BACHELOR OF SCIENCE (INFORMATION TECHNOLOGY)
ADVANCED WEB PROGRAMMING
CBCGS(NOV - 2022)

Q1. a) Explain foreach loop with suitable example. (5)

Ans. A foreach loop mostly used with array and collection such as ArrayList. The foreach statement repeats a group of embedded statements for each element in an array or an object collection. You do not need to specify the loop bounds minimum or maximum. The foreach loop uses the 'in' keyword to iterate over the iterable item. The "in" keyword selects an item from the collection for the iteration and stores it in a variable called the loop variable, and the value of the loop variable changes in every iteration. For example, the first iteration process will have the first item of the collection stored; the second iteration will have the second item of the collection stored, and so on. A foreach loop enables us to address each element in an array using this simple syntax:

```
foreach (<baseType><name> in <array>)
{
// can use <name> for each element
}
```

This loop will cycle through each element, placing it in the variable <name> in turn, without danger of accessing illegal elements. We don't have to worry about how many elements are in the array, and we can be sure that we get to use each one in the loop. The main difference between using this method and a standard for loop is that foreach gives us read only access to the array contents, so we can't change the values of any of the elements.

Example :

```
// foreach loop program to illustrate the working of the loop
using System;
class loopworks
{
  static public void Main()
   {
     Console.WriteLine("Print array:");       //displays text on output window
     int[] arr_array = new int[] { 0, 1, 2, 3, 4, 5, };    // array creation

// foreach loop begins and runs till last item

foreach(int i in arr_array)
     {
       Console.WriteLine(i);
     }
   }
} // end of code
```

Output

```
Print array :
0
1
2
3
4
5
```

Q1. b) Distinguish between interface and abstract classes. (5)

Ans.

Aspect	Interface	Abstract Class
Definition	Collection of abstract members.	A class that cannot be instantiated directly.
Members	All members are implicitly abstract and public.	Can have both abstract and concrete members.
Inheritance	Multiple Inheritance supported.	Single inheritance supported.
Access Modifiers	Members are always public	Members can have various access modifiers

Constructors	Cannot have constructors	Can have constructors to initialize the class
Implementation	Classes implement interfaces.	Classes extend abstract class and provide implementations for abstract methods.
Purpose	Defines a contract that classes must follow.	Provides a common base for related classes.
Relationship	Promotes loose coupling.	Provides a hierarchical relationship.

Q1. c) What is namespace? Describe the system namespace. (5)

Ans. 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. It is not possible to use any access specifiers like private, public etc with a namespace declarations. The namespaces in C# implicitly have public access and this is not modifiable. The namespace elements can't be explicitly declared as private or protected. The namespace allows only public and internal elements as it members. The default is internal.

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 System namespace is the root namespace for fundamental types in . NET. This namespace includes classes that represent the base data types used by all applications, for example, Object (the root of the inheritance hierarchy), Byte, Char, Array, Int32, and String.

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();
 }
}
```

Q1. d) What is an assembly? Explain the difference between public and private assembly? (5)

Ans. 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. An assembly is a fundamental building block of any .NET framework application. It contains the code that is executed by common language runtime. For example, when we build a simple C# application, Visual Studio creates an assembly in the form of a single portable executable (PE) file, specifically an EXE or DLL.

Components of Assembly

1. Manifest

It describes the assembly. The manifest file contains all the metadata needed to specify the assembly's version requirements, security identity, and all metadata needed to define the scope of the assembly and resolve references to resources and classes.

2.Type Metadata

It contains metadata, which describes each type (class, structure, enumeration, and so forth)

3.MSIL

It contains Intermediate language code.

4.Resources

It contains bitmaps, icons, audios and other types of resources.

Public Assembly	Private Assembly
1.Public Assembly can be used by multiple applications.	1.Private assembly can be used by only one application.
2.Public assembly is stored in GAC (Global Assembly Cache.)	2.Private assembly will be stored in the specific application's directory or sub-directory.
3. Public assembly is also known as shared assembly.	3. There is no other name for private assembly.
4.Strong name has to be created for public assembly.	4.Strong name is not required for private assembly.

5.Public assembly should strictly enforce version constraint.	5.Private assembly doesn’t have any version constraint.
6.An example to public assembly is the actuate report classes which can be imported in the library and used by any application that prefers to implement actuate reports.	6.By default, all assemblies you create are examples of private assembly.Only when you associate a strong name to it and store it in GAC, it becomes public assembly.

Q1. e) Write a sample C# program to demonstrate class, object and method call. Use comments wherever required. (5)

Ans.

In C#,

Class: A blueprint or template that defines the structure and behavior of objects.

Object: An instance of a class that represents a real-world entity and holds its own set of data.

Method: A function defined within a class that operates on the data of its object.

Code :

```
using System;
class Car
{
   public string Make { get; set; }
   public string Model { get; set; }
   public void StartEngine()
   {
      Console.WriteLine("Engine started!");
   }
}
class Program
{
   static void Main()
   {
      // Creating an object of the Car class
```

```
        Car myCar = new Car();
        myCar.Make = "Toyota";
        myCar.Model = "Corolla";

        // Calling the StartEngine method of the Car class on the object
        myCar.StartEngine();

        // Creating another object of the Car class
        Car anotherCar = new Car();
        anotherCar.Make = "Honda";
        anotherCar.Model = "Civic";

        // Calling the StartEngine method on the second object
        anotherCar.StartEngine();

        Console.ReadLine();
    }
}
```

In this program, we defined a ‘Car’ class with properties ‘Make’ and ‘Model’, and a method ‘StartEngine’. Then, in the Main method, we created two Car objects using the ‘new’ keyword, set their properties, and call the ‘StartEngine’ method on each of them.

When we run this code, the output will be :

```
Engine started!

Engine started!
```

In C# Each object has its own set of properties and can invoke methods independently.

Q1. f) Define the accessibility modifiers – public, private, protected, internal and protected internal. (5)

Ans. Accessibility modifiers are used to control the visibility and accessibility of classes, methods, properties, and other members within a program. The following modifiers are as follows :

1. Public: The `public` modifier makes a class or member accessible from any other code in the same assembly or from other assemblies that reference it. It has the widest accessibility.

Example:

```
 public class MyClass
{
   public int MyProperty { get; set; }
   public void MyMethod() { }
}
```

2. Private: The `private` modifier restricts the accessibility of a class member to within the same class. It cannot be accessed from outside the class, including derived classes.

Example:

```
public class MyClass
{
   private int myPrivateField;
   private void MyPrivateMethod() { }
}
```

3. Protected: The `protected` modifier allows access to a member within the same class or from derived classes. It is not accessible from outside the class hierarchy.

Example:

```
 public class MyBaseClass
{
   protected int myProtectedField;
   protected void MyProtectedMethod() { }
}
   public class MyDerivedClass : MyBaseClass
{
   public void AccessProtectedMembers()
   {
      myProtectedField = 42;
      MyProtectedMethod();
    }
```

4.Internal: The `internal` modifier allows access to a member within the same assembly but restricts access from other assemblies. It is useful when you want to expose members only to code within the same project.

Example:

```
internal class InternalClass
{
   internal int MyInternalField;
   internal void MyInternalMethod() { }
}
```

5. **Protected Internal:** The `protected internal` modifier combines the behavior of `protected` and `internal`. It allows access to a member within the same assembly or from derived classes, regardless of whether they are in the same assembly or a different one.

Example:

```
 public class MyBaseClass
{
   protected internal int myProtectedInternalField;
   protected internal void MyProtectedInternalMethod() { }
}
  public class MyDerivedClass : MyBaseClass
{
   public void AccessProtectedInternalMembers()
   {
      myProtectedInternalField = 42;
      MyProtectedInternalMethod();
   }
}
```

Accessibility Modifiers play a vital role in defining the scope and security of your code and controlling how different parts of your application interact with each other.

Q2. a) What is postback?Explain IsPostBack property with example. (5)

Ans. Defining postback

"postback" refers to the process where a web page is posted back to the server for processing after the initial request was sent by the client (usually a user clicking a button or triggering an event on the web page). When a postback occurs, the server processes the data, raises the appropriate events, and sends a response back to the client, updating the web page accordingly.

IsPostBack property

IsPostBack is a property is the property that allows you to determine whether the current request to the server is the result of a postback or the initial page load. It returns a boolean value, true if the current request is a postback, and false if it's the initial page load. IsPostBack property is always false by default.

Example :

<u>WebForm1.aspx</u>

```
<asp:TextBox ID="txtName" runat="server"></asp:TextBox>
<asp:Button ID="btnSubmit" runat="server" Text="Submit"
OnClick="btnSubmit_Click" />
<asp:Label ID="lblMessage" runat="server"></asp:Label>
```

<u>WebForm1.aspx.cs</u>

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
   {
      lblMessage.Text = "Welcome! Please enter your name.";
   }
}
protected void btnSubmit_Click(object sender, EventArgs e)
{
   string name = txtName.Text;
   lblMessage.Text = "Hello, " + name + "! Thank you for submitting.";
}
```

Explanation:

- The `Page_Load` event handler runs when the page is first loaded (`IsPostBack` is `false`). It sets the initial text of the `lblMessage` label to display a welcome message with instructions.

- When the user enters their name in the TextBox (`txtName`) and clicks the "Submit" Button (`btnSubmit`), the `btnSubmit_Click` event handler is triggered.
- The `btnSubmit_Click` event handler retrieves the entered name, and updates the `lblMessage` label with a personalized greeting message, thanking the user for submitting their name.
- The `IsPostBack` property helps in setting up the initial welcome message during the first page load and handling the personalized greeting message only on subsequent postbacks (when the button is clicked).

Q2. b) List and describe the various file types used in an ASP.NET application.(5)

Ans. In ASP.NET, we can work with various file types for different purposes, such as source code files, configuration files, data files, content files, and more. Some common file types are as follows :

File Name	Description
Ends with .aspx	These are ASP.NET web pages. They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start your web application.
Ends with .ascx	These are ASP.NET user controls. User controls are similar to web pages, except that the user can't access these files directly. Instead, they must be hosted inside an ASP.NET web page. 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..
web.config	This is the configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more.
global.asax	This is the global application file. You can use this file 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).

Ends with .cs	These are code-behind files that contain C# code. They allow you to separate the application logic from the user interface of a web page.

ASP.NET Folders

Directory	Description
App_Browsers	Contains .browser files that ASP.NET uses to identify the browsers that are using your application and determine their capabilities.
App_Code	Contains source code files that are dynamically compiled for use in your application.
App_GlobalResources	Stores global resources that are accessible to every page in the web application. This directory is used in localization scenarios, when you need to have a website in more than one language.
App_LocalResources	Serves the same purpose as App_GlobalResources, except these resources are accessible to a specific page only.
App_WebReferences	Stores references to web services, which are remote code routines that a web application can call over a network or the Internet.
App_Data	Stores data, including SQL Server Express database files. You're free to store data files in other directories.
App_Themes	Stores the themes that are used to standardize and reuse formatting in your web application
Bin	Contains all the compiled .NET components (DLLs) that the ASP.NET web application uses here.

Q2. c) What is an event ? How is an event handler added? (5)

Ans. An event

An event in ASP.NET is an action or occurrence that happens on a web page, such as a button click or page load. Events allow you to respond to user interactions or system actions.

Adding an event handler

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

1. Define the Event Handler Method:

In the code-behind file (e.g., ExamplePage.aspx.cs) of your ASP.NET web page or user control, create a method that will handle the event. This method should have a specific signature based on the event you want to handle.

2. Subscribe to the Event:

In the markup (ASPX) file (e.g., ExamplePage.aspx), locate the control (e.g., button) that triggers the event. Add an attribute like OnClick or OnLoad to the control, and set its value to the name of the event handler method you defined in step 1.

3. Handle the Event:

When the event occurs during the page lifecycle (e.g., button is clicked), the event handler method you defined will automatically execute. You can now add the desired logic inside that method to respond to the event.

Q2. d) Write short note on list controls in ASP.NET. (5)

Ans. 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

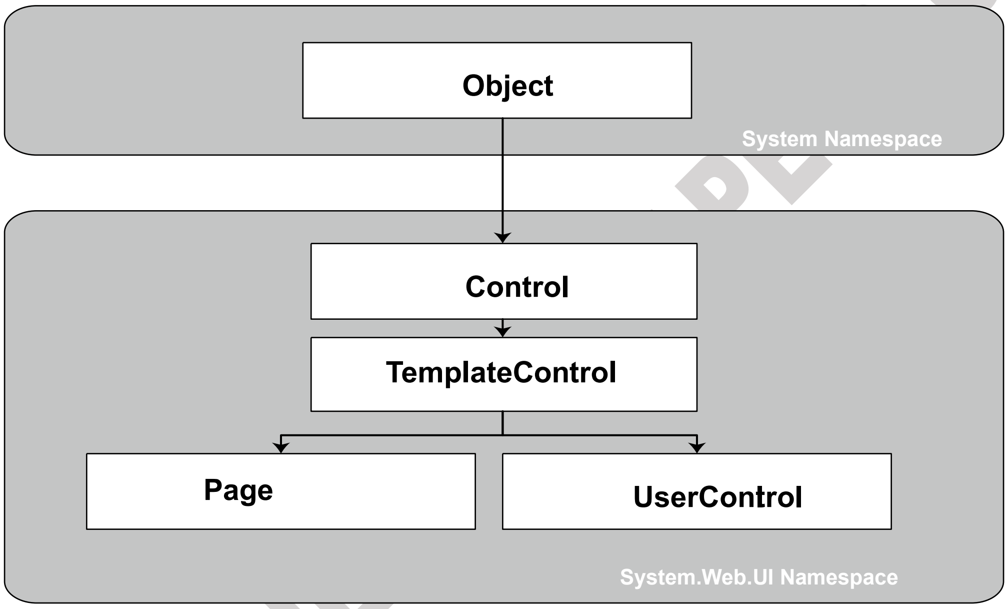
ListBox, but every item is rendered as a check box or option button, respectively.

Multiple-Select List Controls
Some list controls can allow multiple selections. This isn't allowed for the DropDownList or RadioButtonList, but it is supported for a ListBox, provided you have set the SelectionMode property to the enumerated value ListSelectionMode.Multiple. The user can then select multiple items by holding down the Ctrl key while clicking the items in the list.The SelectedIndex and SelectedItem properties aren't much help with a list that supports multiple selection, because they will simply return the first selected item. Instead, we can find all the selected items by iterating through the Items collection of the list control and checking the ListItem.

The Bulleted List Control
The Bulleted List control is a server-side equivalent of the <ul> (unordered list) and <ol> (ordered list) elements. As with all list controls, we can set the collection of items that should be displayed through the Items property.

Q2. e) Explain the need of user control. How is it used and created? (5)
Ans. A User Control is a separate, reusable part of a page. We can put a piece of a page in a User Control, and then reuse it from a different location. A notable difference is that User Controls can be included on multiple pages, while a page can't. User Controls are used much like regular server controls, and they can be added to a page declaratively, just like server controls can. A big advantage of the User Control is that it can be cached, using the Output Cache functionality described in a previous chapter, so instead of caching an entire page, we may cache only the User Control, so that the rest of the page is still re-loaded on each request.
Creation of User Control:
Following steps are used to create User Control.
1. Open Visual Studio.
2. "File" -> "New" -> "Project..." then select ASP.NET Webform Application.

3. Add a new web form.
To create a new User Control, in Solution Explorer, Add New Item, provide your File Name and click Add. Design User Control as per our requirement. Next step to use User Control in .aspx page.
Add the following line below the standard page declaration:
<%@RegisterTagPrefix="My"TagName="UserInfoBoxControl"Src="~/UserInfoBoxControl.ascx" % >
Make sure that the src value matches the path to your User Control file. Now you may use the User Control in your page, like any other control. For instance, like this:
<My:UserInfoBoxControlrunat="server"ID="MyUserInfoBoxControl"/>



Q2. f) What is the purpose of validation controls? List and explain the use of validation controls available in ASP.NET. (5)
Ans. 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

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-touse but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET

- RequiredFieldValidation Control
- CompareValidator Control
- RangeValidator Control
- RegularExpressionValidator Control
- CustomValidator Control
- ValidationSummary

The below table describes the controls and their work:

Validation Control	Description
RequiredFieldValidation	Makes an input control a required field.
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value.
RangeValidator	Checks that the user enters a value that falls between two values.
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern.
CustomValidator	Allows you to write a method to handle the validation of the value entered.
ValidationSummary	Displays a report of all validation errors occurred in a Web page.

1. RequiredFieldValidator Control

The RequiredFieldValidator control is simple validation control, which checks to see if the data is entered for the input control. We can have a RequiredFieldValidator control for each form element

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate" runat="server"
ControlToValidate="txtName" ErrorMessage="Please enter name !!">

</asp:RequiredFieldValidator>
```

2. RangeValidator Control

The RangeValidator Server Control is another validator control, which checks to see if a control value s within a valid range. The attributes that are necessary to this control are: MaximumValue, MinimumValue, and Type.

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
ErrorMessage="Enter your class (6 - 12)" MaximumValue="12"
 MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

3. RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

The syntax of the control is as given:

```
<asp:RegularExpressionValidator ID="string" runat="server"
ErrorMessage="string"
 ValidationExpression="string" ValidationGroup="string">

</asp:RegularExpressionValidator>
```

Q3. a) Describe the use of multiple catch statements in exception handling using example. (5)

Ans. Exception handling:

The mechanism of Exception Handling is throwing an exception and catching it C# uses try-catch block. Code which may give rise to exceptions is enclosed in a try block, and Catch block catches that exception and handles it appropriately. The try block is followed by one or more catch blocks.

Structured exception handling provides several key features:

1. Exceptions are object-based: Each exception provides a significant amount of diagnostic information wrapped into a neat object, instead of a simple message and error code. These exception objects also support an InnerException property that allows you to wrap a generic error over the more specific error that caused it.

2. Exceptions are caught based on their type: This allows you to streamline error-handling code without needing to sift through obscure error codes.
3. Exception handlers use a modern block structure: This makes it easy to activate and deactivate different error handlers for different sections of code and handle their errors individually.
4. Exception handlers are multilayered: You can easily layer exception handlers on top of other exception handlers, some of which may check for only a specialized set of errors. As you'll see, this gives you the flexibility to handle different types of problems in different parts of your code, thereby keeping your code clean and organized.
5. Exceptions are a generic part of the .NET Framework: This means they're completely cross-language compatible. Thus, a .NET component written in C# can throw an exception that you can catch in a web page written in VB.

Basic syntax:

```
try
{
//programming logic(code which may give rise to exceptions)
}
catch (Exception e)
{
//message on exception
}
finally
{
// always executes
}
```

Try: A try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more catch blocks.
Catch: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
Finally: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
Example:

```
using System;
```

```
class tryCatch
{
public static void Main()
{
int k=0;
try
{
int n= 10/k;
Console.WriteLine(”n=” + n);
}
catch(Exception e)
{
Console .WriteLine (“Division By zero exception”);
}
Console.WriteLtne(”Statement executed after Exception because of try catch”);
}
```

Q3. b) What is QueryString? How to send a name and marks of a student from one page to another using QueryString? (5)

Ans. 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=foo
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 foo is its value. We can send multiple values through querystring, separated by the & symbol.

To send a name and marks of a student from one page to another using QueryString following are the steps:

Step 1: Create Page1.aspx (Source Page)

Page1.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Page1.aspx.cs"
Inherits="MyNamespace.Page1" %>
<!DOCTYPE html>
```

```
<html>
<head>
   <title>Page 1</title>
</head>
<body>
   <form runat="server">
      <label for="txtName">Name:</label>
      <asp:TextBox runat="server" ID="txtName"></asp:TextBox>
      <br />
      <label for="txtMarks">Marks:</label>
      <asp:TextBox runat="server" ID="txtMarks"></asp:TextBox>
      <br />
      <asp:Button runat="server" ID="btnSubmit" Text="Submit"
OnClick="btnSubmit_Click" />
   </form>
</body>
</html>
```

Page1.aspx.cs

```
using System;
namespace MyNamespace
{
   public partial class Page1 : System.Web.UI.Page
   {
      protected void btnSubmit_Click(object sender, EventArgs e)
      {
          string name = txtName.Text;
          int marks = Convert.ToInt32(txtMarks.Text);

         // Redirect to Page2 with QueryString parameters
         Response.Redirect($"Page2.aspx?name={name}&marks={marks}");
      }
   }
}
```

Step 2: Create Page2.aspx

Page2.aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Page2.aspx.cs"
Inherits="MyNamespace.Page2" %>

<!DOCTYPE html>
<html>
```

```
<head>
   <title>Page 2</title>
</head>
<body>
   <form runat="server">
      <asp:Label runat="server" ID="lblResult"></asp:Label>
   </form>
</body>
</html>
```

Page2.aspx.cs

```
using System;
namespace MyNamespace
{
   public partial class Page2 : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         // Check if the QueryString parameters exist
         if (Request.QueryString["name"] != null && Request.QueryString["marks"] !=
null)
         {
            // Get the name and marks from the QueryString
            string name = Request.QueryString["name"];
             int marks = Convert.ToInt32(Request.QueryString["marks"]);

             // Display the result on the page
            lblResult.Text = $"Name: {name}, Marks: {marks}";
         }
      }
   }
}
```

In this example, when the user enters the name and marks on Page1.aspx and clicks the "Submit" button, the data will be sent to Page2.aspx using QueryString. On Page2.aspx, the received name and marks will be displayed on the page.

Q3. c) Explain the events in global.asax file with respect to state management. (5)

Ans.ASP.NET implements application state using the System.Web.HttpApplicationState class. It provides methods for storing information which can be accessed globally. Information stored on application state will be available for all the users using the website. Usage of application state is the same as sessions. The following code shows storing a value in an application variable and reading from it.

```
Application["pageTitle"] = "Welcome to my website - ";
String pageTitle;
if (Application["pageTitle"] != null)
pageTitle = Application["pageTitle"].ToString();
```

We should get a basic knowledge about the events associated with application and session.

These events can be seen in the global.asax file.

- **Application_Start:** This event executes when application initializes. This will execute when ASP.NET worker process recycles and starts again.
- **Application_End:** Executes when the application ends.
- **Session_Start:** Executes when a new session starts.
- **Session_End:** Executes when session ends.

  Note: this event will be fired only if you are using InProc as session mode.

Example

The most common usage of application variables is to count the active number of visitors that are browsing currently.

The following code shows how this is done.

```
void Application_Start(object sender, EventArgs e)
{
// Application started - Initializing to 0
Application["activeVisitors"] = 0;
}
void Session_Start(object sender, EventArgs e)
{
if (Application["activeVisitors"] != null)
{
```

```
Application.Lock();
int visitorCount = (int)Application["activeVisitors"];
Application["activeVisitors"] = visitorCount++;
Application.UnLock();
}
}
```

Data stored in session will be kept in server memory and it is protected as it will never get transmitted to a client. Every client that uses the application will have separate sessions. Session state is ideal for storing
user specific information.

Session State

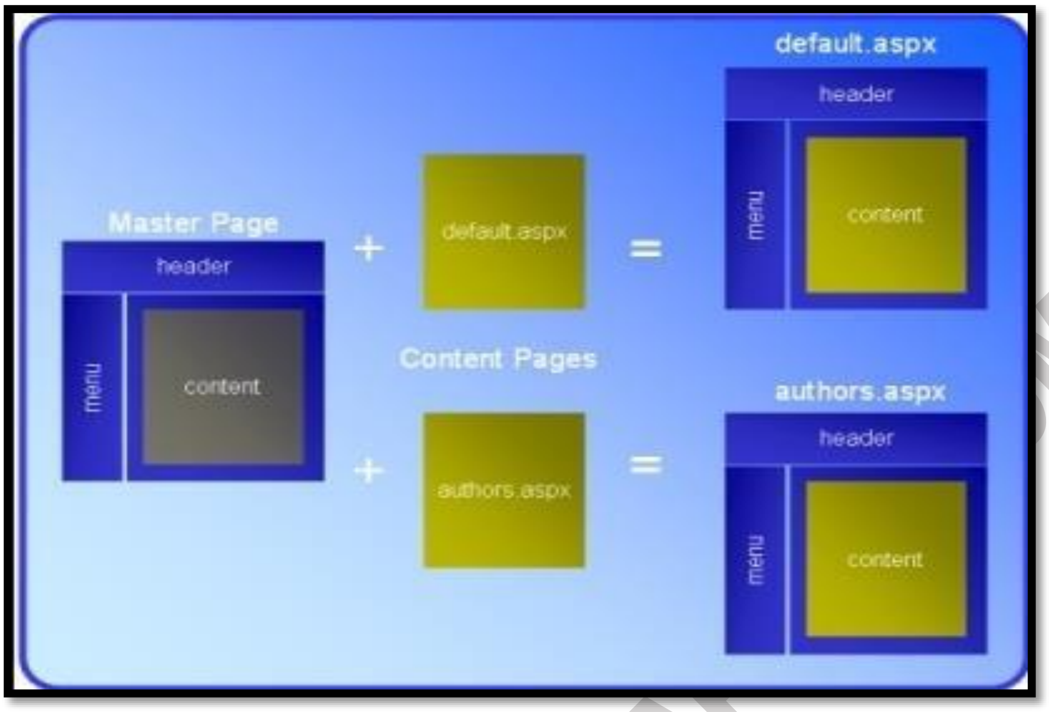
- Session state is user and browser specific.
- Session state can be stored in memory on the server as well as client's cookies. If client has disabled cookies in his browser then session state will be stored in URL.
- Session state has scope to the current browser only. If we change the browser session id is changed.

Application State

- Application state is application specific.
- Application state is stored only in the memory on the server.
- Application state does not track client's cookies or URL.
- Application state has no scope to the current browser. If we change the browser application id remains same.

Q3. d) How is the connection between content page and master page is established? (5)

Ans. ASP.NET master pages allow us to create a consistent layout for the pages in our application. A single master page defines the look and feel and standard behavior that we want for all of the pages (or a group of pages) in our application. We can then create individual content pages that contain the content we want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page. Master pages actually consist of two pieces, the master page itself and one or more content pages.

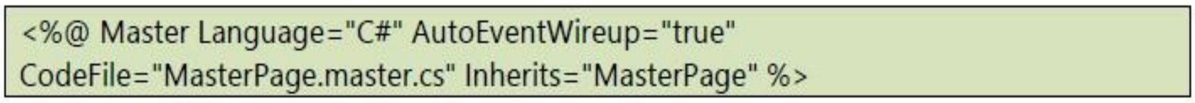


Use of Master Pages

The master pages can be used to accomplish the following:

- ✓ Creating a set of controls that are common across all the web pages and attaching them to all the web pages.
- ✓ A centralized way to change the above created set of controls which will effectively change all the web pages.
- ✓ To some extent, a master page looks like a normal ASPX page.
- ✓ It contains static HTML such as the <html>, <head>, and <body> elements, and it can also contain other HTML and ASP.NET server controls. Inside the master page, you set up the markup that you want to repeat on every page, like the general structure of the page and the menu.
- ✓ However, a master page is not a true ASPX page and cannot be requested in the browser directly it only serves as the template that real web pages called content pages
- ✓ One difference is that while web forms start with the Page directive, a master page starts with a Master directive that specifies the same

information, as shown here



```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

Q3. e) What is the theme? Explain how to create and use a theme on a website. (5)

Ans. A theme decides the look and feel of the website. It is a collection of files that define the looks of a page. It can include skin files, CSS files & images.

We define themes in a special App_Themes folder. Inside this folder is one or more subfolders named Theme1, Theme2 etc. that define the actual themes. The theme property is applied late in the page's life cycle, effectively overriding any customization we may have for individual controls on our page.

There are 3 different options to apply themes to our website:

1. Setting the theme at the page level: The Theme attribute is added to the page directive of the page.

```
<%@PageLanguage="C#"AutoEventWireup="true"CodeFile="Default.aspx.cs"Inherits="Default"Theme="Theme1"%>
```

2. Setting the theme at the site level: To set the theme for the entire website we can set the theme in the web.config of the website. Open the web.config file and locate the <pages> element and add the theme attribute to it:

```
<pagestheme="Theme1">
....
....
</pages>
```

3. Setting the theme programmatically at runtime: here the theme is set at runtime through coding. It should be applied earlier in the page's life cycle ie. Page_PreInit event should be handled for setting the theme. The better option is to apply this to the Base page class of the site as every page in the site inherits from this class.

```
Page.Theme = Theme1;
```

Uses of Themes

1. Since themes can contain CSS files, images and skins, you can change colors, fonts, positioning and images simply by applying the desired themes.

2. We can have as many themes as we want, and we can switch between them by setting a single attribute in the web.config file or an individual aspx page. Also, we can switch between themes programmatically.

3. Setting the themes programmatically, we are offering our users a quick and easy way to change the page to their likings.

4. Themes allow us to improve the usability of our site by giving users with vision problems the option to select a high contrast theme with a large font size.

Q3. f) What is URL mapping? How is URL mapping and routing implemented in ASP.NET? (5)

Ans. In some situations, you might want to have several URLs lead to the same page. This might be the case for a number of reasons—maybe you want to implement your logic in one page and use query string arguments but still provide shorter and easier to remember URLs to your website users (often called friendly URLs).

The mapping rules are stored in the web.config file and they are applied before any other processing takes place. You define URL mapping in the <urlMapping> section of the web.config file. You supply two pieces of information—the request URL(as the url attribute) and the new destination URL (mappedUrl).

```
<configuration>
<system.web>
<urlMapping enabled=”true”>
<add url=”~/category.aspx” mappedUrl=”~/default.aspx?category=default” />
<add url=”~/software.aspx” mappedUrl=”~/deault.aspx?category=software” />
</urlMappings>
</system.web>
</configuration>
```

URL Routing

URL routing doesn't take place in the web.config file. Instead, it's implemented using code. To register a route, you use the RouteTable class from the System.Web.Routing namespace. To make life easier, you can start by importing that namespace: using System.Web.Routing; create custom routes by calling the MapPageRoute() method, which takes three arguments:

- **routeName:** This is a name that uniquely identifies the route. It can be whatever you want.
- **routeUrl:** This specifies the URL format that browsers will use. Typically, a route URL consists of one or more pieces of variable information, separated by slashes, which are extracted and provided to your code. For example, you might request a product page by using a URL such as /products/4312.
- **physicalFile:** This is the target web form—the place where users will be redirected when they use the route. The information from the original routeUrl will be parsed and made available to this page as a collection through the Page.RouteData property.

Example

```
Protected void Application_Start(object sender, EventArgs e)
{
RouteTable.Routes.MapPageRoute("routeName","routeUrl", "physicalFile");
RouteTable.Routes.MapPageRoute("product-details",
"product/{productiID}", "~/productinfo.aspx"");
}
http://localhost:[PortNumber]/Routing/product/FI_00345

protected void Page_Load(object sender, EventArgs e)
{
  String productid = (string)Page.RouteDataValues["productID"];
  lblInfo.Text = "You requested" + productID;
}
```

Q4. a) Describe the SqlConnection class with an example. (5)

Ans. The SqlConnection class

- The most important property of this class is ConnectionString. A Connection String is a string that provides the information that's needed to connect to a database server. It can also contain authentication information such as a user id and password.

- The two methods of this class that are shown in this figure let you open and close the connection. In general, you should leave a connection open only while data is being retrieved or updated. When you use a data adapter, though the connection is opened and closed for you so you don't need to use these methods.

Some Common members of the SqlConnection class

Property	Description
ConnectionString	Contains information that let's you connect to a SQL Server database,including the server name, the database name and login information.
Method	Description
Open	Opens a connection to a database.
Close	Closes a connection to a database.

```
using System;
using System.Collection.Generic;
using System.Text;
using System.Data.SqlClient;

namespace CommandTypeEnumeration
{
  Class Program
   {
     Static void Main(string[] args)
      {
        //Create a connection string
        String ConnectionString = "Integrated Security = SSPI; +"Initial
Catalog= Northwind; "+" Data source = localhost; ";
String SQL = "SELECT * FROM Customers";

//create a connection object
SqlConnection conn = new SqlConnection(ConnectionString);


//Create a command object
SqlCommand cmd = new SqlCommand(SQL,conn);
conn.Open();

//Call ExecuteReader to return a DataReader
SqlDataReader reader = cmd.ExecuteReader();
Console.WriteLine("customer ID, Contact Name, "+" Contact Title, Address");
Console.WriteLine("=====================");

while(reader.Read())
{
   Console.Write(reader["CustomerID"].ToString() + ", ");
   Console.Write(reader["ContactName"].ToString() +", ");
  Console.Write(reader["ContactTitle"].ToString() +", ");
  Console.WriteLine(reader["Address"].ToString() +", ");
 }

//Release resources
    Reader.Close()
    Conn.Close();
          }
      }
 }
```

Q4. b) Differentiate between DataSet and DataReader. (5)

Ans.

DataSet	**DataReader**
1. The DataSet class in ADO.Net operates in an entirely disconnected nature.	1.DataReader is a connection oriented service.
2.DataSet is an in-memory representation of a collection of Database objects including related tables, constraints and relationships among the tables.	2.DataReader is designed to retrieve a read-only, forward-only stream of data from data sources.
3. It fetches entire table or tables at a time so greater network cost.	3.It fetches one row at a time so very less network.
4. DataSet is not read-only so we can do any transaction on them.	4.DataReader is read-only so we can't do any transaction on them.
5.DataAdapter is used to get data in DataSet.	5.DataAdapter is not required.

6.DataSet works with the help of xml technology.	6.DataReader doesn’t provide this feature.

7. **Example**
Dataset ds=new Dataset();
DataAdapter1Fill(ds,”newtablename”)
//where new table name is table alias name in dataset.
	7.**Example:**
	SqlCommand cmd = new sqlcommand(select * from emptable);
	Data Reader dr = cmd.ExecuteReader()

Q4. c) Write a C# code to insert data in database table. Write comments wherever required. (5)

Ans.

```
using System;
using System.Data.SqlClient;

class Program
{
   static void Main()
   {
      // Replace the connection string with your actual database connection string
      string connectionString = "Data Source=YourServer;Initial
Catalog=YourDatabase;Integrated Security=True";

      // Sample data to be inserted into the table
      string name = "John Doe";
      int age = 25;

      try
      {
         // Create a new SqlConnection object with the connection string
         using (SqlConnection connection = new SqlConnection(connectionString))
         {
            // Open the database connection
            connection.Open();

            // Write your SQL query to insert data into the table
            string insertQuery = "INSERT INTO YourTableName (Name, Age) VALUES
(@Name, @Age)";

            // Create a new SqlCommand object with the SQL query and connection
            using (SqlCommand command = new SqlCommand(insertQuery,
connection))
            {
              // Add parameters to the SqlCommand to prevent SQL injection and
provide values
              command.Parameters.AddWithValue("@Name", name);
              command.Parameters.AddWithValue("@Age", age);

              // Execute the INSERT query
              int rowsAffected = command.ExecuteNonQuery();

              // Check the number of rows affected to ensure data insertion
              if (rowsAffected > 0)
              {
                Console.WriteLine("Data inserted successfully!");
              }
              else
              {
                Console.WriteLine("Data insertion failed!");
              }
            }
          }
        }
        catch (Exception ex)
        {
          // Handle any exceptions that may occur during the process
          Console.WriteLine("An error occurred: " + ex.Message);
        }
    }
}
```

Q4. d) What is the use of data source control? Explain various types of data sources in ASP.NET.

Ans. 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

---

Q4. e) Write a code to display data from a table named Students(Roll no, Name, Marks and display it on gridview control when page is loaded. (5)

Ans.
```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Web.UI;
using System.Web.UI.WebControls;

public partial class StudentsPage : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      if (!IsPostBack)
      {
         // Replace the connection string with your actual database connection string
```

```
        string connectionString = "Data Source=YourServer;Initial
Catalog=YourDatabase;Integrated Security=True";

        try
        {
            // Create a new SqlConnection object with the connection string
            using (SqlConnection connection = new SqlConnection(connectionString))
            {
                // Write your SQL query to retrieve data from the Students table
                string query = "SELECT RollNo, Name, Marks FROM Students";

                // Create a new SqlCommand object with the SQL query and connection
                using (SqlCommand command = new SqlCommand(query, connection))
                {
                    // Open the database connection
                    connection.Open();

                    // Create a new SqlDataAdapter to fetch data from the database
                    using (SqlDataAdapter dataAdapter = new SqlDataAdapter(command))
                    {
                        // Create a new DataTable to store the retrieved data
                        DataTable dataTable = new DataTable();

                        // Fill the DataTable with data from the Students table
                        dataAdapter.Fill(dataTable);

                        // Bind the DataTable to the GridView control to display the data
                        gvStudents.DataSource = dataTable;
                        gvStudents.DataBind();
                    }
                }
            }
        }
        catch (Exception ex)
        {
            // Handle any exceptions that may occur during the process
            // For simplicity, just display the error message on the page
            lblError.Text = "An error occurred: " + ex.Message;
        }
    }
    }
}
```

Q4. f) Describe i)ExecuteNonQuery ii) ExecuteScalar iii)ExecuteReader. (5)

Ans. In C#, these three methods are part of the ADO.NET framework, which is used for data access and manipulation with databases. Each of these methods serves a different purpose and is commonly used in different scenarios:

i) ExecuteNonQuery:

- The ExecuteNonQuery method is used to execute SQL statements that do not return any data, such as INSERT, UPDATE, DELETE, and DDL (Data Definition Language) statements (e.g., CREATE, ALTER, DROP).
- It is primarily used for modifying data in the database rather than retrieving data.
- The method returns the number of rows affected by the SQL statement.
- For example, if you execute an INSERT statement, it will return the number of rows inserted into the table.

ii) ExecuteScalar:

- The ExecuteScalar method is used to execute SQL queries that return a single value, typically an aggregate value (e.g., COUNT, SUM, AVG) or the result of a scalar subquery.
- It is commonly used when you expect a single result from the query, and you want to retrieve that specific value from the database.
- The method returns an object, so you need to cast it to the appropriate data type.
- If the query does not return any results, ExecuteScalar will return null.

iii) ExecuteReader:

- The ExecuteReader method is used to execute SQL queries that return a set of rows (e.g., SELECT queries) and retrieve the data as a SqlDataReader object.

- It is typically used when you need to read and process multiple rows of data from the database.
- The SqlDataReader provides forward-only, read-only access to the data, which makes it efficient for large result sets.
- You need to iterate through the reader to access each row of data.

These methods are fundamental in ADO.NET for performing various database operations and retrieving data from databases in C#.

Q5. a) Write a code to save employee data as empId, empName, empDept, and empDesignation data in XML file. (5)

Ans. Working with XML Documents in Memory

The XDocument class provides a different approach to XML data. It provides an in-memory model of an entire XML document. You can then browse through the entire document, reading, inserting, or removing nodes at any location. (You can find the XDocument and all related classes in the System.Xml.Linq namespace.)

To start building a next XML document, you need to create the XDocument, XElement, and XAttribute objects that constitute it. All these classes have useful constructors that allows you to create and initialize them in one step. For example, you can create an element and supply text content that should be placed inside using code like this.

```
XElement element = new XElement("employee",
                        new XElement("empId", 3),
                        new XElement("cmpName", "TS"),
                        new XElement("empDep", "IT"),
                        new XElement("cmpDesignation", "MANAGER"),
);
```

Here's the scrap of XML that this code creates:

```
<employee>
        <empId>3</empId>
        <cmpName>TS</cmpName>
        <empDep>IT</empDep>
        <cmpDesignation>IT</cmpDesignation>
 </employee>
```

Q5. b) What is XML? List and explain the various XML classes. (5)

Ans. 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.

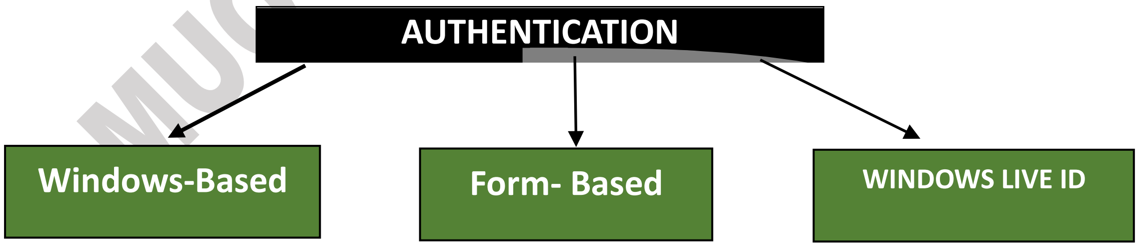
XDocument

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.

Q5. c) What do you mean by authentication? Describe its various types of authentication. (5)

Ans. 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.

There are 3 types of authentication as follows:



Windows-based authentication:

- It causes the browser to display a login dialog box when the user attempts to access restricted page.
- It is supported by most browsers.

- It is configured through the IIS management console.
- It uses windows user accounts and directory rights to grant access to restricted pages.

```
<authentication mode =”Windows”>
<forms name=”AuthenticationDemo” loginUrl=”logon.aspx” protection=”All”
path=”/” timeout=”30” />
</authentication>
```

Forms-based authentication:

- Developer codes a login form that gets the user name and password.
- The username and password entered by user are encrypted if the login page uses a secure connection.
- It doesn’t reply on windows user account.

```
<authentication mode=”Forms”>
  <forms name=”AuthenticationDemo” loginUrl =”logon.aspx”
protection=”All” path=”/”  timeout=”30” />
```

Deny access to anonymous user in the <authorization> section as follows:

```
 <authorization>
       <deny users =”?” />
</authorization>
```

Windows Live ID authentication:

- It is centralized authentication service offered by Microsoft.
- The advantage is that the user only has one maintain one username and password.

Q5. d) Explain the use of UpdateProgress Control in AJAX. (5)

Ans. The UpdateProgress provides status information about partial page updates in UpdatePanel controls. Despite the visual problems that post backs usually cause, they have one big advantage: the user can see something is happening. The UpdatePanel makes this a little more difficult. Users have no visual cue that something is happening until it has happened. To tell users to hold on for a few seconds while their request is being processed, we can use the UpdateProgress control. We usually put text such as “Please wait” or an animated image in this template to let the user know something is happening, although any other markup is acceptable as well. We can connect the UpdateProgress control to an UpdatePanel using the AssociatedUpdatePanelID property. Its contents, defined in the <ProgressTemplate> element, are then displayed whenever the associated UpdatePanel is busy refreshing.

Example:

In the example below, we use the same .gif to display progress while the UpdatePanel is updating its content. For understanding purposes, we have emulated a time-consuming operation by setting a delay of 3 seconds by using **Thread.Sleep(3000)** on the button click.

```
protected void btnInvoke_Click(object sender, EventArgs e)
{
System.Threading.Thread.Sleep(3000);
lblText.Text = "Processing completed";
}
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<asp:UpdateProgress ID="updProgress"
AssociatedUpdatePanelID="UpdatePanel1"
runat="server">
<ProgressTemplate>
<img alt="progress" src="images/progress.gif"/>
Processing...
</ProgressTemplate>
</asp:UpdateProgress>

 <asp:UpdatePanel ID="UpdatePanel1" runat="server">
 <ContentTemplate>
 <asp:Label ID="lblText" runat="server" Text=""></asp:Label>
 <br />
 <asp:Button ID="btnInvoke" runat="server" Text="Click"
 onclick="btnInvoke_Click" />
</ContentTemplate>
 </asp:UpdatePanel>
```

In the code shown above, we use the AssociatedUpdatePanelID property of the UpdateProgress control to associate it with an UpdatePanel control.

Q5. e) What is the 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. (5)

Ans. 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();

 }
```

Q5. f) What are the benefits of using AJAX? Explain Update panel and Script Manager. (5)

Ans. The major benefit of Ajax is partial page rendering. The partial update of a page does not necessitate full reload of the page and hence leads to flicker-free page rendering.

UpdatePanel

- We can refresh the selected part of the web page by using UpdatePanel control, Ajax UpdatePanelcontrol contains a two child tags that is ContentTemplate and Triggers. In a ContentTemplate tag we used to place the user controls and the Trigger tag allows us to define certain triggers which will make the panel update its content.

```
<asp:UpdatePanel ID="updatepnl" runat="server">
<ContentTemplate>
```

- All the contents that must be updated asynchronously (only ContentTemplate parts are updated and rest of the web page part is untouched) are placed here. It allows us to send request or post data to server without submit the whole page so that is called asynchronous.
- UpdatePanel is a container control. A page can have multiple update panels.
- The UpdatePanel control enables you to create a flicker free page by providing partial-page update support to it.
- It identifies a set of server controls to be updated using an asynchronous post back.
- If a control within the UpdatePanel causes a post back to the server, only the content within that UpdatePanel is refreshed.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate>
</ContentTemplate>
</asp:UpdatePanel>
```

ScriptManager

- The ScripManager Control manages the partial page updates for UpdatPanel controls that are on the ASP.NET web page or inside a user control on the web page.
- This control manages the client script for AJAX-enabled ASP.NET web page and ScripManager control support the feature as partial-page rendering and web-service calls.
- The ScriptManager control manages client script for AJAX-enabled ASP.NET Web pages.
- Although this control is not visible at runtime, it is one of the most important controls for an Ajax
- enabled web page.
- There can be only one ScriptManager in an Ajax enabled web page.

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