AWP Assignment

1. Explain Calendar control with example.
   ANS:-
   a. The Calendar control presents a miniature calendar that you can place in any web page.
   b. the Calendar can be programmed as a single object (and defined in a single simple tag), but it renders itself with dozens of lines of HTML output.
   c. The Calendar control presents a single-month view.
   d. The user can navigate from month to month by using the navigational arrows, at which point the page is posted back and ASP.NET automatically provides a new page with the correct month values.
   e. When the user clicks a date, the date becomes highlighted in a gray box.
   f. Different selection modes can be configured CalendarSelectionMode property. It allows one to select days(Day), entire weeks (DayWeek), whole months (DayWeekMonth), or render the control as a static calendar that doesn't allow selection (None).
   g. he Calendar.FirstDayOfWeek property can be set to configure how a week is shown.
   h. the SelectedDates property provides a collection of all the selected dates when multiple dates are selected.
   i. The Calendar can also be formatted using formatting -related properties, the dates available for selection can be restricted using the Calendar.DayRender event and perform certain actions when a certain date is selected using the SelectionChanged and VisibleMonthChanged events.
   j. Eg:
      Aspx file
```
<asp:Calendar id="MyCalendar" runat="server" onDayRender =
“MyCalendar_DayRender” SelectionMode=”day”/>
```
      CS file
```
protected void MyCalendar_DayRender(Object source, DayRenderEventArgs e)
{
if (e.Day.IsWeekend || e.Day.Date.Year > 2013)
 {
e.Day.IsSelectable = false;
}
}
```

2. How CustomValidator control is used to define our own function to validate data?
   a. Validation is performed by a user-defined function

b. CustomValidator specific properties :- ClientValidationFunction, ValidateEmptyText, ServerValidate event
c. Eg: Perform server side validation
   Aspx File
   ```
   <asp:CustomValidator id="vldCode" runat="server" ErrorMessage="Try a string
   that starts with A." ValidateEmptyText="False"
   OnServerValidate="vldCode_ServerValidate" ControlToValidate="txtCode" />
   ```

   CS File
   ```
   protected void vldCode_ServerValidate(Object source, ServerValidateEventArgs
   e)
   {
   string value=e.Value;
   string firstchar = value.Substring(0,1);

   if(firstchar.ToLower() ! = ‘a’ ){
   e.IsValid= false;
   }
   else{
   e.IsValid= true;
   }
   }
   ```
d. To perform client side validation , the e CustomValidator.ClientValidationFunction property can be used. A javascript function is assigned as a value to this property and it takes two arguments - ‘source’ and ‘arguments’.
e. Eg:
   Aspx file, in head tag or separate js file which is linked to current aspx file
   ```
   <script type="text/javascript">
   function  MyCustomValidation(objSource, objArgs) {
    // Get value.
   var value = objArgs.Value;
   var firstchar = value.substr(0,1);
   if(firstchar.toLowerCase() ! = ‘a’ ){
   objArgs.IsValid= false;
   }
   else{
   objArgs.IsValid= true;
   }
   }
   </script>

   //The Validation control
   <asp:CustomValidator id="vldCode" runat="server" ErrorMessage="Try a string
   that starts with 014." ControlToValidate="txtCode"
   ```

```
OnServerValidate="vldCode_ServerValidate"
ClientValidationFunction="MyCustomValidation" />
```

3. Explain the use CompareValidator with example.
   ANS:-
   a. Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that you specify.
   b. validation will automatically succeed if the input control is empty, because there is no value to validate.
   c. Its specific properties : ControlToCompare, Operator, Type, ValueToCompare.
   d. Eg:
      Aspx file
      ```
      <asp:TextBox id=”pwd” runat=”server” textmode=”password”/>
      <asp:TextBox id=”confirmpwd” runat=”server” textmode=”password”/>

      <asp:CompareValidator id=”vldPwd” runat=”server” ErrorMessage =”Passwords
      dont match” ControlToCompare=”pwd” ControlToValidate=”confirmpwd”/>
      ```

4. How can we throw our own exception in ASP.NET application?
   ANS:-
   a. To throw our own exceptions, we need to create an instance of the appropriate exception class and then use the ‘throw’ statement.
   b. Throwing an exception is most useful in component-based programming
   c. In component-based programming, your ASP.NET page creates objects and calling methods from a class defined in a separately compiled assembly.
   d. In this case, the class in the component needs to be able to notify the calling code (the web application) of any errors.
   e. The component should handle recoverable errors quietly and not pass them up to the calling code.
   f. On the other hand, if an unrecoverable error occurs, it should always be indicated with an exception and never through another mechanism (such as a return code)
   g. If you need to return additional or specialized information, you can create your own custom exception class.

h. Custom exception classes should always inherit from System.ApplicationException, which itself derives from the base Exception class
i. When you create an exception class, you can add properties to record additional information.
j. Eg:

i. Throw a ASP.NET native exception

```
public class Example{
public static void Main(string[] args){
 int a =10,b=0;
try {
Console.WriteLine("a /b is:{0}" , a/b);
}
catch(DivideByZeroException err){
Console.WriteLine("Cant divide by zero. The message is: {0}" ,err.Message);
}
}
}
```

ii. Custom exception

```
using System;
public class CustomException : ApplicationException {
private decimal num1;
public decimal Num1_Prop {
get { return num1; }
set { num1= value; }
 }
public CustomException() : base() {}
public CustomException(string message) : base(message) {}
public CustomException(string message, Exception inner) : base(message,
inner) {}
}
public class Example{
public static void Main(string[] args){
 int a =10,b=0;
try {
if(b==0){
CustomException err =new CustomException();
err.Num1_Prop=b;
throw err;
}
 else{
Console.WriteLine("a /b is:{0}" , a/b);
}
}
catch(CustomException err){
Console.WriteLine("Cant divide by zero. Denominator is:{0}",err.Num1_Prop);
}
}
}
```

5. Short note on ViewState state management and Query strings state management.
   ANS:-
   <u>SN on ViewState</u>
   a. One of the most common ways to store information is in 'view state'.
   b. View state uses a hidden field that ASP.NET automatically inserts in the final, rendered HTML of a web page.
   c. It's a perfect place to store information that's used for multiple postbacks in a single web page.
   d. View state isn't limited to web controls.
   e. Your web page code can add bits of information directly to the view state of the containing page and retrieve it later after the page is posted back.
   f. The type of information you can store includes simple data types and your own custom objects.
   g. The ViewState property of the page provides the current view-state information.
   h. This property provides an instance of the StateBag collection class.
   i. The StateBag is a dictionary collection, which means every item is stored in a separate "slot" using a unique string name, which is also called the key name.
   j. Eg:
   ```
   public partial class SimpleCounter : System.Web.UI.Page {
   protected void cmdIncrement_Click(object sender, EventArgs e) {
    int counter;
    if (ViewState["Counter"] == null) {
   counter = 1;
    }
    else {
    counter = (int)ViewState["Counter"] + 1;
   }
   ViewState["Counter"] = counter;
   lblCount.Text = "Counter: " + counter.ToString();
   }
   }
   ```
   k. <u>SN on Query string state Management</u>
      i. Another approach is to pass information from one webpage to another is by using a query string in the URL.
      ii. This approach is commonly found in search engines.
      iii. The query string is the portion of the URL after the question mark.

iv. The advantage of the query string is that it's lightweight and doesn't exert any kind of burden on the server.

v. Disadvantages:
   1. Information is limited to simple strings, which must contain URL-legal characters.
   2. Information is clearly visible to the user and to anyone else who cares to eavesdrop on the Internet.
   3. The enterprising user might decide to modify the query string and supply new values, which your program won't expect and can't protect against.
   4. Many browsers impose a limit on the length of a URL (usually from 1 KB to 2 KB). Therefore, you can't place a large amount of information in the query string and still be assured of compatibility with most browsers.

vi. Eg:
It's particularly well suited in database applications in which you present the user with a list of items that correspond to records in a database, such as products.
When the user clicks on a product , they are redirected to a separate page which displays the specific details of the product.
This is implemented by sending the product_id from page1 to page2 via query strings. This product_id is then used by page2 to fetch the details from the database and display the details

vii. Eg:
On the sending page,
```
// Go to newpage.aspx.
//Submit two query string arguments: // recordID (10) and mode (full).
// multiple arguments are separated by '&'
Response.Redirect("newpage.aspx?recordID=10&mode=full");
```

On the receiving page,
```
 string ID = Request.QueryString["recordID"];
```

6. What is state management? Why state management is implemented explicitly in ASP.NET?
ANS:-
   a. State management is concerned with how the information is stored over the lifetime of the application.
   b. This information can be a simple string or a complex object.
   c. Because the web applications follow a stateless design, meaning when the client requests for a web page from a server , the page is delivered and the connection is severed and all the page objects are discarded from memory. By the time the user receives a page, the web page code has already stopped running, and there's no information left in the web server's memory.

d. This stateless design allows web applications to handle a large number of simultaneous requests.
e. Hence to retain information for a longer period of time so that it can be used over multiple postbacks or on multiple pages , state management is explicitly implemented in ASP.NET.
f. The various ways to implement state management in ASP.NET are:-
    i. View state and Cross page posting
    ii. Query string
    iii. Cookies
    iv. Sessions
    v. Application state
g. Advantages of explicitly managing the state are:-
    i. Flexibility
    ii. Scalability
    iii. Security
    iv. Performance
    v. State isolation

7. What is theme? List the advantages of using theme in a website.
   ANS:-
   a. Themes allow you to define a set of style details that you can apply to controls on multiple pages. However, unlike CSS, themes aren't implemented by the browser but rather ASP.NET processes your themes when it creates the page.
   b. Advantages are:-
      i. Consistency: Themes allow you to maintain a consistent look and feel across your entire website or application.
      ii. Reusability: Themes enable you to reuse style definitions, layouts, and resources (such as images and fonts) throughout your website
      iii. Easy Maintenance: With themes, you can make design changes or updates in one central location
      iv. Scalability: Themes are well-suited for large websites or applications. As your site grows, you can continue to maintain a consistent design by adding new pages or components that inherit styles from the theme.
      v. Separation of Concerns: Themes promote the separation of design concerns from content and functionality. Developers can focus on building functionality while designers can work on the visual aspects, leading to cleaner and more maintainable code.
      vi. Quick Updates: If you need to change the overall design of your website, you can do so by modifying the theme, and the changes will be applied automatically to all pages that use that theme.

8. List and explain the process of creating and using CSS using Visual Developer 2010
   ANS:-

a. To create a style sheet in Visual Studio, choose Website➤Add New Item from the Visual Studio menu.
b. Then, pick the Style Sheet template, specify a file name (like StyleSheet.css), and click Add.
c. In a style sheet, several styles are defined (also known as rules) that are used to format ordinary HTML and ASP.NET controls.
d. Eg:
body { font-family: Verdana, Arial, Sans-Serif; font-size: small; }
.heading1 { font-weight: bold; font-size: large; color: lime; }
e. To apply the styles to a web page, the CSS file needs to be linked to the webpage . This can be done by referencing the CSS file in the <link> element in the <head> section of the page.
f. Once this is done, the ASP.NET controls have the CssClass property to which the style rule is assigned. This can be also done from the properties window. The styles can be applied to ordinary HTML elements using the class attribute.
g. Eg:
<asp:Label ID="Label1" runat="server" Text="This Label Uses heading1" CssClass="heading1" ></asp:Label>
<div class=”heading1" id="paragraph" runat="server"></div>
h. The Apply Styles Window can be used to do the above.
i. The New Style dialog box can be used to add new styles to an existing CSS file.

9. What is master page? Explain the advantages of using master page
ANS:-
a. Master pages are like ordinary pages that contain HTML,web controls, code,etc .
b. But, they have a different file extension .master and cant be viewed directly by the browser.
c. Master pages are used by other pages which are known as content pages.
d. The master page defines the page structure and common ingredients and content pages adopt these structures and fill in the appropriate content.
e. Advantages:
    i. Consistency: Master pages allow one to define a consistent layout and design for your entire website or a specific section of it. Elements like headers, footers, navigation menus, and sidebars can be defined once in the master page and applied consistently to all content pages.
    ii. Centralized Design: Design elements and structural components are centralized in the master page. This simplifies design updates and changes because modifications to the master page automatically apply to all content pages that use it.
    iii. Code Reusability: The code-behind logic associated with a master page (e.g., C# or VB.NET code) can be reused across multiple content pages.
    iv. Ease of Maintenance: Because design and layout are isolated in the master page, maintenance tasks become more straightforward. The

updates need to be done at a single place which reduces the risk of inconsistencies.

v. Separation of Concerns:Developers can focus on building the specific features of each content page without worrying about the overall structure and layout.

f. Eg:

Master page

```
<%@ Master Language="C#" AutoEventWireup ="true"
CodeFile="SiteTemplate.master.cs" Inherits="SiteTemplate" %>
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
</asp:ContentPlaceHolder>
<i>This is a simple footer.</i>
</form>
</body>
</html>
```

Content Page

```
<%@ Page Language="C#" MasterPageFile ="~/SiteTemplate.master"
AutoEventWireup="true" CodeFile="SimpleContentPage.aspx.cs"
Inherits="SimpleContentPage" Title="Content Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
runat="Server">
Here's some new content
</asp:Content>
```

10. List and explain common properties of web server controls.

ANS:-(write any 8, the number 8 is chosen at random by me ,list all of them)

a. **AccessKey**:- Specifies the keyboard shortcut as one letter.
b. **BackColor, ForeColor, and BorderColor**:- Sets the colors used for the background, foreground, and border of the control. In most controls, the foreground color sets the text color.
c. **BorderWidth**:-Specifies the size of the control border.
d. **BorderStyle**:-One of the values from the BorderStyle enumeration, including Dashed, Dotted, Double, Groove, Ridge, Inset, Outset, Solid, and None.
e. **Controls**:- Provides a collection of all the controls contained inside the current control. Each object is provided as a generic System.Web.UI.Control object, so you will need to cast the reference to access control-specific properties.
f. **Enabled**:- When set to false, the control will be visible, but it will not be able to receive user input or focus.

g. **EnableViewState**:- Set this to false to disable the automatic state management for this control. In this case, the control will be reset to the properties and formatting specified in the control tag (in the .aspx page) every time the page is posted back. If this is set to true (the default), the control uses the hidden input field to store information about its properties, ensuring that any changes you make in code are remembered.
h. **Font** :- Specifies the font used to render any text in the control as a System.Web. UI.WebControls.FontInfo object.
i. **Height and Width**:- Specifies the width and height of the control. For some controls, these properties will be ignored when used with older browsers.
j. **ID**:- Specifies the name that you use to interact with the control in your code (and also serves as the basis for the ID that's used to name the top-level element in the rendered HTML).
k. **Page**:- Provides a reference to the web page that contains this control as a System.Web. UI.Page object.
l. **Parent**:- Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.
m. **TabIndex**:- A number that allows you to control the tab order. The control with a TabIndex of 0 has the focus when the page first loads. Pressing Tab moves the user to the control with the next lowest TabIndex, provided it is enabled.
n. **ToolTip**:- Displays a text message when the user hovers the mouse above the control. Many older browsers don't support this property. Visible When set to false, the control will be hidden and will not be rendered to the final HTML page that is sent to the client.

11. Explain ASP.NET Application life cycle and ASP.NET page life cycle.

ANS:-

a. Application Life Cycle
    i. Application start - The life cycle of an ASP.NET application starts when a request is made by a user. During this time, there is a method called Application_start which is executed by the web server. Usually, in this method, all global variables are set to their default values.
    ii. Object creation- The next stage is the creation of the HttpContext, HttpRequest & HttpResponse by the web server. The HttpContext is just the container for the HttpRequest and HttpResponse objects. The HttpRequest object contains information about the current request, including cookies and browser information. The HttpResponse object contains the response that is sent to the client.
    iii. HttpApplication creation - This object is created by the web server. It is this object that is used to process each subsequent request sent to the application.

iv. Dispose - This event is called before the application instance is destroyed. During this time, one can use this method to manually release any unmanaged resources.
v. Application end - In this part, the application is finally unloaded from memory.

b. Page Life Cycle
When an ASP.Net page is called, it goes through a particular lifecycle. This is done before the response is sent to the user. There are series of steps which are followed for the processing of an ASP.Net page :-
i. Page Request:- This is when the page is first requested from the server. When the page is requested, the server checks if it is requested for the first time.
ii. Page Start – During this time, 2 objects, known as the Request and Response objects are created.
iii. Page Initialization – During this time, all the controls on a web page are initialized.
iv. Page Load – This is when the page is actually loaded with all the default values.
v. Validation – The validation set on the form is carried out.
vi. Postback event handling – This event is triggered if the same page is being loaded again. For eg:When a button is clicked or radio button option is selected.
vii. Page Rendering – All the information on the form is saved, and the result is sent to the user as a complete web page. (transformed from a set of objects to an HTML page)
viii. Unload – Once the page output is sent to the user, all unwanted objects are removed from the memory.

OR

(below ans is given in textbook . It only describes the process when the autopostback is executed.I wont be writing the below answer.)

ix. On the client side, the JavaScript __doPostBack function is invoked, and the page is resubmitted to the server.
x. ASP.NET re-creates the Page object by using the .aspx file.
xi. ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
xii. The page life cycle begins, and the Page.Init and Page.Load events fire
xiii. All other control events fire.
xiv. The Page.PreRender event fires, and the page is rendered (transformed from a set of objects to an HTML page).
xv. Finally, the Page.Unload event is fired.
xvi. The new page is sent to the client.

12. List and explain various types of files which are used in an ASP.NET application.
    ANS:-
    a. Ends with .aspx: -
        i. These are ASP.NET web pages.
        ii. They contain the user interface and, optionally, the underlying application code.
        iii. Users request or navigate directly to one of these pages to start your web application.
    b. Ends with .ascx : -
        i. These are ASP.NET user controls.
        ii. User controls are similar to web pages, except that the user can't access these files directly.
        iii. Instead, they must be hosted inside an ASP.NET web page.
        iv. User controls allow you to develop a small piece of user interface and reuse it in as many web forms as you want without repetitive code
    c. web.config: -
        i. This is the configuration file for your ASP.NET application.
        ii. It includes settings for customizing security, state management, memory management, and much more.
    d. global.asax : -
        i. This is the global application file.
        ii. It is used to define global variables (variables that can be accessed from any web page in the web application) and react to global events (such as when a web application first starts).
    e. Ends with .cs :-
        i. These are code-behind files that contain C# code.
        ii. They allow you to separate the application logic from the user interface of a web page.
    f. In addition, your web application can contain other resources such as a directory that holds image files, HTML files CSS style sheets,etc.

13. Explain the TreeView control in ASP.NET.
    ANS:-
    a. The TreeView control in ASP.NET is a user interface control used to display hierarchical data in a tree-like structure, making it easier for users to navigate and interact with nested information.
    b. It is commonly used for purposes like representing folder structures, organization charts, displaying sitemaps or, any data with a parent-child relationship.
    c. Its attributes are:-
        i. NavigateUrl: Determines the URL to navigate to when a node is clicked
        ii. DataSource: Specifies the data source for the TreeView control, often bound to a data control like a SiteMapDataSource or XmlDataSource.

iii. DataTextField: Defines the field in the data source that provides the text for each tree node.
iv. DataValueField: Specifies the field in the data source that stores a unique identifier for each tree node.
v. ExpandDepth: Sets the initial depth to which the tree nodes are expanded when the TreeView is loaded.
vi. ShowLines: Indicates whether lines connecting the nodes are displayed to represent the hierarchy visually.
vii. ShowCheckBoxes: Determines whether checkboxes are displayed next to the nodes, enabling node selection.
viii. SelectedNodeStyle: Defines the style for the currently selected node.
ix. NodeStyle: Specifies the default style for the tree nodes.
x. LeafNodeStyle: Defines the style for nodes that have no child nodes.
xi. RootNodeStyle: Specifies the style for the root node(s) in the tree.
xii. OnTreeNodePopulate: Specifies the server-side event handler to populate child nodes dynamically.

14. State the advantages of Themes over CSS.

ANS:-

a. Integration with ASP.NET Controls: ASP.NET Themes are designed to work seamlessly with ASP.NET server controls. They allow you to define consistent styles for all server controls of a particular type across your application. This level of integration is not easily achievable with traditional CSS alone.
b. Control-Level Styling: With Themes, you can apply styles directly to individual server controls. This means you can define styles for specific controls without having to assign class names or IDs, making it easier to manage and apply styles at the control level.
c. Consistency and Reusability: Themes promote consistency in your web application's appearance by enabling you to define and reuse styles across multiple pages and controls. This consistency can be challenging to achieve with standalone CSS files.
d. Automatic Skinning: ASP.NET Themes can automatically adapt to different devices and user preferences, such as mobile devices, by using the built-in device adapter framework. This can simplify the process of creating responsive web designs.
e. Theme Inheritance: Themes support inheritance, allowing you to create a hierarchy of themes. This means you can define a common set of styles in a base theme and then create child themes that inherit those styles and customize them as needed. This can save development time and maintain a consistent look and feel.
f. Server-Side Control Over Styles: ASP.NET Themes provide the flexibility to change styles on the server side based on dynamic conditions or user

preferences. This level of control is not readily achievable with CSS alone, which is primarily a client-side technology.

g. Easy Deployment: Themes are easy to deploy in ASP.NET applications. They can be packaged into separate theme folders, and you can switch themes dynamically in your application's configuration without modifying individual pages.
h. Resource Management: Themes help in organizing and managing resources such as images, skins, and style sheets, making it easier to locate and maintain these resources.
i. Tool Support: Visual Studio and other ASP.NET development tools provide built-in support for creating and managing Themes, making it more convenient for developers to work with Themes.
j. Built-In Skins: ASP.NET Themes often come with built-in skins for common controls, allowing you to easily apply predefined styles to controls without writing custom CSS.

15. Explain multiple catch statements in exception handling. Also write the significance of finally block.
ANS:-
    a. Structured exception handling is flexible because it allows catching specific types of exceptions.
    b. To do so, multiple catch statements can be used, each for handling the specific exceptions.
    c. An exception will be caught as long as it’s an instance of the indicated class or if it’s derived from that class.
    d. Every exception is derived from the System.Exception base class.
    e. Similar to conditional code, as soon as a matching exception handler is found, the appropriate catch code is invoked.
    f. Therefore ‘catch statements’ must be organized from most specific to least specific.
    g. Eg:

```
try {
// Risky code goes here.
}
catch (System.Data.DataException err) {
 // System.Data class exceptions}
}
catch (System.IndexOutOfRangeException err) {
 // System class exceptions
 }
catch (System.IO.IOException err) {
 // System.IO class exceptions
 }
 catch (System.Exception err) {
```

```
// Catches any other errors.
 }
```

h. Finally:
    i. the ‘finally’ block of the code will be executed last, regardless of whether a bug occurs or not.
    ii. This allows you to perform some basic cleanup, such as closing a database connection.
    iii. The ‘finally’ code is important because it will execute even if an error has occurred that will prevent the program from continuing.
    iv. In other words, if an unrecoverable exception halts your application, you’ll still have the chance to release resources.
    v. Eg:

```
try {
// Risky code goes here (opening a file, connecting to a database, and so on).
}
catch {
// An error has been detected. You can deal with it here.
}
 finally {
// Time to clean up, regardless of whether or not there was an error.
 }
```