#** is **internal type,** and all **members** of class is **private type**

Class: Class is a keyword that is used to define user-define classes. Whenever you want to create a new class in your program class keyword is used.

Syntax:

```
class class_name
{
    //variable
    //method
}
```

Program: Program is the name of user-defined class. It is an identifier. According to its use you can assign any name to a user-defined class.

Static: Static is a keyword. Static keyword can be used with user-defined class, variable, method. A static type member does not work with object of the class, because it does not have different memory spaces in every object of class. Only one time memory space is created for static type member.

Note: Main function in **C#** always should be **static type.**

void: void is a keyword that is used to define a user-defined function. Void is the datatype that has no memory space. Void is only used to define methods; it cannot be used to declare variables because void is used to show absence of data. Where of data. Where no data requires saving in computer memory, void is used

Syntax:

```
Void function_name ()
{
    //statement(s)
}
```

NOTE: void also known as return type of **UDF** (User Defined Function)

Main: Main is the function that every console application in **C#** must should have. Main is the function, which is very first called by compiler automatically. Main is the function that does not need to call explicitly/external, it is called implicitly by compiler.

Very first, code of Main function will execute.

Note: Main function of **C#** accepts string array type parameter. This parameter is used to accept command line inputs at the time of calling of main function.

(string [] args)

Console: Console is pre-defined static type class. Console is a class that contains functions, that is used for input output operations in Console Application like.

Write (), WriteLine (), ReadLine () etc…

Note: Because console is a static class, you cannot create object of the Console class.

WriteLine (): WriteLine () is pre-defined function of console class. WriteLine is a output function, that is used to print different type values on the console window. If you want to display any message to the user in console App, Write () or WriteLine () function is used

NOTE: After compilation CLR returns BYTE coded files that is .DLL (dynamic link library) OT. EXE

This file is compiled library of your code

Console: It is a pre-defined class that contains methods that is used for displaying output on Console window to the users or takes input the console window.

All input output function in C# is written/ defined within Console class. It is a static class so we cannot create object of the class. All members of class inherited with class name.

Console → Write () →**Print Output**

WriteLine () → **Print Output and then print next line**

Read () → **One word**

Syntax: **string variable_name = Console.Read()**

ReadKey () → **One Character**

Syntax: **string variable_name = Console.ReadKey()**

ReadLine () → **Whole line input**

→**It is used to read a value from Console Window, that value inputted by user and returns a string value**

Syntax: **string variable_name = Console.ReadLine()**

Keywords: Keywords are some reserved words of programming that is used to perform a pre-defined special task in program. In **C#** there are more than **79 keywords.** Like: - class, interface, using, namespace, int, float, string, double, byte, short, for, foreach, while, do…

Operators: Operators are symbols that are used to perform different calculation on one or more operands and returns output.

Operators are multiple types—

1. Arithmetic Operators (**+ , - , * , / , %**)
2. Relational Operators (**==,!= ,>,<,<=**)
3. Logical Operators (**&&,||,!**)
4. Assignment Operators (**=,+=,-=,*=,/=,%=**)
5. Increment & Decrement Operator (**++,--**)
6. Conditional Operator (**? ,:**)

Variables: Data Store

1 time Declaration and multiple time initializations

Declaration

```
byte b;
b = 0;
Console.WriteLine(b + "is the minimum value of byte type");
b = 255;
Console.WriteLine(b+ "is the maximum value of byte type");

short s;
s = -32768;
Console.WriteLine(s+ "is the minimum value of short type");
s = 32767;
Console.WriteLine(s+ "is the maximum value of short type");

int i;
i = -2147483648;
Console.WriteLine(i+ "is the minimum value of int type");
i = 2147483647;
Console.WriteLine(i+ "is the maximum value of int type");

long l;
```

```
l = -9223372036854775808;
Console.WriteLine(l+ "is the minimum value of long type");
l = 9223372036854775807;
Console.WriteLine(l+ "is the minimum value of long type");

float f1;
f1 = -3.40282347E+38f;
Console.WriteLine(f1+" minimum value of float");
f1 = 3.40282347E+38f;
Console.WriteLine(f1+" maximum value of float");

double dl;
dl = -1.7976931348623157E+308;
Console.WriteLine(dl + " minimum value of float");
dl = 1.7976931348623157E+308;
Console.WriteLine(dl + " maximum value of float");
char c;
string str;
bool ba;
sbyte sb;//signed byte
uint u;//unsigned integer
ulong ul;//unsigned short
ushort us;//unsigned long
```

→**Type Conversion:** Type conversion is a method to change one data type value to another data type:

1. **Implicit Type Conversion**: Automatic done by compiler. This type of type casting take place where no possibility of data loss.
   Byte > char>short>int>long>float>double>string(ascending)

   For Example:

   Smaller data type values are being stored in larger type variable

2. **Explicit Type Conversion:** Developer needs to write code to convert.
   Explicit type conversion where developer forcefully converts a data type value to another data type
   Explicit type conversion cannot do automatic, you have to write code.

Explicit type conversion can be done by using **Parse** function or **Convert** class

Type Conversion Example:

```
int a = 40.5f; //needs explicit type conversion
int b = 40.4;//needs explicit type conversion
long l = 40.3f;//needs explicit type conversion
long l1 = 40.3;//needs explicit type conversion
float f = 4000;
double d1 = 6876544;
long l2 = 563773;
double b1;
b1 = l2;
int i1 = 5454;
short s1;
//s1 = i1;
l2 = i1;//implicit type conversion

long l;
double d;
l = 50;
d = 50.6;
int i;
i = (int)l;//expicit type conversion only works on numeric type to numeric type
i = (int)d;//expicit type conversion
string s = "10";
int i1;
i1 = 3463454;
short s2;
s2 = (short)i1;
```

Type Casting: Type casting is a process to convert a data type value to another data type.

In **C#** type casting is done with the help of Convert class

Convert is pre-defined a static class, that provides many functions that is used for type conversion

Ex.

int variable = Convert.ToInt32(expression);

long variable_name = Convert.ToInt64(expression);

float variable_name = Convert.ToSingle(expression);

double variable_name = Convert.ToDouble(expression);

string variable_name = Convert.ToString(espression);

char variable_name = Convert.ToChar(expression);

```
Console.Write("Enter the value of first number: ");
int a = Convert.ToInt32(Console.ReadLine());

Console.Write("Enter the value of second number: ");
int b = Convert.ToInt32(Console.ReadLine());

Console.Write("Sum of "+a+ " and "+b+" is: "+(a+b));
```

Conditional Operator: Conditional statement is optional of if-else.

If you want to initialize value in a variable based on condition, conditional operator is used.

Conditional statement is used to execute different expression based on the given condition.

FOREACH: Foreach is a loop that can be used only for the collection type variables.

Foreach loop executes for each element of collection.

foreach is used to iterate element of collection one by one and returns the element from starting to the end.

Syntax:

```
foreach(data_type variable_name in collection){
    //statement
}
```

foreach loop returns element of collection one by one from starting index to the last.

User Defined Function: Function is the block of code that is used to perform a special task. Function is the collection of statement which is written to achieve an output from input.

There are 2 types of function

1. Pre-defined function / Built-in function: Which definition is already in the library, developer just needs to call it.
2. User defined function: A function which definition is written by developer is user defined function.
   Functions are re-usable. You need to define a function once and then you can call it multiple times.
   There are four types of UDF (User Defined Function)
   - No return type and no Parameter
   - Return type and no parameter
   - No return type with parameter
   - Return type with parameter

→Syntax to define an UDF/ Syntax to create a new function:

```
Return_type function_name(Parameter_list){
//statement(s)
//statement(s)
}
```

→How to create object:

Class_name obj_name = new class_name();

→No return type and no parameter

void function_name(){

//statement

}

→Return type and no parameter

int function_name(){

//statement

}

NOTE: Return type function saves a value in memory after execution and return that value to the caller.

Return type function is created when you want to return output of function to the caller.

Points:

- Return type function is a function which is not defines with void keyword.
- Return type function must should contain at least one statement that is return statement

→No return type with parameter

Points:

1. A variable declared within the parenthesis of UDF is known formal arguments.

2. A UDF with formal arguments is known as parameterized function/ UDF
3. You can use formal arguments without initialization within the block of UDF
4. At the time of calling user have to supply actual arguments to the parameterized function where actual arguments should be same in length and type as formal arguments.
5. A parameterized UDF is created when UDF wants external input from caller.

→No return type and no parameter

```
Void function_name(){
//Statements
}
```

Calling:

```
Function_name();
```

→Return type and no parameter

```
Return_type function_name(){
//statements
return return_value;
}
```

Calling:

```
Data_type variable_name = function_naem()
```

→No return type with parameter

Void function_name(data_type variable_name, data_type variable,….)

{

//Statements

}

Calling:

Function_name(value1,value2….);

→**Return type with parameter**

Return_type function_name(data_type variable_name, data_type variable,…){

//statements

Return return_value

}

Calling:

Return return_value = function_name(value1,value2….)

MVC

MVC stands for model view controller. The MVC is an Architectural pattern that separates an application into three main logical Components. The **Model**, **View** and **Controller.** Each of these Components is built to handle specific development of an application.

MODEL:

- Model Components Corresponds to all the data related logic that the user works with.

VIEW:

- The view Component is used for all the UI logic of the application.

CONTROLLER:

- Controller can act as an interface between Model and View Components to process all the business logic and incoming request.

How MVC works (Request Flow In MVC):

1. **User Request:**

   A user sends a request through the browser by typing a URL

   Or clicking a link.

2. **Routing:**

   The .NET routing engine maps the incoming request to the right controller action

3. **Controller:**

   The controller process the request, interacts with the model if necessary and return the view.

4. **View:**

   The view renders the HTML based on the data from the controller and sends it back to the browser.

5. **Response:**

   The Browser displays the final web page to the user.

***Razor View Engine:**

.net uses the Razor view engine to render HTML Views.

Razor syntax allows embedding C# code inside HTML using the @ Symbol.

Example:

@model List<employee>

@foreach(var employee in model){

employee.name

}

***Controller Actions**

Action Method: These are public method in a controller class that handle incoming request. Each Method Corresponds to a specific route or URL.

ActionResult Type:

1. **View result:** Resturn a view to the Browser.
2. **JsonResult:** Return **JSON** data.
3. **RedirectToActionResult:** Redirect the user to another action Method.
4. **ContentResult:** Returns Plain Text.
5. **PartialViewResult:** Renders a partial View.

***Layout in .net MVC:**

A layout is similar to a master page in traditional web development.

It defines a Consistent layout for your application(e.g. a header, footer, navigation bar etc..)

Example:

<html>

<head></head>

<body>

HEADER

@RenderBody()

Footer

</body>

</html>

How to call layout in our view page

@{

Layout = “~/View/Shared/_Layout.cshtml”

}

How @RenderBody Works:

@RenderBody act was as a placeholder for the content of view will be injection within the layout.

MVC:

Web Application:

Frontend (presentation layer) -> HTML, CSS, BOOTSTRAP

Business Logic->C#

Database -> MSSQL

M – Model – Model is used to **hold data travelling in web application**. Model is a **.cs** fille that is used as container to hold data for a while

V – View - View is the **presentation layer.** All frontend files is kept in view. In MVC view is **.cshtml** file, which can hold html content as well as C# content. In **.cshtml** file c# coding is inserted by **using razor syntax.**

C- Controller – Controller is the most important file of MVC. Controller contains **all business logics of web application.** As well as controller is responsible to handle all coming request from browser to the web application. Controller **connects view and model.**

Controller is a **.cs** file that has **suffix name controller** and it is inherited **via a pre-defined class named controller**

Folder Structure:

1. **References:** It contains libraries of C#. Libraries of C# has **.dll extension**
2. **App_Data:** It contains backup file of database.
3. **App_Start:** App_Start folder contains some .cs file that is runs firstly when web application starts
   Like: It contains a file that is RouteConfig.cs

   RouteConfig.cs: This file defines URL pattern of web application.
4. **Content:** Content folder contains static files of web application.

   Like: All images, audio, video, pdf, js, css, icon etc..
5. **Controller:** Controller folder contains all controllers of web application. Every controller has suffix “controller”. All controllers are .cs files. That has methods that is known as action methods.