(2½ Hours)

[Total Marks: 75]

N. B.: (1) **All** questions are **compulsory**.
(2) Make **suitable assumptions** wherever necessary and **state the assumptions** made.
(3) Answers to the **same question** must be **written together**.
(4) Numbers to the **right** indicate **marks**.
(5) Draw **neat labeled diagrams** wherever **necessary**.
(6) Use of **Non-programmable** calculators is **allowed**.

1.	**Attempt *any three* of the following:**	**15**
a.	**Explain foreach loop with suitable example.** Answer: C# also provides a foreach loop that allows you to loop through the items in a set of data. With a foreach loop, you don't need to create an explicit counter variable. Instead, you create a variable that represents the type of data for which you're looking. Your code will then loop until you've had a chance to process each piece of data in the set. The foreach loop is particularly useful for traversing the data in collections and arrays. **Example:** string[] stringArray = {"one", "two", "three"}; foreach (string element in stringArray) { // This code loops three times, with the element variable set to // "one", then "two", and then "three". System.Diagnostics.Debug.Write(element + " "); } In this case, the foreach loop examines each item in the array and tries to convert it to a string. Thus, the foreach loop defifines a string variable named element. If you used a different data type, you'd receive an error. The foreach loop has one key limitation: it's read-only. For example, if you wanted to loop through an array and change the values in that array at the same time, foreach code wouldn't work. Here's an example of some flflawed code: int[] intArray = {1,2,3}; foreach (int num in intArray)	

{

num += 1;

}

In this case, you would need to fall back on a basic for loop with a counter.

b. **Distinguish between interface and abstract classes.**

Answer:

Following are the important differences between Abstract Class and Interface.

Sr. No.	Key	Abstract Class	Interface
1	Definition	In terms of standard definition an Abstract class is, conceptually, a class that cannot be instantiated and is usually implemented as a class that has one or more pure virtual (abstract) functions.	On other hand a a description of functions must inherits this inte implement. In o interface descril of the class.
2	Implementation	As like of other general class design in C# Abstract class also have its own implementation along with its declaration.	On other hand a can only have a the implementa implementation provided by the implements it.
3	Inheritance	As per specification in C# a class can extends only one other class hence multiple inheritance is not achieved by abstract class.	On other hand i Interface a class implements mu interfaces and h inheritance is ac interface.
4	Constructor	Like other classes in C# for instantiation abstract class also have constructor which provide an instance of abstract class to access its non-static methods.	On other hand I not have constru can't instantiate directly althoug could get access

			creating instance of class which implementing it.
5	Modifiers	As abstract class is most like of other ordinary class in C# so it can contain different types of access modifiers like public, private, protected etc.	On other hand as Interface needs to be get implemented in order to provide its methods implementation by other class so can only contains public access modifier.
6	Performanc e	As abstract class have its method as well as their implementations also for its abstract methods implementation it have reference for its implementing class so performance is comparatively faster as compare to that of Interface.	On other hand the performance of interface is slow because it requires time to search actual method in the corresponding class.

c. **What is namespace? Describe the System namespace.**

Answer:
A namespace is designed for providing a way to keep one set of names separate from another. The class names declared in one namespace does not conflict with the same class names declared in another.

Defining a Namespace A namespace definition begins with the keyword namespace followed by the namespace name as follows:

```
namespace namespace_name
{
// code declarations
}
```

Example:

```
using System;
namespace first_space
{
class namespace_cl
{
public void func()
{
Console.WriteLine("Inside first_space");
}
}
}
namespace second_space
{
class namespace_cl
```

```
{
public void func()
{
Console.WriteLine("Inside second_space");
}
}
}
class TestClass
{
static void Main(string[] args)
{
first_space.namespace_cl fc = new first_space.namespace_cl();
second_space.namespace_cl sc = new second_space.namespace_cl();
fc.func();
sc.func();
Console.ReadKey();
}
}
```

The using Keyword The using keyword states that the program is using the names in the given namespace. For example, we are using the System namespace in our programs. The class Console is defined there. We just write:

```
Console.WriteLine ("Hello there");
```

We could have written the fully qualified name as:

```
System.Console.WriteLine("Hello there");
```

Alias of Namespace:

```
using A=System.Console;
class Test
{
static void Main()
{
A.Write("Craetion of Alias");
A.ReadKey();
}
}
```

d. **What is an assembly? Explain the difference between public and private assembly.**

Answer:

Assembly:

An assembly in ASP.NET is a collection of single-file or multiple files. The assembly that has more than one file contains either a dynamic link library (DLL) or an EXE file. The assembly also contains metadata that is known as assembly manifest.

The assembly manifest contains data about the versioning requirements of the assembly, author name of the assembly, the security requirements that the assembly requires to run, and the various files that form part of the assembly.

	The biggest advantage of using ASP.NET Assemblies is that developers can create applications without interfering with other applications on the system. When the developer creates an application that requires an assembly that assembly will not affect other applications. The assembly used for one application is not applied to another application. However, one assembly can be shared with other applications. In this case the assembly has to be placed in the bin directory of the application that uses it. This is in contrast to DLL in the past. Earlier developers used to share libraries of code through DLL. To use the DLL that is developed by another developer for another application, we must register that DLL in our machine. In ASP.NET, the assembly is created by default whenever we build a DLL. We can check the details of the manifest of the assembly by using classes located in the System.Reflection namespace. Thus, we can create two types of ASP.NET Assemblies in ASP.NET: private ASP.NET Assemblies and shared assemblies. **Private Assembly:** Private ASP.NET Assemblies are created when you build component files like DLLs that can be applied to one application. **Public Assembly:** Shared ASP.NET Assemblies are created when you want to share the component files across multiple applications. Shared ASP.NET Assemblies must have a unique name and must be placed in Global Assembly Cache (GAC). The GAC is located in the Assembly directory in WinNT. You can view both the manifest and the IL using ILDisassembler (ildasm.exe).	
e.	**Write a sample C# program to demonstrate class, object and method call. Use comments wherever require.** **Answer:** using System; class SampleClass { **//Method declared outside the main.** void show() { int x = 100; int y = 200; Console.WriteLine(x); Console.WriteLine(y); }	

```
  public static void Main()
  {
    //Object created
    SampleClass a = new SampleClass ();

    //Instance method called
    a.show();
  }
}
```

f. **Define the accessibility modifiers-public, private, protected, internal, and protected internal.**

Answer:

public: The type or member can be accessed by any other code in the same assembly or another assembly that references it. The accessibility level of public members of a type is controlled by the accessibility level of the type itself.

private: The type or member can be accessed only by code in the same class or struct.

protected: The type or member can be accessed only by code in the same class, or in a class that is derived from that class.

internal: The type or member can be accessed by any code in the same assembly, but not from another assembly. In other words, internal types or members can be accessed from code that is part of the same compilation.

protected internal: The type or member can be accessed by any code in the assembly in which it's declared, or from within a derived class in another assembly.

private protected: The type or member can be accessed by types derived from the class that are declared within its containing assembly.

Summary table:

Summary:

Caller's location	**public**	**protected internal**	**protected**	**internal**	**private protected**	**private**
Within the class	YES	YES	YES	YES	YES	YES
Derived class (same assembly)	YES	YES	YES	YES	YES	NO
Non-derived class (same assembly)	YES	YES	NO	YES	NO	NO
Derived class (different assembly)	YES	YES	YES	NO	NO	NO

Non-derived class (different assembly)	YES	NO	NO	NO	NO	NO	

2.	**Attempt *any three* of the following:**	**15**
a.	**What is postback? Explain IsPostBack property with suitable example.**	

Answer:

Postback:

In an ASP.NET web page, there's at least one more element. Inside the<body>element is a<form>element.

The<form>element is required because it defifines a portion of the page that can send information back to the web server. This becomes important when you start adding text boxes, lists, and other controls. As long as they're in a form, information such as the current text in the text box and the current selection in the list will be sent to the web server by using a process known as a postback.

Example:

```
private void Page_Load()
{
   if (!IsPostBack)
   {
```

	// Validate() method will be called if page is loaded first time from server to the client. Validate(); } }	
b.	**List and describe the various file types used in an ASP.NET application.** **Answer:** **.aspx** An ASP.NET Web forms file (page) that can contain Web controls and presentation and business logic. **.cs** Class source-code file that is compiled at run time. The class can be an HTTP Module, an HTTP Handler, a code-behind file for an ASP.NET page, or a stand-alone class file containing application logic. **.asax** Typically a Global.asax file that contains code that derives from the HttpApplication class. This file represents the application and contains optional methods that run at the start or end of the application lifetime. **.ascx** A Web user control file that defines a custom, reusable control. **.config** A configuration file (typically Web.config) containing XML elements that represent settings for ASP.NET features. **.master** A master page that defines the layout for other Web pages in the application. **.sitemap** A site-map file that contains the structure of the Web site. ASP.NET comes with a default site-map provider that uses site-map files to easily display a navigational control in a Web page. **.skin** A skin file containing property settings to apply to Web controls for consistent formatting.	
c.	**What is an event? How is an event handler added?** **Answer:** **Event:** An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification.	

Adding Event Handlers

Most of the code in an ASP.NET web page is placed inside event handlers that react to web control events. Using

Visual Studio, you have three easy ways to add an event handler to your code:

Type it in manually: In this case, you add the subroutine directly to the page class in your C# code file. You must specify the appropriate parameters.

Double-click a control in design view: In this case, Visual Studio will create an event handler for that control's default event, if it doesn't already exist. For example, if you

double-click a Button control, it will create an event handler for the Button.Click event.

If you double-click a TextBox control, you'll get an event handler for the TextBox.TextChanged event. If the event handler already exists, Visual Studio simply takes you to the relevant place in your code.

Choose the event from the Properties window: Just select the control, and click the lightning bolt in the Properties window. You'll see a list of all the events provided by that control. Double-click next to the event you want to handle, and Visual Studio will automatically generate the event handler in your page class. Alternatively, if you've already created the event handler method, just select the event in the Properties window, and click the drop-down arrow at the right. You'll see a list that includes all the methods in your class that match the signature this event requires. You can then choose a method from the list to connect it. Figure 4-13 shows an example where the Button.Click event is connected to the Button1_Click method in the page class.

d. **Write a short note on List controls in ASP.NET.**

Answer:

List Controls

The list controls include the ListBox, DropDownList, CheckBoxList, RadioButtonList, and BulletedList. They all work in essentially the same way but are rendered differently in the browser. The ListBox, for example, is a rectangular list that displays several entries, while the DropDownList shows only the selected item. The CheckBoxList and RadioButtonList are similar to the ListBox, but every item is rendered as a check box or option button, respectively. Finally, the BulletedList is the odd one out—it's the only list control that isn't selectable.

Instead, it renders itself as a sequence of numbered or bulleted items.

All the selectable list controls provide a SelectedIndex property that indicates the selected row as a zero based index (just like the HtmlSelect control you used in the previous chapter). For example, if the first item in the list is selected, the SelectedIndex will be 0. Selectable list controls also provide an additional SelectedItem property, which allows your code to retrieve the ListItem object that represents the selected item.

	The ListItem object provides three important properties: Text (the displayed content), Value (the hidden value from the HTML markup), and Selected (true or false depending on whether the item is selected). In the previous chapter, you used code like this to retrieve the selected ListItem object from an HtmlSelect control called Currency, as follows: ListItem item; item = Currency.Items[Currency.SelectedIndex]; If you used the ListBox web control, you can simplify this code with a clearer syntax: ListItem item; item = Currency.SelectedItem;	
e.	**Explain the need of user control. How it is created and used?** **Answer:** The web user controls are containers that can be created by combining one or more web server controls. After creating a Web user control, you can treat it as a unit and define properties and methods for it. They are similar to the ASP.NET web pages in the context that they contain both a user interface page and code. □ **The file name extension of the user control is .ascx** A user control contains the @Control directive instead of the @Page directive They cannot run as stand alone files. They need to be added to the ASP.NET pages to make them work. User controls do not have <html>, <body>, or <form> elements. The elements must be present on the web page that is hosting these controls. A web user control on a web page must be registered before it is used using @Register directive. **Step1:** Open Visual Studio, Right click a website, select Add New Item,->Web User Control and give it a new name example: myControl.ascx <%@ Control Language="C#" AutoEventWireup="true" CodeFile="myControl.ascx.cs" Inherits="myControl" %> <asp:Button ID="tbShow" runat="server" Font-Bold="True" Font-Size="Larger" onclick="tbShow_Click" Text="Show" /> <asp:Label ID="lblMessage" runat="server" Font-Bold="True" Font-Size="Larger"></asp:Label> myControl.ascx.cs public partial class myControl : System.Web.UI.UserControl { protected void tbShow_Click(object sender, EventArgs e)	

```
    {
        lblMessage.Text = "Hello from User Control";
    }
  }
```

Step2:

Include User Control to web form**:**

After creating User Control we have to include that to our web form. So here we have to crate @Register directive that includes following attributes,

A TagPrefix attribute, which associates a prefix with the user control. This prefix will be included in opening tag of the user control element.

A TagName attribute, which associates a name with the user control. This name will be included in the opening tag of the user control element.

A Src attribute, which defines the virtual path to the user control file that you are including.

UseUserControl.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="UserControlUseEx.aspx.cs" Inherits="UserControlUseEx" %>
<%@ Register TagPrefix="Jeet" TagName="Message" Src="~/myControl.ascx" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>

  <Jeet:Message ID="firstUserControl" runat="server" />
  </div>
  </form>
</body>
</html>
```

f. **What is the purpose of validation controls? List and explain the use of validation controls available in ASP.NET.**

Answer:

Validation is important part of any web application. User's input must always be validated before
sending across different layers of the application.

Validation controls are used to:

- Implement presentation logic.
- To validate user input data.
- Data format, data type and data range is used for validation.
- Validation Controls in ASP.NET

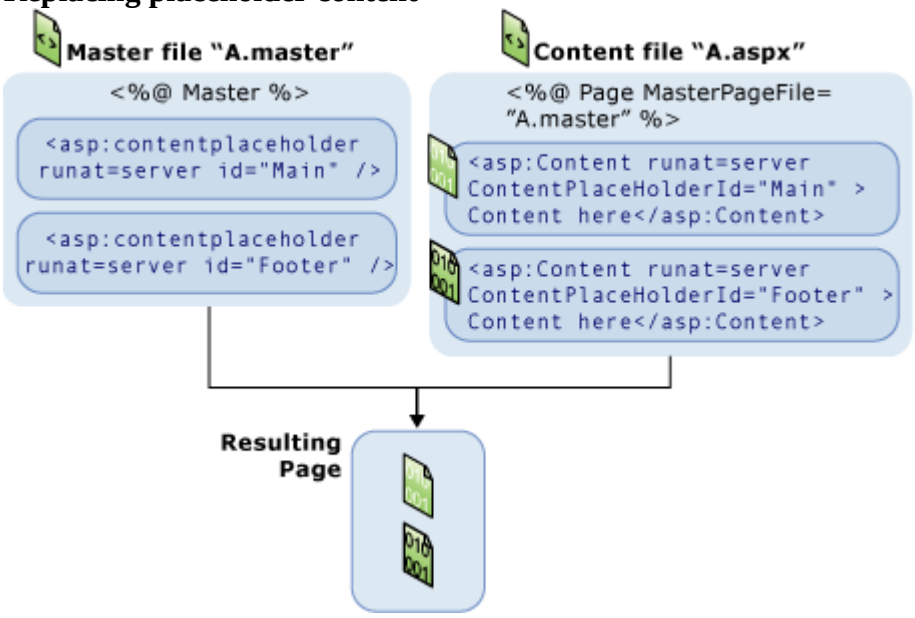
Validator Controls

Control Class	Description
RequiredFieldValidator	Validation succeeds as long as the input control doesn't contain an empty string.
RangeValidator	Validation succeeds if the input control contains a value within a specific numeric, alphabetic, or date range.
CompareValidator	Validation succeeds if the input control contains a value that matches the value in another input control, or a fixed value that you specify.
RegularExpressionValidator	Validation succeeds if the value in an input control matches a specified regular expression.
CustomValidator	Validation is performed by a user-defined function.

3.	**Attempt *any three* of the following:**	**15**
a.	**Describe the use of multiple catch statements in exception handling using example.** **Answer:**	
b.	**What is QueryString? How to send a name and marks of a student from one web page to another web page using QueryString?** **Answer:** **QueryString:** Query String is the most simple and efficient way of maintaining information across requests. The information we want to maintain will be sent along with the URL. A typical URL with a query string looks like **www.somewebsite.com/search.aspx?query=value1** The URL part which comes after the ? symbol is called a QueryString. QueryString has two parts, a key and a value. In the above example, query is the key and value is its value. **Example:** We can send multiple values through querystring, separated by the & symbol. The following code shows sending multiple values to the foo.aspx page. **Response.Redirect("foo.aspx?name=john&marks=87");** **The following code shows reading the QueryString values in Page2.aspx** string name = Request.QueryString["name"];	

	string marks = Request.QueryString["marks"];	
c.	**Explain the events in global.asax file with respect to state management.** **Answer:** Global.asax file contains the following events Application Events Application.EndRequest is only one of more than a dozen events you can respond to in your code. To create a different event handler, you simply need to create a subroutine with the defined name. *Basic Application Events* **Event-Handling Method** — **Description** Application_Start() — Occurs when the application starts, which is the first time it receives a request from any user. It doesn't occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later. Application_End() — Occurs when the application is shutting down, generally because the web server is being restarted. You can insert cleanup code here. Application_BeginRequest() — Occurs with each request the application receives, just before the page code is executed. Application_EndRequest() — Occurs with each request the application receives, just after the page code is executed. Session_Start() — Occurs whenever a new user request is received and a session is started. Session_End() — Occurs when a session times out or is programmatically ended. This event is raised only if you are using in-process session-state storage (the InProc mode, not the StateServer or SQLServer modes). Application_Error() — Occurs in response to an unhandled error.	
d.	**How the connection between the content page and the master page is established?** **Answer:** **Content Pages** You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages (.aspx files and, optionally, code-behind files) that are bound to a specific master page. The binding is established in the content page's **@ Page** directive by including a MasterPageFile attribute that points to the master page to be used. For example, a content page might have the following **@ Page** directive, which binds it to the Master1.master page. <%@ Page Language="C#" MasterPageFile="~/MasterPages/Master1.master" Title="Content Page"%> In the content page, you create the content by adding Content controls and mapping them to ContentPlaceHolder controls on the master page**.** **For example**, the master page might have content placeholders called Main and Footer. In the content page, you can create two Content controls, one that is mapped to the ContentPlaceHolder control Main and the other mapped to the ContentPlaceHolder control Footer, as shown in the following figure.	

Replacing placeholder content



A content page might look like the following.

```
<% @ Page Language="C#" MasterPageFile="~/Master.master" Title="Content Page 1" %>
<asp:Content ID="Content1" ContentPlaceHolderID="Main" Runat="Server">
   Main content.
</asp:Content>

<asp:Content ID="Content2" ContentPlaceHolderID="Footer" Runat="Server" >
   Footer content.
</asp:content>
```

e.

What is the theme? Explain how to create and use a theme on a website.

Answer:

Theme:

A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server.

To create a page theme

1. In Solution Explorer, right-click the name of the Web site for which you want to create a page theme, and then click Add ASP.NET Folder.
2. Click Theme.
3. If the App_Themes folder does not already exist, Visual Web Developer creates it. Visual Web Developer then creates a new folder for the theme as a child folder of the App_Themes folder.
4. Type a name for the new folder.

	5. The name of this folder is also the name of the page theme. For example, if you create a folder named \App_Themes\FirstTheme, the name of your theme is FirstTheme. **Add files to your new folder for control skins, style sheets, and images that make up the theme.** **To add a skin file and a skin to a page theme** 1. In Solution Explorer, right-click the name of your theme and then click Add New Item. 2. In the Add New Item dialog box, click Skin File. 3. In the Name box, type a name for the .skin file, and then click Add. 4. The typical convention is to create one .skin file per control, such as Button.skin or Calendar.skin. However, you can create as many or as few .skin files as you need. 5. In the .skin file, add a normal control definition by using declarative syntax, but include only the properties that you want to set for the theme. The control definition must include the runat="server" attribute, and it must not include the ID="" attribute. **The following code example shows a default control skin for a Button control, defining the color and font for all of the Button controls in the theme.** <asp:Button runat="server" BackColor="Red" ForeColor="White" Font-Name="Arial" Font-Size="9px" />	
f.	**What is URL mapping? How is URL mapping and routing implemented in ASP.NET?** **Answer:** URL Mapping-2 Marks Steps/Example-3 Marks	
4.	**Attempt *any three* of the following:**	**15**
a	**Describe the SqlConnection class with an example.** **Answer:** **Connection** To interact with a database, we must have a connection to it. The connection helps identify the database server, the database name, user name, password, and other parameters that are required for connecting to the data base. A connection object is used by command objects so they will know which database to execute the command on.	

Connection Object is used for connecting your application to data source or database. It carries required authentic information like username and password in the connection string and opens a connection. You need to different type of connection object for different type of data providers. For example:
OLE DB – OleDbConnection
SQL Server – SqlConnection
ODBC – OdbcConnection
Oracle – OracleConnection

Example:
SqlConnection conn = new SqlConnection ("Data Source =(local);Initial Catalog=WorkingDatabase;Integrated Security=SSPI";

Parameter Name	**Description**
Data Source	This identifies the server as local, a domain name, or an IP address.
Initial Catalog	This specifies the database by name.
Integrated Security	When set to SSPI, this connects with a user's Windows login.
User ID	This provides the username for SQL Server.
Password	This provides the password associated with the SQL Server username.

b **Differentiate between DataSet and DataReader.**

Answer:

DataSet	DataReader
The DataSet class in ADO.Net operates in an entirely disconnected nature.	DataReader is a connection oriented service.
DataSet is an in-memory representation of a collection of Database objects including related tables, constraints, and relationships among the tables.	DataReader is designed to retrieve a read-only, forward-only stream of data from data sources.
It fetches entire table or tables at a time so greater network cost.	It fetches one row at a time so very less network cos.
DataSet is not read-only so we can do any transaction on them.	DataReader is readonly so we can't do any transaction on them.
DataAdapter is used to get data in DataSet	DataAdapter is not required
Dataset works with the help of xml technology	DataReader doesn't provide this feature.
Example: Dataset ds=new Dataset ( ); DataAdapter1Fill(ds,"newtablename") // where newtablename is table alias name in dataset	**Example**: Sqlcommand cmd =new sqlcommand (select * from emptable); Data Reader dr= cmd.ExecuteReader ( )

c **Write c# code to insert data in database table from text boxes. Write comments wherever required.**

Answer:

```
using System.Configuration;
public partial class InsertDetails : System.Web.UI.Page
{
    SqlConnection con = new
SqlConnection(ConfigurationManager.ConnectionStrings["ConnectionString"].ConnectionString);
    protected void Page_Load(object sender, EventArgs e)
    {
        con.Open();
    }
    protected void btSubmit_Click(object sender, EventArgs e)
    {
        SqlCommand cmd=new SqlCommand ("INSERT INTO tbl
VALUES("+txtID.Text+",'"+txtFName.Text+"','"+txtLName.Text +"','"+txtCity.Text+"')",con);
        cmd.ExecuteNonQuery();
        lblMessage.Visible=true;
        lblMessage.Text="Your data has been store successfully";

        txtID.Text="";
        txtFName.Text="";
        txtLName.Text="";
        txtCity.Text="";

    }
}
```

d **What is use of data source control? Explain various types of data sources in ASP.NET.**
The Data source control connects to and retrieves data from a data source and makes it available for other controls to bind to, without requiring code. ASP.NET allows a variety of data sources such as a database, an XML file, or a middle-tier business object.

The common data source controls are:

- **AccessDataSource –** Enables you to work with a Microsoft Access database.
- **XmlDataSource –** Enables you to work with an XML file.

- **SqlDataSource –** Enables you to work with Microsoft SQL Server, OLE DB, ODBC, or Oracle databases.
- **ObjectDataSource –** Enables you to work with a business object or other class
- **SiteMapDataSource –** Used for ASP.NET site navigation.
- **EntityDataSource** - Enables you to bind to data that is based on the Entity Data Model.
- **LinqDataSource** – Enables you to use Language-Integrated Query (LINQ) in an ASP.NET Web page.

e **Write a code to display data from a table named Students(RollNo, Name, Marks) and display on grid view control when page is loaded.**

Answer:

```
SqlConnection conn;
SqlDataAdapter adapter;
DataSet ds;
SqlCommand cmd;
string cs = ConfigurationManager.ConnectionStrings["conString"].ConnectionString; protected
void PopulateDetailView()
   {
        try
        {
        conn = new SqlConnection(cs);
        adapter = new SqlDataAdapter("select * from tblEmps", conn);
        ds = new DataSet();
        adapter.Fill(ds); DetailsView1.DataSource = ds;      DetailsView1.DataBind();
        }
        catch (Exception ex)
        {
        Label1.Text = "ERROR :: " + ex.Message;
        }
   }
```

f **Describe (i) ExecuteNonQuery, (ii) ExecuteScalar, and (iii) ExecuteReader.**

Answer:

ExecuteScalar(): only returns the value from the first column of the first row of your query.Execute Scalar will return single row single column value i.e. single value, on execution of SQL Query or Stored procedure using command object. It's very fast to retrieve single values from database. Used to execute SQL Select command which is used to return a single value. ExecuteScalar only returns the value from the first column of the first row of your query.

Example: string result = (string)cmd.ExecuteScalar();

Where cmd-is an object of SqlCommand class.

ExecuteReader(): returns an object that can iterate over the entire result set.

ExecuteNonQuery():

ExecuteNonQuery method will return number of rows effected with INSERT, DELETE or UPDATE operations. This ExecuteNonQuery method will be used only for insert, update and delete, Create, and SET statements.

ExecuteNonQuerydoes not return data at all. It returns only the number of rows affected by an insert, update, or delete.

Example:

```
int result= cmd.ExecuteNonQuery();
```

Where cmd-is an object of SqlCommand class.

5. Attempt *any three* of the following: 15

a. **Write a code to write employee data as empId, empName, empDept, and empDesignation data from text boxes to an XML file.**

Answer:

Sample Code:

```
XmlTextWriter xWriter = new XmlTextWriter(Server.MapPath("EmployeeDetails.xml"),
Encoding.UTF8);
      xWriter.WriteStartDocument();
      //Create Parent element
      xWriter.WriteStartElement("EmployeeDetails");
      //Create Child elements
      xWriter.WriteStartElement("Details");
          xWriter.WriteElementString("Id: ", txtId.Text);
         xWriter.WriteElementString("Name: ", txtName.Text);
         xWriter.WriteElementString("Department: ", txtDept.Text);
         xWriter.WriteElementString("Designation: ", empDesignation.Text);
      xWriter.WriteEndElement();

      //End writing top element and XML document
      xWriter.WriteEndElement();
      xWriter.WriteEndDocument();
      xWriter.Close();
```

b. **What is XML? List and explain the various XML classes.**

Answer:

Extensible Markup Language (XML) stores and transports data. If we use a XML file to store the data then we can do operations with the XML file directly without using the database. The XML format is supported for all applications.

It is independent of all software applications and it is accessible by all applications. It is a very widely used format for exchanging data, mainly because it's easy readable for both humans and machines. If we have ever written a website in HTML, XML will look very familiar to us, as it's basically a stricter version of HTML. XML is made up of tags, attributes and values and looks something like this:

```
<?xmlversion="1.0"encoding="utf-8"?>
<EmployeeInformation>
<Details>
<Name>Richa</Name>
```

	<Emp_id>1</Emp_id> <Qualification>MCA</Qualification> </Details> </EmployeeInformation> **XML Classes:** ASP.NET provides a rich set of classes for XML manipulation in several namespaces that start with **System.Xml**. The classes here allow us to read and write XML files, manipulate XML data in memory, and even validate XML documents. The following options for dealing with XML data: **XmlTextWriter** The XmlTextWriter class allows us to write XML to a file. This class contains a number of methods and properties that will do a lot of the work for us. To use this class, we create a new XmlTextWriter object. **XmlTextReader** Reading the XML document in our code is just as easy with the corresponding XmlTextReader class. The XmlTextReader moves through our document from top to bottom, one node at a time. We call the **Read()** method to move to the next node. This method returns true if there are more nodes to read or false once it has read the final node. **XDocument** The XDocument class contains the information necessary for a valid XML document. This includes an XML declaration, processing instructions, and comments. The XDocument makes it easy to read and navigate XML content. We can use the static XDocument.Load() method to read XML documents from a file, URI, or stream.	
c.	**What do you mean by "authentication"? Describe its various types of authentication.** **Answer:** **Authentication:** Authentication is process of validating the identity of a user so the user can be granted access to an application. A user must typically supply a user name and password to be authenticated. After a user authenticated, the user must still be authorized to use the required application. The process of granting user access to an application is called authorization. **ASP.NET supports 3 types of authentication as follows:** ● Forms Authentication, ● Passport Authentication, and ● Windows authentication providers.	
d.	**Explain the use of UpdateProgress control in AJAX.** **Answer:**	

The UpdateProgress control

works in conjunction with the UpdatePanel. Essentially, the UpdateProgress control allows you to show a message while a time-consuming update is under way.

The markup for this page defines an UpdatePanel followed by an UpdateProgress:

```
<asp:UpdatePanel ID ="UpdatePanel1" runat="server">
   <ContentTemplate>
       <div style= "background-color:#FFFFE0;padding: 20px">
           <asp:Label ID = "lblTime" runat= "server" Font-Bold= "True" > </asp:Label>
          <br />< br />
         <asp:Button ID ="cmdRefreshTime" runat="server"   OnClick= "cmdRefreshTime_Click"
Text= "Start the Refresh Process" />
     </div>
 </ContentTemplate>

</asp:UpdatePanel>
    <br />
     <asp:UpdateProgress ID ="updateProgress1" runat ="server">
       <ProgressTemplate>
        <div style= "font-size: xx-small">
            Contacting Server … <img src="wait.gif" alt= "Waiting…" />
        </div>
     </ProgressTemplate>
</asp:UpdateProgress>
```

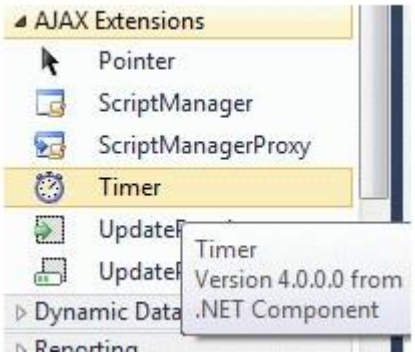
This isn't the only possible arrangement. Depending on the layout you want, you can place your UpdateProgress control somewhere inside your UpdatePanel control.
The code for this page has a slight modification from the earlier examples. Because the UpdateProgress control shows its content only while the asynchronous callback is under way, it makes sense to use the control only with an operation that takes time. Otherwise, the UpdateProgress will show its ProgressTemplate for only a few fractions of a second. To simulate a slow process, you can add a line to delay your code 10 seconds, as shown here:

```
protected void cmdRefreshTime_Click(object sender, EventArgs e)
{
 System.Threading.Thread.Sleep(TimeSpan.FromSeconds(10));
 lblTime.Text= DateTime.Now.ToLongTimeString();
}
```

e. **What is use of timer control? Write the steps with appropriate code to create an application to display real-time timing (clock) on an asp.net web page.**

Answer:

Timer controls allow us to do postbacks at certain intervals. If used together with UpdatePanel, which is the most common approach, it allows for timed partial updates of our page, but it can be used for posting back the entire page as well.
The Timer control uses the interval attribute to define the number of milliseconds to occur before firing the Tick event.



<asp:Timer ID="Timer1" runat="server" Interval="2000" OnTick="Timer1_Tick">

</asp:Timer>

Example:

Here is a small example of using the Timer control. It simply updates a timestamp every 5 seconds.

<asp:ScriptManager ID="ScriptManager1" runat="server" />

<asp:Timer runat="server" id="UpdateTimer" interval="5000" ontick="UpdateTimer_Tick"/>

<asp:UpdatePanel runat="server" id="TimedPanel" updatemode="Conditional">

<Triggers>

<asp:AsyncPostBackTrigger controlid="UpdateTimer" eventname="Tick" />

</Triggers>

<ContentTemplate>

<asp:Label runat="server" id="DateStampLabel" />

</ContentTemplate>

</asp:UpdatePanel>

We only have a single CodeBehind function, which we should add to our CodeBehind file:

protected void UpdateTimer_Tick(object sender, EventArgs e)

{

DateStampLabel.Text = DateTime.Now.ToString();

}

f. **What are the benefits using Ajax? Explain UpdatePanel and ScriptManager.**

Answer:

Benefits of AJAX

- Reduce the traffic travels between the client and the server.
- Response time is faster so increases performance and speed.
- You can use JSON (JavaScript Object Notation) which is alternative to XML. JSON is key value pair and works like an array.
- Ready Open source JavaScript libraries available for use – JQuery, etc..
- AJAX communicates over HTTP Protocol.

The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

The UpdatePanel Control:

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page.

For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back

Properties of the UpdatePanel Control:

The following table shows the properties of the update panel control:

Properties	Description
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls which will cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
ContentTemplateContainer	Retrieves the dynamically created template container object and used for adding child controls programmatically.
IsInPartialRendering	Indicates whether the panel is being updated as part of the partial post back.
RenderMode	Shows the render modes. The available modes are Block and Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control:

The following table shows the methods of the update panel control:

Methods	Description
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content