ASP.NET with C#: Solution Set October 2016

1.	**Attempt *any two* of the following:**
a.	**What is flow control? Explain use of break and continue statements in loop.** **Answer(Flow control1+ break 2+ continue 2):** **Flow Control(Looping and Branching):** In each case, program execution has proceeded from one line to the next in top-to-bottom order, missing nothing. If all applications worked like this, then you would be very limited in what you could do. This chapter describes two methods for controlling program flow — that is, the order of execution of lines of C# code: *branching* and *looping*. Branching executes code conditionally, depending on the outcome of an evaluation, such as ‘‘only execute this code if the variable myVal is less than 10.’’ Looping repeatedly executes the same statements, either a certain number of times or until a test condition has been reached. Both of these techniques involve the use of *Boolean logic.* In the last chapter you saw the bool type, but didn’t actually do much with it. This chapter uses it a lot, so the chapter begins by discussing what is meant by Boolean logic so that you can use it in flow control scenarios. **break:** Causes the loop to end immediately while(test-condition) { --- --- if(condition) break; --- --- skip } --- --- **continue:** Continue statement tells the compiler “SKIP THE FOLLOWING STATEMENT AND CONTINUE WITH THE NEXT ITERATION”. Causes the current loop cycle to end immediately (execution continues with the next loop cycle) Example: while(test-condition) { --- --- if(condition) continue; --- --- skip }
b.	**What is namespace? How to create namespace and its alias?** **Answer(Marks: namespace 1+ syntax 2+2 example):** Namespaces: These are the .NET way of providing containers for application code, such that code and its contents may be uniquely identified. Namespaces are also used as a means of categorizing items in the .NET Framework. They may contain union, classes, structures, interfaces, enumerators and delegates. Defining a Namespace

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

Syntax:

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

Syntax for alias:

```
    using alias_name=Namespace_name;
```

Example:

```
using System;
using Cat = System.Text.StringBuilder; // Cat is alia

class Program
{
   static void Main()
   {
       Cat cat = new Cat();
       cat.Append("sparky");
       cat.Append(100);

       Console.WriteLine(cat);
   }
}
```

Output

sparky100

c. **Differentiate between function overloading and function overriding.**

Answer(Marks: 1 mark for each point):

Overloading

Overloading means we will declare methods with same name but different signatures because of this we will perform different tasks with same method name. This overloading also called as compile time polymorphism or early binding.

Def:

Method Overloading or compile time polymorphism means same method names with different signatures (different parameters)

Overriding

Overriding also called as **run time polymorphism** or **late binding** or **dynamic polymorphism**. Method overriding or run time polymorphism means same method names with same signatures.

Def:

In this method overriding or run time polymorphism we can override a method in base class by creating similar function in derived class this can be achieved by using inheritance principle and using “**virtual** & **override**” keywords.

	Function Overriding	Function Overloading
1	Methods name and signatures must be same.	Having same method name with different **Signatures**.
2	Overriding is the concept of **runtime polymorphism**	Overloading is the concept of **compile time polymorphism**
3	When a function of base class is re-defined in the derived class called as Overriding	Two functions having same name and return type, but with different type and/or number of arguments is called as Overloading
4	It needs inheritance.	It doesn't need inheritance.
5	Method should have same data type.	Method can have different data types
6	Method should be public.	Method can be different access specifies

d. **Explain try, catch and finally blocks in exception handling with suitable example.**

Answer(Marks: explanation 3 + example 2):

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.

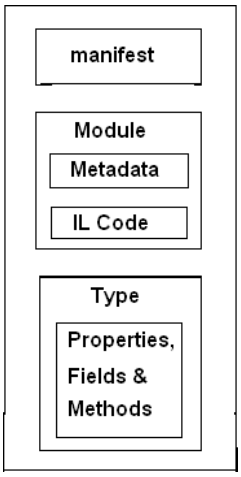
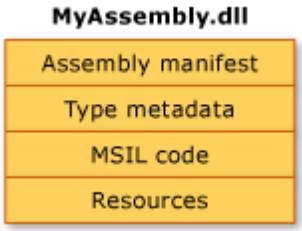
catch: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

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.

Syntax:

```
try
{
   // code that may raise exceptions
}
catch(Exception ex)
{
   // handle exception
}
finally
{
   // final cleanup code
}
```

Example:

2.	**Attempt *any two* of the following:**
a.	**What is an assembly? Explain different components of assembly.** **Answer(Marks: assembly 2 marks + components 3 marks):** **Assembly:** The .NET assembly is the standard for components developed with the Microsoft.NET. Dot NET assemblies may or may not be executable, i.e., they might exist as the executable (.exe) file or dynamic link library (DLL) file. All the .NET assemblies contain the definition of types, versioning information for the type, meta-data, and manifest. **Components of Assembly**  **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. Type Metadata: It contains metadata, which describes each and every type (class, structure, enumeration, and so forth) MSIL: It contains Intermediate language code. Resources: It contains bitmaps, icons, audios and other types of resources. 
b.	**What is garbage collection? How it works?** **Answer(Marks: 2+3):** **Garbage collection:** The Microsoft .NET Framework provides an automated mechanism for reclaiming an object that is no longer in use. This process is usually referred as Garbage Collection. The Microsoft .NET Framework CLR reserves space for each object instantiated in the system. Since memory is not infinite, CLR needs to get rid of those objects that are no longer in use so that the space can be used for other objects. In almost all cases, you do not have to call this method, because the **garbage collector** runs continuously.

Working:

1. The very first step in Garbage Collection is identifying those objects that can be wiped out.
2. To accomplish this step, CLR maintains the list of references for an object.
3. If an object has no more references, i.e. there is no way that the object could be referred to by the application, CLR considers that object as garbage.
4. During Garbage Collection, CLR reclaims memory for all garbage objects.

GC.**Collect**() method is used to call garbage collector explicitly.

Note:
Some students might have written about

Generation 0
Generation 1
Generation 2

c. **Explain the steps in ASP.NET page life cycle.**

Answer(Each step 0.5 mark):

Page Life Cycle in ASP.NET describes the process of developing ASP.NET apps.
When a web page is filled up with details and sent to the Web Server for processing, it goes through a sequence of steps before it finally displays the result in the Web Browser.

Stages of ASP.NET page life cycle:

Stage 0: Page request:
Stage 1: Page initialization:
Stage 2 : View state loading
Stage 3 : Postback data processing
Stage 4 & 5 : Page loading & PostBack change notification
Stage 6: PostBack event handling
Step 7: Page pre rendering phase
Step 8: View state saving:
Step 9: Page rendering:
Step 10: Page unloading:

Methods
Page_Init
LoadViewState
LoadPostData
Page_Load
RaisePostDataChangedEvent
RaisePostBackEvent
Page_PreRender
SaveViewState
Page_Render
Page_Unload

d. **What is CSS selector? Explain different selectors present in CSS.**

Answer(Marks: 1 CSS+1 for each selector):

CSS Selector:

CSS selectors are used to "find" (or select) HTML elements based on their element name, id, class, attribute, and more

Universal Selector (*):

The Universal selector, indicated by an asterisk (*), applies to all elements in your page. The Universal selector can be used to set global settings like a font family. The following rule set changes the font for all elements in your page to Arial:

Example:

```
*{
    font-family: Arial;
 }
```

Type Selector(element selector):

The Type selector enables us to point to an HTML element of a specific type. With a Type selector all HTML elements of that type will be styled accordingly.

Example:

```
h1
{
color: Green;
}
```

This Type selector now applies to all <h1> elements in your code and gives them a green color. Type

Selectors are not case sensitive, so you can use both h1 and H1 to refer to the same heading.

ID Selector (#):

The ID selector is always prefixed by a hash symbol (#) and enables us to refer to a single element in the page. Within an HTML or ASPX page, you can give an element a unique ID using the id attribute. With the ID selector, we can change the behavior for that single element, for example:

Example:

```
#IntroText
{
font-style: italic;
}
```

Because you can reuse this ID across multiple pages in your site (it only has to be unique within a single page), you can use this rule to quickly change the appearance of an element that you use once per page, but more than once in your site, for example with the following HTML code:

```
<p id="IntroText">I am italic because I have the right ID. </p>
```

Class Selector (.):

The Class selector enables us to style multiple HTML elements through the class attribute. This is handy when we want to give the same type of formatting to a number of unrelated HTML elements. The following rule changes the text to red and bold for all HTML elements that have their class attributes set to highlight:

Example:

```
.Highlight
{
    font-weight: bold;
    color: Red;
}
```

3. **Attempt *any two* of the following:**

a. **What is the difference between inline code and code behind?**

Answer:(

Inline code:

Inline code written alongside the html in a page.

The inline is design and design will be mixed in the .aspx page. The events and methods can be written part of the .aspx page. The inline can be selected while creating the web page, it asks the option "Separate the code File". When it is unchecked it will not create the .aspx.cs file.

<script runat="server" type="text/C#">

</script>

The developer can write the code file in the script tag. When you write the code in .aspx page, it does not mean the written code will be visible to the users in the client browser like javascript.

Code Behind:

Code-behind is code written in a separate .aspx.cs file and referenced by the .aspx page The code behind is code will be separated from the design page. It is used to write the properties, events and methods in the class. It will be referred to the code file in the page directive as shown below.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default4.aspx.cs" Inherits="Default4" %>

Benefits of code behind file over inline code:

1) Logic is separated from the design, so it is good readable and easy to understand
2) The designers can work on the .aspx page and the developers can focus on the function in the server side.
3) It will avoid to make the confuse of javascript section and code section

OR

(1 mark for each point)

Code Behind	Inline code
The HTML and controls are in the *.aspx* file, and the code is in a separate *.aspx.vb* or *.aspx.cs* file.	The code is in <script> blocks in the same *.aspx* file that contains the HTML and controls.
The code for the page is compiled into a separate class from which the *.aspx* file derives.	The *.aspx* file derives from the Page class.
All project class files (without the *.aspx* file itself) are compiled into a *.dll* file, which is deployed to the server without any source code. When a request for the page is received, then an instance of the project *.dll* file is created and executed.	When the page is deployed, the source code is deployed along with the Web Form page, because it is physically in the *.aspx* file. However, you do not see the code, only the results are rendered when the page runs.
The program logic is separated from user interface design code. So it is easy to understand.	Both program logic and user interface design code are placed in the same .aspx file, which makes it cumbersome.

b. **Explain CustomValidator control with example.**
Answer (Explanation & imp properties 3 + Examples 2):

CustomValidator:

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

Below table shows the properties of CustomValidator.

Property	Description
ControlToValidate	The id of the control to validate
Display	• None (the control is not displayed and error message is shown in the ValidationSummary control) • Static (in case of failure error message is shown in the space reserved). • Dynamic (In case of failure error message is shown. Space is not reserved.)
Enabled	A boolean value that specifies whether the validation control is enabled or not
Id	A unique id for the control
IsValid	A Boolean value that indicates whether the control specified by ControlToValidate is determined to be valid
OnServerValidate	Specifies the name of the server-side validation script function to be executed
Text	The message to display when validation fails

Example:

In this below example we will simply check the length of the string in the TextBox.

Code in .aspx file:

```
Enter yourName:<br />
<asp:TextBox runat="server" id="txtCustom" />
<asp:CustomValidator runat="server" id="cusCustom" controltovalidate="txtCustom"
onservervalidate="cusCustom_ServerValidate" errormessage="The text must be exactly 8
characters long!" />
<br /><br />
```

Code in .cs file:

```
protected void cusCustom_ServerValidate(object sender, ServerValidateEventArgs e)
{
  if(e.Value.Length == 8)
     e.IsValid = true;
  else
     e.IsValid = false;
 }
```

c.	**Explain following properties supported by ListBox control** **Answer (First 3 marks+ Selection Mode 2 Marks):** **(i) SelectedIndex:** Gets or sets the zero-based index of the currently selected item in a ListBox. Example: listBox1.SelectedIndex = 1; **(ii) SelectedValue:** Gets the value of the selected item in the list control, or selects the item in the list control that contains the specified value. **Example**: label1.Text = Listbox1.SelectedValue.ToString(); **(iii) Rows:** No. of rows (items) can be set to display in the List. ListBox1.Rows=5 **(iv) SelectionMode:** SelectionMode property defines how items are selected in a ListBox. The SelectionMode value can be one of the following four SelectionMode enumeration values. **None** - No item can be selected. **One** - Only one item can be selected. **MultiSimple** - Multiple items can be selected. **MultiExtended** - Multiple items can be selected, and the user can use the SHIFT, CTRL, and arrow keys to make selections. listBox1.SelectionMode = SelectionMode.MultiSimple; **(v) SelectedItes:** Gets or sets the currently selected item in the ListBox. We can get text associated with currently selected item by using Items property. string selectedItem = listBox1.Items[listBox1.SelectedIndex].ToString();
d.	**What is web.config file? Explain <customErrors> and <connectionStrings> tags in web.config file.** **Answer(1 web.config + 2 <cutomError> + 2 <connectionString>):** **Web.config:** Configuration file is used to manage various settings that define a website. Generally a website contains a single Web.config file stored inside the application root directory. However there can be many configuration files that manage settings at various levels within an application. **<customError>:** You can configure how information is displayed by ASP.NET when unhandled errors occur during the execution of a Web request. You might to do this to provide additional information to the user, or to customize the standard message that is displayed by ASP.NET. When customErrors's mode property is set to On or RemoteOnly, we need to specify the defaultRedirect attribute. This attribute contains the error page to which the user will be redirected. **Example:** <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm"> <error statusCode="403" redirect="NoAccess.htm" /> <error statusCode="404" redirect="FileNotFound.htm" /> </customErrors>

customErrors attribute:

Off Mode

ASP.NET uses its default error page.

On Mode

ASP.NET uses user-defined custom error page instead of its default error.

RemoteOnly

ASP.NET error page is shown only to local users. Remote requests will first check the configuration settings for the custom error page or finally show an IIS error

<connectionStrings>:

It is a used to store the connection string for your application in a config file rather than as a hard coded string in your code**.**

Example:

```
<connectionStrings>
<add name="myConnectionString"
connectionString="server=localhost;database=myDb;uid=myUser;password=myPass;" />
</connectionStrings>
```

To read the connection string into your code, use the ConfigurationManager class.

```
string connStr =
ConfigurationManager.ConnectionStrings["myConnectionString"].ConnectionString;
```

4.	**Attempt *any two* of the following:**
a.	What is state management? Explain QueryString with example.

Answer (StateMgmt 2+ QueryString 2+ Example 1)

State Management:

HTTP is a stateless protocol. Therefore web application is stateless. That means that a new instance of a page is created every time when we make a request to the server to get the page and after the round trip our page has been lost immediately.

State management is the process by which you maintain state and page information over multiple requests for the same or different pages. It is a server side state management technique.

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=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. The following code shows sending multiple values to the foo.aspx page.

```
Response.Redirect("foo.aspx?id=1&name=foo");
```

The following code shows reading the QueryString values in foo.aspx

```
string id = Request.QueryString["id"];
string name = Request.QueryString["name"];
```

b.	**Why UrlEncode() and UrlDecode() methods are used in ASP.NET? Explain.** **Answer:(reason 1 mark + UrlEncode 2 marks + UrlDecode 2 Marks)** We cannot send special characters through query string. All special characters should be encoded when you pass them through the query string. The encoded string must be decoded at the receiver. There are two methods to achieve this: **UrlEncode and UrlDecode()**: The main purpose of these two methods is to encode and decode the **URL** respectively. We need to encode the URL because some of the characters are not safe sending those across browser. Some characters are being misunderstood by the browser and that leads to data mismatch on the receiving end. Characters such as a question mark (?), ampersand (&), slash mark (/), and spaces might be truncated or corrupted by some browsers. **UrlEncode()** This method is used to encode a string to be passed through URL to another web page. URLencode replaces unsafe ASCII characters with a "%" followed by two hexadecimal digits. **Syntax**: UrlEncode (string str) For encoding: URLEncode()method is used. it is the process of converting string into valid URL format . string URLEncode = Server.UrlEncode("Redirect.aspx? Name = jitu&ID = 5"); Response.Write("Encoding URL:-" + URLEncode); //shows like this Redirect.aspx%3f+Name+%3d+jitu%26ID+%3d+5 **UrlDecode()** This method is used to decode the encoded URL string. Decodes any %## encoding in the given string. **Syntax:** UrlDecode (string str) For decoding: URLDecode() method is used. **UrlDecode**:- URL-decodes a string and returns the decoded string. string URLDecode = Server.UrlDecode(URLEncode); Response.Write("Decoding URL:-" + URLDecode); //shows like this Redirect.aspx? Name = jitu&ID = 5
c.	**What is sitemap? Explain TreeView control.** **Answer (sitemap 2 + Tree View 2+ Example/syntax 1):** **Sitemap:** All the navigation controls uses an XML-based file which is known as sitemap. Sitemaps are XML files which are used to describe the logical structure of web application. it is used to define the layout of all pages in web application and how they relate to each other. Whenever you want you can add or remove pages to your site map there by managing navigation of website efficiently. Site map files are defined with .sitemap extension. To add the site map file to your application, right click on the web application in solution explorer and select the menu Add new item. From the add new item window select the file as site map. By default it has a name web. Sitemap keep it as it is.

The root node of sitemap file is <sitemap> element. Only one <sitemap> element can exist in the site map file. Within the <sitemap> element, there is a <siteMapNode> element. This is generally the start page of the application. **<siteMapNode> element have the following attributes** :

Title: It provides a textual description of the link
Description: It is used for ToolTip of the link
URL: It gives the location of the valid physical file

Depending upon the your web application, you can add as many additional <siteMapNode> elements as you want. Additional link-levels can be created by adding child <SiteMapNode> elements for any parent <SiteMapNode> in the structure.

TreeView:

A Tree View control displays a hierarchical list of data. When TreeView is displayed for the first time, it displays all its nodes. You can control it by setting ExpandDepth property.

Following are some important properties of TreeView contro

Properties	Desciption
DataSourceID	It is to specify the data source to be used using sitemap file as datasource.
ShowLines	It is to specify the lines to connect the individual items in the tree.
CssClass	It is to specify CSS class attribute for the control
ExpandDepth	It is to specify level at which items in the tree are expanded

To display TreeView control in all your pages automatically, add it in the master page of the website. To create a TreeView control follows the following steps:

Syntax:

<asp:TreeView ID=”TreeView1” runat=”server’> </asp : TreeView>

d. **Explain CustomValidator control with example.**
Answer (Explanation & properties 3 + Examples 2):

CustomValidator:

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.
The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.
Below table shows the properties of CustomValidator.

Property	Description
ControlToValidate	The id of the control to validate
Display	• None (the control is not displayed and error message is shown in the ValidationSummary control) • Static (in case of failure error message is shown in the space reserved). • Dynamic (In case of failure error message is shown. Space is not reserved.)
Enabled	A boolean value that specifies whether the validation control is enabled or not
Id	A unique id for the control
IsValid	A Boolean value that indicates whether the control specified by ControlToValidate is determined to be valid
OnServerValidate	Specifies the name of the server-side validation script function to be executed
Text	The message to display when validation fails

Example:

In this below example we will simply check the length of the string in the TextBox.

Code in .aspx file:

```
Enter yourName:<br />
<asp:TextBox runat="server" id="txtCustom" />
<asp:CustomValidator runat="server" id="cusCustom" controltovalidate="txtCustom"
onservervalidate="cusCustom_ServerValidate" errormessage="The text must be exactly 8
characters long!" />
<br /><br />
```

Code in .cs file:

```
protected void cusCustom_ServerValidate(object sender, ServerValidateEventArgs e)
{
  if(e.Value.Length == 8)
      e.IsValid = true;
  else
      e.IsValid = false;
}
```

5. Attempt *any two* of the following:

a. What is data-bound control? Explain GridView control with example.

Answer(data-bound 1+ GridView 1+ Properties 1+ Example 1):

Data-bound web server controls are controls that can be bound to a data source control to make it easy to display and modify data in your web application. All of these controls provide a variety of properties that you can set to control the appearance of the UI that they generateData-bound web server controls are composite controls that combine other ASP.NET web controls, such as Label and TextBox controls, into a single layout.

For example, a data-bound control such as a DetailsView control can bind to a set of results such as a table of employees containing each employee's name, address, job title, and so on.

GridView Control

The GridView control displays data as a table and provides the capability to sort columns, page through data, and edit or delete a single record.

Properties:

AllowPaging: Indicate whether the control should support paging.
AllowSorting: Indicate whether the control should support sorting.
DataSourceID: Indicate the bound data source control to use (Generally used when we are using SqlDataSource or AccessDataSource to bind the data.).
PageSize: Total number of records we want in each page.

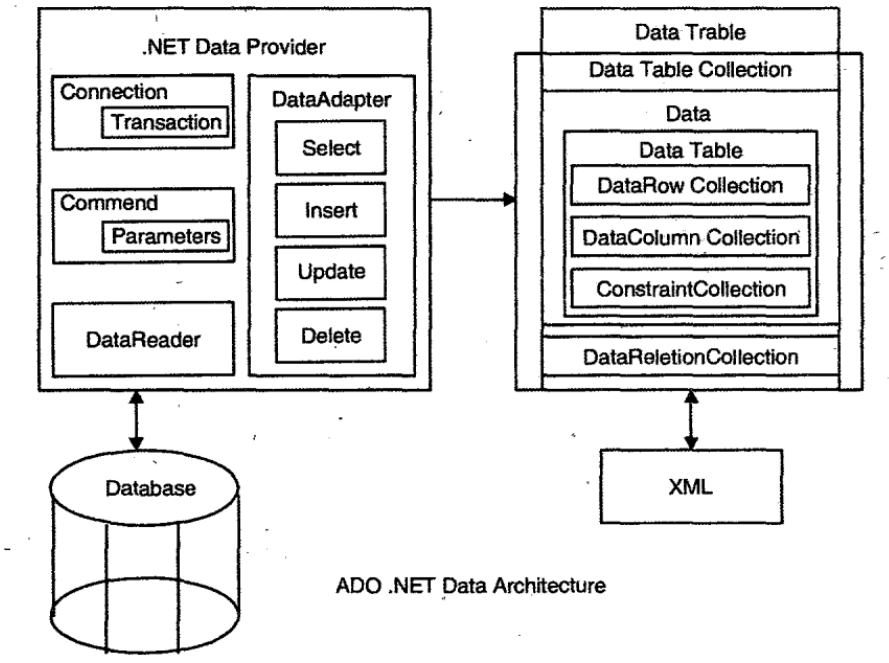
Example:

```
<asp:gridview AllowSorting="true" AllowPaging="true" PageSize="5" ID="Gridview1"
runat="server" DataKeyNames="pid" DataSourceID="SqlDS" >
<Columns>
<asp:BoundField DataField="pname" HeaderText="PRODUCT NAME"
SortExpression="pname"> </asp:BoundField>
</Columns>
</asp:gridview>
```

b. **Differentiate between DataSet and DataReader objects.**
Answer(1 mark for each point):

DataSet	**DataReader**
Works in Disconnected Mode	Works in Connected Mode
Can navigate back and forth	Can navigate forward only
Data is editable	Data is Readonly
Can contain more than one table and relationships	Can contain only one row at a time
Slower as having more overhead	Faster when compared with DataSet
Example: DataSet ds = new DataSet()	**Example:** DataReader dr = new DataReader();

c. **Draw and explain the architecture of ADO.NET in brief.**
Answer(Diagram 3 Marks+ Explanation 2):



ADO .NET Data Architecture

d. **What is impersonation in ASP.NET? Explain windows authentication.**

Impersonation:

Impersonation is the process of executing code in the framework of another user identity. By default all ASP.NET code is executed using a fixed machine-specific account. To execute code using another identity you can use the built-in impersonation capabilities of ASP.NET. Impersonation is a technique that allows the ASP.NET process act as the authenticated user or as an arbitrary specified user.

ASP.NET impersonation is controlled by <identity> tag in the applications web.config file. The default setting is impersonation as false. You also can explicitly specify that

ASP.NET should not use impersonation by including the following code in file:

```
<identity impersonate="false"/>
```

Now ASP.NET does not perform impersonation. It means that ASP.NET will runs with its own privileges. The second possible setting is to turn on impersonation.

```
<identity impersonate="true"/>
```

Now ASP.NET takes on the identity IIS passes to it. By turning impersonation on and using a non-anonymous method of authentication in IIS, you can let users log on and use their identities within your ASP.NET application.

To impersonate a specific user for all the requests on all pages of an ASP.NET application, you can specify the userName and password attributes in the <identity> tag of the Web.config file for that application as follows:

```
<identity impersonate='true" username="DOMAIN\username" password="password"/>
```

Windows authentication:

The Windows authentication provider is the default provider for ASP .NET. It authenticates the users based on the users Windows accounts. Windows authentication in ASP.NET actually relies on IIS to do the authentication. IIS be configured so that only users on a Windows domain can log in. if a user attempts access a page and is not authenticated then user will be shown a dialog box asking them to enter their username and password. This information is then passed to the Web server and checked against the list of users in the domain. if the user has supplied valid credentials access is granted.

To configure ASP.NET to use Windows authentication use the following code:

```
<system. web>
<authentication mode="Windows"/>
<authorization>
<allow users"*"/>
</authorization>
</system.web>
<allow users="*"/>
```

Statement specifies permissions are provided to authorized users as well anonymous users. There are four different kinds of Windows authentication options available that can configured in IIS and they are given bellow:

6.	**Attempt *any two* of the following:**
a.	**Explain the syntax for query expression in LINQ with example.** **Answer(3 Marks Syntax and explanation + 2 Marks Example):** **Query Expression:** A *query expression* is a query expressed in query syntax. A query expression is a first-class language construct. It is just like any other expression and can be used in any context in which a C# expression is valid. A query expression consists of a set of clauses written in a declarative syntax similar to SQL. Each clause in turn contains one or more C# expressions, and these expressions may themselves be either a query expression or contain a query expression. **SYNTAX:** from [identifier] in [source collection] let [expression] where [Boolean expression] order by [[expression](ascending/descending)], [optionally repeat] select [expression] group [expression] by [expression] into [expression] **Where,** from / in - Specifies the data source where - Conditional Boolean expression order by (ascending/descending) - Sorts the results into ascending or descending order select - Adds the result to the return type group / by - Groups the results based on a given key **Example:** static void Main() { // Data source. int[] scores = { 90, 71, 82, 93, 75, 82 }; // Query Expression. IEnumerable<int> scoreQuery = //query variable from score in scores //required where score > 80 // optional orderby score descending // optional select score; //must end with select or group // Execute the query to produce the results foreach (int testScore in scoreQuery) { Console.WriteLine(testScore); } } **// Outputs:** 93 90 82 82
b.	**What is jQuery selector? Write some examples.** **Answer(2 Explanation + 1 each selector):** **jQuery Selector:** jQuery Selector is used to select one or a group of HTML elements from your web page. jQuery support all the CSS selectors as well as many additional custom selectors. jQuery selectors always start with dollar sign and parentheses: $()

There are three building blocks to select the elements in a web document.

a) Select elements by tag name
Example: $(div)
It will select all the div elements in the document.

b) Select elements by ID
Example: $(#xyzid")
It will select single element that has an ID of xyzid

c) Select elements by class
Example: $(".xyzclass")
It will select all the elements having class xyzclass

c. **Explain Timer and UpdatePanel controls.**

The Timer Control

1. The Timer control that you find in the AJAX Extensions category of the Toolbox is great for executing server-side code on a repetitive basis.
2. For example, you can use it to update the contents of an UpdatePanel every 5 seconds.
3. The contents of this UpdatePanel could come from a variety of sources, such as a database with the latest forum posts on a forum or news items on a news site, an XML file with information to rotate advertisements in the browser, stock quotes from a stock web service, and more.
4. At a specified interval, the control fires its Tick event. Inside an event handler for this event you can execute any code you see fit.
5. The following code snippet shows the markup for a simple UpdatePanel and a Timer control that you can place inside a content page based on your master page (because the master page already contains the required ScriptManager):

```
<asp:UpdatePanel 1D"UpdatePanel runat"server"> <ContentTemplate>
<asp :Label ID="Labell" runat="server"></asp:Label>
<asp:Timer lD"Timerl" runat="server" Interval=" 5000" OnTick="Timerl_Tick"/>
</ContentTemplate> </asp:UpdatePanel>
```

When the timer "ticks" it raises **its Tick event, which you can handle with the following code:**

```
protected void Timerl_Tick(object sender, EventArgs e)
{
Labell.Text System.DateTime.Now.ToString();
}
```

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.

d. **What is LINQ? What are advantages and disadvantages of LINQ?**
Answer: (1 mark for LINQ + 2 marks for adv + 2 for disad)

LINQ Syntax: Language-Integrated Query
LINQ query syntax is a set of query keywords built into the .NET framework (3.5 and higher) that allow the developer to write SQL style commands in line straight in the code editor, without the use of quotes

Advantages:

The availability of strong typed queries: The classes are auto generated according to the relations in relational databases. The language is very much easy to understand as in SQL.

The automatic join functioning in foreign keys: In normal SQL, the user has to join the tables if it is necessary. In LINQ, it provides the ability to join each function automatically when there is a foreign key reference.

The code size: There are many occasions that the users have to write long sentences for getting a SQL query. LINQ provides relatively short codes in such advanced occasions. It reduces the complexity of the code and makes it much easy for the program to read.

Code equality: One of the most advantages in using LINQ is that its availability over any .NET platform language such as C#.net, VB.NET and F#.NET.

Disadvantages:

- There is an overhead for creating queries
- When queries are moved from sql to application side, joins are very slow
- DBML concurrency issues
- Hard to understand advance queries using Expressions

7. **Attempt *any three* of the following:**

a. **What is constructor? Explain parameterized constructor with suitable example.**

Answer (Constructor 1 mark+ Parameterized 2+ Example 2):

Constructor:

Constructor is a special method of the class that will be automatically invoked when an instance of the class is created is called a constructor. The main use of constructors is to initialize private fields of the class while creating an instance for the class. When you have not created a constructor in the class, the compiler will automatically create a default constructor in the class. The default constructor initializes all numeric fields in the class to zero and all string and object fields to null.

Parameterized Constructor

A constructor with at least one parameter is called a parametrized constructor. The advantage of a parametrized constructor is that you can initialize each instance of the class to different values.

```
using System;
namespace Constructor
{
   class paraconstrctor
   {
     public  int a, b;
     public paraconstrctor(int x, int y)  // decalaring Paremetrized Constructor with ing x,y parameter
      {
         a = x;
         b = y;
      }
   }
   class MainClass
   {
      static void Main()
```

```
        {
            paraconstrctor v = new paraconstrctor(100, 175);   // invoking constructor
            Console.WriteLine("value of a=" + v.a );
            Console.WriteLine("value of b=" + v.b);
            Console.Read();
        }
    }
}
```

Output:
value of a=100
value of b=175

b. **What is delegate? Explain the steps to implement delegates in C#.NET.**
Answer(delegate def 1 + Steps 1 for each):

Delegate:
The dictionary meaning of **'Delegate'** is **"A person acting for another person".**
A delegate in C# is a class type object & is used to invoke a method that has been encapsulated into it at the time of its creation.

Creating & using delegates involves four steps
(i) Delegate declaration
(ii) Delegate methods definition
(iii) Delegate instantiation
(iv) Delegate invocation

c. **Explain following properties**

(i) AutoPostBack

AutoPostBack means that if any change happens on a control by the client, it should postback to the server to handle that change at server side. If this property is set to TRUE the automatic post back is enabled, otherwise FALSE. Default is FALSE.

Example

```
<asp:TextBox id= " t1" AutoPostBack="true" runat="server" />
```

Web server controls are created on the server and they require a runat="server" attribute to work. This attribute indicates that the control should be treated as a server control.

(ii) AutoEventWiredUp

AutoEventWireup is an attribute in Page directive. It is a Boolean attribute that indicates whether the ASP.NET pages events are auto-wired. By default it is true.

Example:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
```

d. **Explain the relationship between master page and its content page with example.**
Answer:

Every Master Page should include at least one ContentPlaceHolder control, this is a placeholder for the content from the content page.

<asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server"> </asp : ContentPlaceHolder>

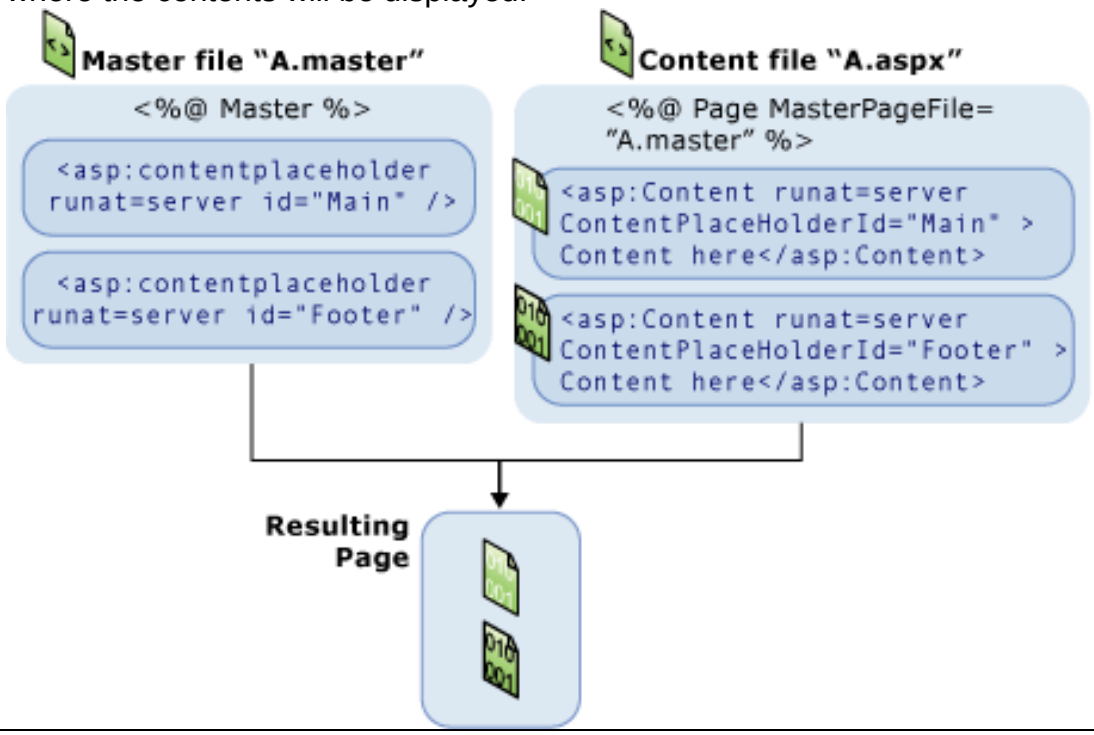
The content page is a standard aspx page in which MasterPageFile attribute should be added to the page directive for binding the content page to the master page. The content pages can not include the common tags as <Body>, <Head> etc. You can design your content page just like any other aspx page by adding static text and server controls.

<%@ Page Language = "C#" MasterPageFile = ".../master/MasterPage.master" AutoEventWireup = "true" CodeFile="Default2.aspx.cs" Inherits="Default2" Title="Untitled Page" %>

Inside the content pages by default Content server control is get added. While designing the, content page you have to add controls to the content server control For example,

<asp: Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
</asp : Content>

The attribute ContentPlaceHolderID is very important as it decides what content will be bound to which ContentPlaceHolder on the master page. It is way to decide the location of where the contents will be displayed.



e. **Explain SqlDataAdapter and SqlConnection objects in ADO.NET.**

SQlDataAdapter:

A DataAdapter bridges the gap between the disconnected DataTable objects and the physical data source. The SqlDataAdapter is capable of executing a SELECT, DELETE and UPDATE statement on a data source as well as extracting input from the result set into a DataTable object. The SqlDataAdapter class provides a method called **Fill()** to copy the result set into the DataTable.

Syntax:

SqlDataAdapter da = new SqlDataAdapter();

DataAdapter class properties and methods

Property	Description
DeleteCommand	Speicfies the Command object with the SQL command to be used for deleting a record.
FillCommandBehavior	Specifies the behavior of the command used to fill the data adapter.
InsertCommand	Specifies the Command object with the SQL command yused for inserting a record.
SelectCommand	Specifies the Command object with the SQL command to be used for inserting a record.
UpdateBatchSize	Specifies the number of commands that can be executed as a batch.
UpdateCommand	Specifies the Command object with the SQL command to be used for inserting a record.

Method	Description
AddToBatch	Adds a Command object to a batch that will be executed.
ClearBatch	Removes all the Command objects from the batch.
ExecuteBatch	Excutes the batch of commands.
Fill	Executes the SelectCommand property and fills a DataSet that is provided.
InitializeBatching	Initialize the batching process.
GetFillParameters	Gets the paremeters of the SelectCommand property.
TerminateBatching	Terminates the batching process.
Update	Calls the corresponding INSERT, DELETE, or UPDATE command of this DataAdapterand updates the data source using the specified commands.

SqlConnection Object

To communicate with any data base, you must have to first connect to it. The connection identifies the data base server, then, the data base name, user name, password, and other parameters that are required for connecting to the data base. This connection object is then Command object so as to identify the database needed for execution of command.

Properties:

Property	Description
ConnectionString	It provides information about the data source and the database name.
State	To determine the current state of a given object at any time.

Example:

SqlConnection con = new SqlConnection("Data Source=Your data source name;Initial Catalog=Demo;Integrated Security=True");

f.	**What are different types of operators in LINQ?** **Answer(each type 1 mark)** **Filtering-**where, OfType etc. **Join**-Join, GroupJoin etc. **Projection-**Select, SelectMany etc. **Sorting-**OrderBy, OrdeByDescending, Reverse etc. **Group-**GroupBy, ToLookup etc. **Aggregation-**Aggregate, Count, min, max, sum etc. **AsEnumerable,** CastToArry, ToList etc. **Concatenation**-Concat etc. **Set** – All, any etc. **Projection**-Distinct, Union etc.