Course Description:

ASP Web Controls, Basic Concepts of Server Side Programmimg using C-Sharp Programming Language over .NET Framework.

Course Duration: 10 hours

Course Goals and Objectives:

After completing this course, trainee will be able to understand the basic concepts of ASP.NET Like Architecture of Web Application, State Management, Validation, CSS, Event Handling, ASP Web Controls, User and Custom Controls.

Client-Server Architecture

Client-Server Architecture is a distributed system architecture where the workload of client and server are separated. Clients are those running programs like web browser or mobile app who request for the services or resources and Server means the running program acts as a resource provider. The server host several programs at its end for sharing resources to its clients whenever requested. Client and server can be on the same system or may be in a network.

Client Server architecture is centralised resource system where Server contain all the resources. The server is highly secured and scalable to respond clients. Client/Server Architecture is Service Oriented Architecture that means client service will never be disrupted.

Why use Client-Server architecture?

The Client-Server architecture decrease network traffic by providing a query response rather than total file transfer.
It allows multi-user updating through a graphical user interface to a shared database. It uses SQL or some other language statements to communicate between Client and Server.

Types of Client Server Architecture:

1 tier architecture
2 tier architecture
3 tier architecture

Two Tier Architecture

In this type of client server environment user interface is stored at client machine and database are stored on server. Database logic & business logic are stored at either client or server but it must be unchanged. If Business Logic & Data Logic are stored at client side, it is called fat client thin server architecture. If Business Logic & Data Logic are stored on server, it is called thin client fat server architecture. This kind of architecture are affordable and comparatively better.

Three Tier Architecture

In this type of client server environment presentation layer, business logic and data layer run of different machines. Client request goes to data server through that middle layer and the response of data server is firstly accepted by middle-ware then to client. This architecture overcomes all the drawbacks of 2-tier architecture and gives best performance. It is costly and easy to handle. The middle-ware stores all the business logic and data access logic. If there are multiple Business Logic & Data Logic, it is called n-tier architecture.

Application Architecture (3-Tier Architecture)

3-tier application architecture is a modular client-server architecture that consists of a presentation tier, an application tier and a data tier. The data tier stores information, the application tier handles logic and the presentation tier is a graphical user interface (GUI) that communications with the other two tiers.

Presentation Tier

The presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as ASP.NET, JSP, HTML5, JavaScript, CSS, and JQuery.

Business / Application Tier

The application tier contains the functional business logic which drives an application’s core capabilities. It’s often written in Java, C#.NET, VB.NET, Python, PHP.

Data Tier

The data tier comprises of the database/data storage system and data access layer. Examples of such systems are MySQL, Oracle, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

Benefits to using 3-layer architecture

There are many benefits to using a 3-layer architecture including speed of development, scalability, performance, and availability.

Modularizing different tiers of an application gives development teams the ability to develop and enhance a product with greater speed than developing a singular code base because a specific layer can be upgraded with minimal impact on the other layers.

Scalability is another great advantage of 3-layer architecture. By separating out the different layers you can scale each independently depending on the need at any given time. For example, if you are receiving many web requests but not many requests which affect your application layer, you can scale your web servers without touching your application servers. This allows you to load balance each layer independently, improving overall performance with minimal resources.

By having disparate layers you can also increase reliability and availability by hosting different parts of your application on different servers and utilizing cached results. With a 3-layer application, the increased independence created when physically separating different parts of an application minimizes performance issues when a server goes down.

General Page Life-Cycle Stages

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. ASP.NET web page life cycle consists of following six stages:

1- **Page request-** The page request occurs before the page life cycle begins. When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page), or whether a cached version of the page can be sent in response without running the page.

2- **Start-** In the start stage, page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property. The page also sets the UICulture property.

3- **Initialization-** During page initialization, controls on the page are available and each control's UniqueID property is set. A master page and themes are also applied to the page if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.

4- **Load-**During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.

5- **Rendering-** Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the Output Stream object of the page's Response property.

6- **Unload-** The Unload event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and cleanup is performed.

Life-Cycle Events

Within each stage of the life cycle of a page, the page raises events that you can handle to run your own code. For control events, you bind the event handler to the event, either declaratively using attributes such as onclick, or in code.
Pages also support automatic event wire-up, meaning that ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. If the AutoEventWireup attribute of the @ Page directive is set to true, page events are automatically bound to methods that use the naming convention of Page_event, such as Page_Load and Page_Init.

1- **PreInit -** Raised after the start stage is complete and before the initialization stage begins.
Use this event for the following:
   1- Check the IsPostBack property to determine whether this is the first time the page is being processed. The IsCallback and IsCrossPagePostBack properties have also been set at this time.
   2- Create or re-create dynamic controls.
   3- Set a master page dynamically.

4- Set the Theme property dynamically.
5- Read or set profile property values.

If the request is a postback, the values of the controls have not yet been restored from view state. If you set a control property at this stage, its value might be overwritten in the next event.

2- **Init -** Raised after all controls have been initialized and any skin settings have been applied. The Init event of individual controls occurs before the Init event of the page.
Use this event to read or initialize control properties.

3- **InitComplete-** Raised at the end of the page's initialization stage. Only one operation takes place between the Init and InitComplete events: tracking of view state changes is turned on. View state tracking enables controls to persist any values that are programmatically added to the ViewState collection. Until view state tracking is turned on, any values added to view state are lost across postbacks. Controls typically turn on view state tracking immediately after they raise their Init event.
Use this event to make changes to view state that you want to make sure are persisted after the next postback.

4- **PreLoad-** Raised after the page loads view state for itself and all controls, and after it processes postback data that is included with the Request instance.

5- **Load-** The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page.
1- Use the OnLoad event method to set properties in controls and to establish database connections.
2- In a postback request, if the page contains validator controls, check the IsValid property of the Page and of individual validation controls before performing any processing.

6- **LoadComplete-** Raised at the end of the event-handling stage.
Use this event for tasks that require that all other controls on the page be loaded.

7- **PreRender-** Raised after the Page object has created all controls that are required in order to render the page, including child controls of composite controls. (To do this, the Page object calls EnsureChildControls for each control and for the page.)
The Page object raises the PreRender event on the Page object, and then recursively does the same for each child control. The PreRender event of individual controls occurs after the PreRender event of the page.
Use the event to make final changes to the contents of the page or its controls before the rendering stage begins.

8- **PreRenderComplete-** Raised after each data bound control whose DataSourceID property is set calls its DataBind method.

9- **SaveStateComplete-** Raised after view state and control state have been saved for the page and for all controls. Any changes to the page or controls at this point affect rendering, but the changes will not be retrieved on the next postback.

10- **Render-** This is not an event; instead, at this stage of processing, the Page object calls this method on each control. All ASP.NET Web server controls have a Render method that writes out the control's markup to send to the browser.
If you create a custom control, you typically override this method to output the control's markup. However, if your custom control incorporates only standard ASP.NET Web server controls and no custom markup, you do not need to override the Render method. A user control (an .ascx file) automatically incorporates rendering, so you do not need to explicitly render the control in code.

11-Unload- Raised for each control and then for the page. In controls, use this event to do final cleanup for specific controls, such as closing control-specific database connections.
For the page itself, use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks.
Note: During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream. If you attempt to call a method such as the Response.Write method, the page will throw an exception.

Event Handling

An event is a message sent by an object to signal the occurrence of an action. The action can be caused by user interaction, such as a button click, or it can result from some other program logic, such as changing a property's value. The object that raises the event is called the event sender.

An event is an action or occurrence such as a mouse click and a key press. A web page communicates with server through events. When events occur, the application should be able to respond to it and manage it.

Events in ASP.NET raised at the client machine, and handled at the server machine. When an event is raised the browser handles this client-side event by posting it to the server. The server has a subroutine describing what to do when the event is raised; it is called the event-handler. Therefore, when the event message is transmitted to the server, it checks whether the specific event has an associated event handler. If it has, the event handler is executed.

Event Arguments

ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.

```
private void EventName (object sender, EventArgs e);
```

Example: The Click event is raised when the Button control is clicked.

```
<%@ Page Language="C#" %>
<form id="form1" runat="server">
      <asp:Button id="btnSave" OnClick="Save Button Clicked"
                        runat="server" Text="Save" />

      <asp:Label id = "lblMessage" Text="ASP Event Generated"
                        runat="server"    Visible = "false" />
</form>

<script runat="server">

void Page_Load(Object sender, EventArgs e)
{
      /*
```

```
            Manually register the event-handling method for the Click event
            of the Button control.
        */

        btnSave.Click += new EventHandler (this.btnSave_Click);
}

void SaveButton_Click(Object sender, EventArgs e)
{
        // When the button is clicked, change the button text, and disable it.

        Button clickedButton = (Button) sender;
        clickedButton.Text = "Button Clicked";
        clickedButton.Enabled = false;

        // Display the label text.

        lblMessage.Visible = true;
}
</script>
```

How to create Button Click Event Handler

Create a ASP.NET Web Form page and Draw a Web Control. Go to Code part of ASPX page Set value of event attribute (OnClick, OnCheckedChanged, OnSelectedIndexChanged) to method name that handles the event.

Create an event handler method which handles the event generated by web control during runtime. Write appropriate code in handler method to handle the event.

```
<asp:Button ID="btnclick" runat="server" Text="Click"
                                        onclick = "btnSave_Click" />
void btnSave_Click(Object sender, EventArgs e)
{
        // When the button is clicked, change the button text, and disable it.
        Button clickedButton = (Button) sender;
        clickedButton.Text = "Button Clicked";
        clickedButton.Enabled = false;

        // Display the greeting label text.
        GreetingLabel.Visible = true;
}
```

Select the button and right click and select properties. In the properties window click a yellow icon of event. Select event whichever you want to create. It automatically creates handler method in ASPX.CS Code-Behind File.
To write event handler code you can double click the the button, it will take you to code behind file where you can write the code if the user click button what action has to be performed.

State Management

View state, control state, hidden fields, cookies, and query strings all involve storing data on the client in various ways. However, application state, session state, and profile properties all store data in memory on the server. Each option has distinct advantages and disadvantages, depending on the scenario.

View State

ViewState is a important client side state management technique. ViewState is used to store user data on page at the time of post back of web page.
ViewState does not hold the controls, it holds the values of controls. It does not restore the value to control after page post back. ViewState can hold the value on single web page, if we go to other page using response.redirect then ViewState will be null.

- ViewState stores data on single page
- ViewState is client side state management technique

Store Value in View State.

```
protected void btnReset_Click(object sender, EventArgs e)
{
        ViewState["userName"] = txtUserName.Text;
        txtUserName.Text = "";
}
```

Retrieve Stored Value from View State.

```
protected void btnRestore_Click(object sender, EventArgs e)
{
        txtUserName.Text = ViewState["userName"].ToString()
        lblMessage.Text = "Recovered: Value of TextBox Was " +
                                ViewState["userName"].ToString();
}
```

Session State

ASP.NET provides session-state management to enable you to store information associated with a unique browser session across multiple requests. You can store a collection of values referenced by a key name or by numerical index. Access to session values and functionality is available using the HttpSessionState class, which is accessible through the Session property of the current HttpContext, or the Session property of the Page.
Session data is associated with a specific browser session using a unique identifier. By default, this identifier is stored in a non-expiring session cookie in the browser, but you can also configure your application to store the session identifier in the URL by setting the cookieless attribute to true or UseUri in the sessionState element of your application configuration.

```
<sessionState
            mode="InProc"
            cookieless="true"
            timeout="90" />

<sessionState mode="InProc"
            timeout="120"
            cookieless= "UseUri" />
```

Sessions are started during the first request and session values will persist as long as a new request is made by the browser before the number of minutes specified in the Timeout property pass.

1. Session stores data on whole website pages
2. Session is a server side state management technique.
3. Session state does not persist across ASP.NET application boundaries. If a browser navigates to another application, the session information is not available to the new application.

Save State Information in Session

State Information can be saved in Session using Session[Key] = value statement.

```
protected void btnSubmit_Click(object sender,EventArgs e)
{
        Session["userName"] = txtName.Text;
        Session["userMobile"] = txtMobile.Text;
        Session["userID"] = txtID.Text;
        Session["Employee"] = new Employee(); // Save Employee Object
}
```

Retrieve State Information in Session

State Information can be saved in Session using value = Session[Key] statement.

```
protected void btnGetUserDetails_Click(object sender,EventArgs e)
{
        Employee emp = (Employee) Session["Employee"];

        txtName.Text = Session["userName"].ToString();
        txtMobile.Text = Session["userMobile"].ToString();
        txtID.Text = Session["userID"].ToString();
}
```

Application State

Application domain for an application is created only when the first request is received by the web server. We can categorise Application life cycle in several stages. These can be:

Stage 1: User first requests any resource from the webserver.
Stage 2: Application receives very first request from the application.
Stage 3: Application basic Objects are created.
Stage 4: An HTTPapplication object is assigned to the request.
Stage 5: And the request is processed by the HTTPApplication pipeline.

Application state is one of the ways available to store some data on the server and faster than accessing the database. The data stored on the Application state is available to all the users (Sessions) accessing the application. So application state is very useful to store small data that is used across the application and same for all the users. Also we can say they are global variables available across the users. We should not store heavy data in ApplicationState because it is stored on the server and can cause performance overhead if it is heavy. Technically the data is shared amongst users via HTTPApplcationState class and the data can be stored here in key value pair. It can also be accessed through Application property of the HTTPContext class.

An instance of HttpApplicationState is created when first time a request comes from any user to access any resource from the application. And this can be accessed through the property Application property of HTTPContext Object.

How to Save values in Application State

Application state stores the data as of Object type, so at the time of reading, the values need to be converted in the appropriate type. So normally, we use to store the Application wise data in Application state which is shared across the users. So we can save the data in Application_OnStart method in Global.asax file as:

```
void Application_Start(object sender, EventArgs e)
{
        Application["Message"] = "Welcome to Equipment Info System";
}
```

We can also save object of some Class in Application variable.

```
Application["EmployeeObject"] = objEmployee;

protected void Page_Load(object sender, EventArgs e)
{
        Application["Message"] = "Corona Virus: Stay Home, Stay Safe";
}
```

We can retrieve Application data using following statement

```
if(Application["Message"] !=null)
{
        string message = Application["Message"].ToString();
}
```

Web Controls

The ASP.NET page framework includes a number of built-in server controls that are designed to provide a more structured programming model for the Web. These controls provide the following features:

1- Automatic state management.
2- Simple access to object values without having to use the Request object.
3- Ability to react to events in server-side code to create applications that are better structured.

4- Common approach to building user interfaces for Web pages.
5- Output is automatically customized based on the capabilities of the browser.

Properties of Web Controls

DisabledCssClass- Gets or sets the CSS class to apply to the rendered HTML element when the control is disabled.

Enabled- Gets or sets a value indicating whether the Web server control is enabled.

IsEnabled- Gets a value indicating whether the control is enabled

CssClass- Gets or sets the Cascading Style Sheet (CSS) class rendered by the Web server control on the client

IsViewStateEnabled- Gets a value indicating whether view state is enabled for this control

ViewState- Gets a dictionary of state information that allows you to save and restore the view state of a server control across multiple requests for the same page.

Visible- Gets or sets a value that indicates whether a server control is rendered as UI on the page (Inherited from Control).

Width- Gets or sets the width of the Web server control

UniqueID- Gets the unique, hierarchically qualified identifier for the server control.

HTML Server Controls

The HTML server controls are Hypertext Markup Language (HTML) elements that include a runat=server attribute. The HTML server controls have the same HTML output and the same properties as their corresponding HTML tags. In addition, HTML server controls provide automatic state management and server-side events. HTML server controls offer the following advantages:

1- The HTML server controls map one to one with their corresponding HTML tags.

2- When the ASP.NET application is compiled, the HTML server controls with the runat=server attribute are compiled into the assembly.
3- Most controls include an OnServer Event for the most commonly used event for the control. For example, the <input type=button> control has an OnServerClick event.
4- The HTML tags that are not implemented as specific HTML server controls can still be used on the server side; however, they are added to the assembly as HtmlGenericControl.
5- When the ASP.NET page is reposted, the HTML server controls keep their values.

Web Server Controls

Web controls are very similar to the HTML server controls such as Button, TextBox, and Hyperlink, except that Web controls have a standardized set of property names. Web server controls are created on the server. The runat="server" attribute is added to the control. The control is processed by the server.

Web server controls offer the following advantages:
1- Make it easier for manufacturers and developers to build tools or applications that automatically generate the user interface.
2- Simplify the process of creating interactive Web forms, which requires less knowledge of how HTML controls work and make the task of using them less prone to errors.

The System.Web.UI.WebControls.WebControl base class contains all of the common properties. Most of the Web server controls derive from this class.
To use a Web server control, use the following syntax (which uses the TextBox control as an example):

```
<asp:textbox text="Enter Your Login ID" runat=server />
```

Web server controls can be divided into four categories:
1- Basic Web Controls.
2- Validation Controls.
3- List Controls.
4- Rich Controls.

Basic Web Controls.

Label control

TextBox control
ListBox control
CheckBox and CheckBoxList control
RadioButton and RadioButtonList control
DropdownList control
Button control
ImageButton control
Wizard control

List Controls

ListBox Web Server Control
CheckBoxList Web Server Control
RadioButtonList Web Server Control
Repeater Web Server Control
DataList Web Server Control
DataGrid Web Server Control
DropDownList Web Server Control

Validation Controls

Validation controls are used to validate the values that are entered into other controls of the page. Validation controls perform client-side validation, server-side validation, or both, depending on the capabilities of the browser in which the page is displayed.

Validation controls offer the following advantages:

You can associate one or more validation controls with each control that you want to validate.
The validation is performed when the page form is submitted.
You can specify programmatically whether validation should occur, which is useful if you want to provide a cancel button so that the user can exit without having to fill valid data in all of the fields.

The validation controls automatically detect whether validation should be performed on the client side or the server side.

1. RequiredFieldValidator Control
2. RangeValidator Control
3. CompareValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary Control

User Controls

Often, you may want to reuse the user interface of your Web Form without having to write any extra code. ASP.NET enables you to do this by converting your Web Forms into user controls. User controls, which have the ASCX file extension, can be used multiple times within a single Web Form.
To convert a Web Form into a user control, follow these steps:
Remove all <html>, <head>, <body> and <form> tags.
If the @ Page directive appears in the page, change it to @ Control.
Include a className attribute in the @ Control directive so that the user control is typed strongly when you instantiate it.
Give the control a descriptive file name, and change the file extension from aspx to ascx.

```
<%@Control Language="C#" AutoEventWireup = "true"
      CodeFile="NameUserControl.ascx.cs" Inherits="NameUserControl" %>

<h3>Name User Control</h3>
<table>
   <tr>  <td>First Name</td>
      <td>
            <asp:TextBox ID = "txtFirstName" runat = "server">
            </asp:TextBox>
      </td>
   </tr>
   <tr>     <td>Last Name: </td>
      <td>
            <asp:TextBox ID="txtLastName" runat="server">
            </asp:TextBox>
      </td>
   </tr>
   <tr><td> <asp:Button ID="btnSave" runat = "server" Text = "Save"
                  Onclick = "btnSave_Click" />
        </td>
   </tr>
</table>
<asp:Label ID="lblMessage" runat="server" Text = " "></asp:Label>
```

Register and use the user Control in your web page using following ASP statements. Add the Register directive just below the @Page directive.

```
<%@ Register Src = "~/ NameUserControl.ascx " TagPrefix="uc"
```

```
            TagName="FullName"%>

<uc:Student ID = "ctrlName" runat = "server" />
```

Custom Controls

In addition to the built-in Web controls, ASP.NET also allows you to create your own custom controls. It may be useful to develop custom controls if you are faced with one of these scenarios:

1. You need to combine the functionality of two or more built-in Web controls.
2. You need to extend the functionality of a built-in control.
3. You need a control that is completely different than any of the controls that currently exist.

To create this control, take the following steps:

1. Create a new website. Right click the solution (not the project) at the top of the tree in the Solution Explorer.
2. In the New Project dialog box, select ASP.NET Server Control from the project templates.

The above step adds a new project and creates a complete custom control to the solution, called ServerControl1. In this example, let us name the project CustomControls.

To use this control, this must be added as a reference to the web site before registering it on a page.

1- To add a reference to the existing project, right click on the project (not the solution), and click Add Reference.
2- Select the CustomControls project from the Projects tab of the Add Reference dialog box. The Solution Explorer should show the reference.

To use the control on a page, add the Register directive just below the @Page directive:

```
<%@ Register Assembly="CustomControls"  Namespace="CustomControls"
TagPrefix="ccs" %>
```

Further, you can use the control, similar to any other controls.

```
<form id="form1" runat="server">
   <div>
```

```
    <ccs:ServerControl1 runat="server" Text = "Custom Server Control" />
  </div>
</form>
```

ASP.NET Validation

ASP.NET Validation Controls are powerful server controls that are used for validating user input. These controls provides both server side and client side validation.

The client side validation features can be used to give your users an improved UI experience. By using the client side features, your users will not need to wait for a full page postback to occur only to discover that they have failed to fill out your form completely. Instead, the client side features can be using to notify the user immediately (before the page is submitted) about any missing and/or invalid data. Here is a list of the validation controls

1. RequiredFieldValidator
2. RegularExpressionValidator
3. CustomValidator
4. ValidationSummary
5. RangeValidator
6. CompareValidator

Property

A property is a member that provides a flexible mechanism to read, write, or compute the value of a private field. Properties can be used as if they are public data members, but they are actually special methods called accessors. This enables data to be accessed easily and still helps promote the safety and flexibility of methods.

Properties Overview

Properties enable a class to expose a public way of getting and setting values, while hiding implementation or verification code. A get property accessor is used to return the property value, and a set property accessor is used to assign a new value. These accessors can have different access levels.

Properties can be read-write (they have both a get and a set accessor), read-only (they have a get accessor but no set accessor), or write-only (they have a set accessor, but no get accessor). Write-only properties are rare and are most commonly used to restrict access to sensitive data.

Simple properties that require no custom accessor code can be implemented either as expression body definitions or as auto-implemented properties.

```
using System;
public class SaleItem
{
        string _name; decimal _cost;
        public SaleItem(string name, decimal cost)
        {
                _name = name;
                _cost = cost;
        }

// Implementation with Lemda Expression
public string Name
{
        get => _name;           // Equivalent to: get {return _name;}
        set => _name = value;   // Equivalent to: set {_name = value;}
}

public decimal Price
{
        get {
                if (_cost != null)
                    return _cost;
        }
        set {
                _cost = value;
        }
}

class Program
{
        static void Main(string[] args)
        {
                var item = new SaleItem("Floor Bag", 220);
                Console.WriteLine($"{item.Name}: sells for {item.Price}");
        }
}
// The example displays output like the following:
// Floor Bag: sells for 220
```

Auto-implemented properties

In some cases, property get and set accessors just assign a value to or retrieve a value from a backing field without including any additional logic. By using auto-implemented properties, you can simplify your code while having the C# compiler transparently provide the backing field for you.
If a property has both a get and a set accessor, both must be auto-implemented. You define an auto-implemented property by using the get and set keywords without providing any implementation.

```
using System;
public class SaleItem
{
        public string Name { get; set; }
        public decimal Price { get; set; }
}

class Program
{
        static void Main(string[] args)
        {
                var item = new SaleItem
                {
                        Name = "Reebok Shoes",
                        Price = 1230
                };

                Console.WriteLine($"{item.Name}: sells for {item.Price}");
        }
}
// Output: Reebok Shoes: sells for 1230
```

What is CSS?

1- CSS stands for Cascading Style Sheets
2- CSS describes how HTML elements are to be displayed on screen, paper, or in other media
3- CSS saves a lot of work. It can control the layout of multiple web pages all at once
4- The style definitions are normally saved in external .css files known as external style-sheet.

5- With an external stylesheet file, you can change the look of an entire website by changing just one file
6- CSS is used to define styles for your web pages, including the design, layout and variations in display for different devices and screen sizes.

```
body
{
  background-color: lightblue;
}

h1
{
  color: white;
  text-align: center;
}

p
{
  font-family: verdana;
  font-size: 20px;
}
```

CSS Syntax

Selector { attribute : value; attribute : value; }

```
p {  color: red;  text-align: center; }
```

1- p is a selector in CSS (It points to the HTML element to style: <p>).
2- color is a property, and red is the property value
3- text-align is a property, and center is the property value

A CSS rule-set consists of a selector and a declaration block:

1- The selector points to the HTML element you want to style.
2- The declaration block contains one or more declarations separated by semicolons.
3- Each declaration includes a CSS property name and a value, separated by a colon.
4- A CSS declaration always ends with a semicolon, and declaration blocks are surrounded by curly braces.

CSS Selectors

CSS selectors are used to "find" (or select) the HTML elements you want to style. We can divide CSS selectors into five categories:

1- Simple selectors (select elements based on name, id, class)
2- Combinator selectors (select elements based on a specific relationship between them)
3- Pseudo-class selectors (select elements based on a certain state)
4- Pseudo-elements selectors (select and style a part of an element)
5- Attribute selectors (select elements based on an attribute or attribute value)

The CSS id Selector

1- The id selector uses the id attribute of an HTML element to select a specific element.
2- The id of an element is unique within a page, so the id selector is used to select one unique element!
3- To select an element with a specific id, write a hash (#) character, followed by the id of the element.

```
#para1
{
        text-align: center;
        color: red;
        border-style : solid
}
```

The CSS class Selector

The class selector selects HTML elements with a specific class attribute.
To select elements with a specific class, write a period (.) character, followed by the class name.

```
<h1 class="center">    Red and center-aligned heading      </h1>
<p class="center">     Red and center-aligned paragraph.  </p>
<p class="left">       Red and left-aligned paragraph.     </p>
```

This CSS applied to both h1 and p elements having class = "center"

```
.center
{
  text-align: center;
  color: red;
}
```
This CSS applied to only p element having class = "center"
```
p.center
{
	text-align: center;
	color: red;
}
```

The CSS Grouping Selector
The grouping selector selects all the HTML elements with the same style definitions.

The following CSS code applicable to all h1, h2, and p elements.

```
<style>
	h1, h2, p
	{
		text-align: center;
		color: red;
	}
</style>

<h1>GP Lohaghat</h1>
<h2>Department of IT</h2>
<p>IT Final Year</p>
```

Three Ways to Insert CSS
There are three ways of inserting a style sheet:
External CSS
Internal CSS
Inline CSS

External CSS
With an external style sheet, you can change the look of an entire website by changing just one file. Each HTML page must include a reference to the external style sheet file inside the <link> element, inside the head section.

```
<head>
<link rel="stylesheet" type="text/css" href="Style.css">
</head>
```

An external style sheet can be written in any text editor, and must be saved with a .css extension.
The external .css file should not contain any HTML tags.

File Name: Style.css

```
body {      background-color: lightblue;  }

h1
{
        color: navy;
        margin-left: 20px;
}

p
{
        border-style: dotted solid;
        border-width: 5px;
        border-color: red;
}
```

References:

1- https://www.tutorialspoint.com
2- https://www.codeproject.com
3- https://docs.microsoft.com
4- https://www.w3schools.com/
5- https://www.c-sharpcorner.com/