Q.1

1.**BCL**:-

 A ASP.NET Framework library, BCL is the standard for the C# runtime library and one of the Common Language Infrastructure (CLI) standard libraries. BCL provides types representing the built-in CLI data types, basic file access, collections, custom attributes, formatting, security attributes, I/O streams, string manipulation, and more.

 The Base Class Library (BCL) is the core set of classes that serve as the basic API of the Common Language Runtime. The classes in mscorlib.dll and some of the classes in System.dll and System.core.dll are considered to be a part of the BCL.

2. B**ase keyword:-**

  The **base keyword** is used to access members of the **base** class from within a derived class: Call a method on the **base** class that has been overridden by another method. Specify which **base**-class constructor should be called when creating instances of the derived class

3.GC (garbage collector):-

 The C# language is a garbage-collected language. This means that memory that is no longer referenced by your program will be reclaimed and is later reused. With GC.Collect, we force a garbage collection to occur at any time.

**Example.** First, this example invokes the GC.Collect method. Three calls to get the total memory usage on the system are present. They occur before the allocation, after the allocation, and after the forced garbage collection.

**Tip:**You can see that memory returns to its low level after the garbage collection.

**Based on:** .NET (2018)

**C# program that uses GC.Collect**

using System;

class Program

{

 static void Main()

 {

 long mem1 = GC.GetTotalMemory(false);

 {

 // Allocate an array and make it unreachable.

 int[] values = new int[50000];

 values = null;

 }

 long mem2 = GC.GetTotalMemory(false);

 {

 // Collect garbage.

 **GC.Collect**();

 }

 long mem3 = GC.GetTotalMemory(false);

 {

 Console.WriteLine(mem1);

 Console.WriteLine(mem2);

 Console.WriteLine(mem3);

 }

 }

}

**Output**

45664

245696

33244

4. MSIL:-

5 CTS in C#?

 In Microsoft's .NET Framework, the Common Type System (CTS) is a standard that specifies how type definitions and specific values of types are represented in computer memory. It is intended to allow programs written in different programming languages to easily share information.

 The Common Type System (CTS) standardizes the data types of all programming languages using .NET under the umbrella of .NET to a common data type for easy and smooth communication among these .NET languages.

**How CTS converts the data type to a common data type**

To implement or see how CTS is converting the data type to a common data type, for example, when we declare an int type data type in C# and VB.Net then they are converted to int32. In other words, now both will have a common data type that provides flexible communication between these two languages.



Q.2

i. **WAP to add an item in Dropdown List**

 **HTML Markup**

The HTML Markup consists of TextBox, Button and a DropDownList.

<asp:TextBox ID="txtFruit" runat="server" />

<asp:Button ID="btnAdd" runat="server" Text="Add Item" OnClick="AddItem" />

<br />

<br />

<asp:DropDownList ID="ddlFruits" runat="server">

    <asp:ListItem Text="----Fruits----" Value=""></asp:ListItem>

</asp:DropDownList>

**Programmatically adding item to ASP.Net DropDownList control from code behind**

Below is the Button click event handler, when the Button is clicked the name of the Fruit entered in the TextBox is stored in a string variable.

Then if the value is not NULL, it is added to the DropDownList using the DropDownList Items Add method which accepts object of type ListItem class.

**C#**

protected void AddItem(object sender, EventArgs e)

{

    string fruit = txtFruit.Text.Trim();

    if (!string.IsNullOrEmpty(fruit))

    {

        ddlFruits.Items.Add(new ListItem(fruit, fruit));

    }

}

**VB.Net**

Protected Sub AddItem(sender As Object, e As EventArgs)

    Dim fruit As String = txtFruit.Text.Trim()

    If Not String.IsNullOrEmpty(fruit) Then

        ddlFruits.Items.Add(New ListItem(fruit, fruit))

    End If

End Sub



ii. Define properties used in c#

 **Properties** are named members of classes, structures, and interfaces. Member variables or methods in a class or structures are called **Fields**. Properties are an extension of fields and are accessed using the same syntax. They use **accessors** through which the values of the private fields can be read, written or manipulated.

An example, which uses a set of set/get methods, is shown below.

//SET/GET methods
//Author: rajeshvs@msn.com
using System;
class MyClass
{
private int x;
public void SetX(int i)
{
x = i;
}
public int GetX()
{
return x;
}
}
class MyClient
{
public static void Main()
{
MyClass mc = new MyClass();
mc.SetX(10);
int xVal = mc.GetX();
Console.WriteLine(xVal);//Displays 10
}
}

But C# provides a built in mechanism called properties to do the above. In C#, properties are defined using the property declaration syntax. The general form of declaring a property is as follows.

<acces\_modifier> <return\_type> <property\_name>
{
get
{
}
set
{
}
}

Where <access\_modifier> can be private, public, protected or internal. The <return\_type> can be any valid C# type. Note that the first part of the syntax looks quite similar to a field declaration and second part consists of a get accessor and a set accessor.

For example the above program can be modifies with a property X as follows.

class MyClass
{
private int x;
public int X
{
get
{
return x;
}
set
{
x = value;
}
}
}

The object of the class MyClass can access the property X as follows.

MyClass mc = new MyClass();

mc.X = 10; // calls set accessor of the property X, and pass 10 as value of the standard field 'value'.
This is used for setting value for the data member x.
Console.WriteLine(mc.X);// displays 10. Calls the get accessor of the property X.

The complete program is shown below.

//C#: Property
//Author: rajeshvs@msn.com
using System;
class MyClass
{
private int x;
public int X
{
get
{
return x;
}
set
{
x = value;
}
}
}
class MyClient
{
public static void Main()
{
MyClass mc = new MyClass();
mc.X = 10;
int xVal = mc.X;
Console.WriteLine(xVal);//Displays 10
}
}

Remember that a property should have at least one accessor, either set or get. The set accessor has a free variable available in it called value, which gets created automatically by the compiler. We can't declare any variable with the name value inside the set accessor.

We can do very complicated calculations inside the set or get accessor. Even they can throw exceptions.

Since normal data fields and properties are stored in the same memory space, in C#, it is not possible to declare a field and property with the same name.

**Static Properties**

C# also supports static properties, which belongs to the class rather than to the objects of the class. All the rules applicable to a static member are applicable to static properties also.

The following program shows a class with a static property.

//C# : static Property
//Author: rajeshvs@msn.com
using System;
class MyClass
{
private static int x;
public static int X
{
get
{
return x;
}
set
{
x = value;
}
}
}
class MyClient
{
public static void Main()
{
MyClass.X = 10;
int xVal = MyClass.X;
Console.WriteLine(xVal);//Displays 10
}
}

Remember that set/get accessor of static property can access only other static members of the class. Also static properties are invoking by using the class name.

**Properties & Inheritance**

The properties of a Base class can be inherited to a Derived class.

//C# : Property : Inheritance
//Author: rajeshvs@msn.com
using System;
class Base
{
public int X
{
get
{
Console.Write("Base GET");
return 10;
}
set
{
Console.Write("Base SET");
}
}
}
class Derived : Base
{
}
class MyClient
{
public static void Main()
{
Derived d1 = new Derived();
d1.X = 10;
Console.WriteLine(d1.X);//Displays 'Base SET Base GET 10'
}
}

The above program is very straightforward. The inheritance of properties is just like inheritance any other member.

**Properties & Polymorphism**

A Base class property can be polymorphicaly overridden in a Derived class. But remember that the modifiers like virtual, override etc are using at property level, not at accessor level.

//C# : Property : Polymorphism
//Author: rajeshvs@msn.com
using System;
class Base
{
public virtual int X
{
get
{
Console.Write("Base GET");
return 10;
}
set
{
Console.Write("Base SET");
}
}
}
class Derived : Base
{
public override int X
{
get
{
Console.Write("Derived GET");
return 10;
}
set
{
Console.Write("Derived SET");
}
}
}
class MyClient
{
public static void Main()
{
Base b1 = new Derived();
b1.X = 10;
Console.WriteLine(b1.X);//Displays 'Derived SET Derived GET 10'
}
}
 **Abstract Properties**

A property inside a class can be declared as abstract by using the keyword abstract. Remember that an abstract property in a class carries no code at all. The get/set accessors are simply represented with a semicolon. In the derived class we must implement both set and get assessors.

If the abstract class contains only set accessor, we can implement only set in the derived class.

The following program shows an abstract property in action.

//C# : Property : Abstract
//Author: rajeshvs@msn.com
using System;
abstract class Abstract
{
public abstract int X
{
get;
set;
}
}
class Concrete : Abstract
{
public override int X
{
get
{
Console.Write(" GET");
return 10;
}
set
{
Console.Write(" SET");
}
}
}
class MyClient
{
public static void Main()
{
Concrete c1 = new Concrete();
c1.X = 10;
Console.WriteLine(c1.X);//Displays 'SET GET 10'
}
}

The properties are an important features added in language level inside C#. They are very useful in GUI programming. Remember that the compiler actually generates the appropriate getter and setter methods when it parses the C# property syntax.

Q.3 client server architecture

 A client/server system operates as outlined in the following diagram:  The client sends a request to the server using its IP address and the port, which is reserved for a particular service running on the server. The server receives the request and responds using the client IP address and port



**2 -TIER ARCHITECTGURE**

 2-tier architecture is used to describe client/server systems where the client requests resources and the server responds directly to the request, using its own resources. This means that the server does not call on another application in order to provide part of the service.



**3-TIER ARCHITECTURE**

 In 3-tier architecture, there is an intermediary level, meaning the architecture is generally split up between:

1. A client, i.e. the computer, which requests the resources, equipped with a user interface (usually a web browser) for presentation purposes

2. The application server (also called **middleware**), whose task it is to provide the requested resources, but by calling on another server

3. The data server, which provides the application server with the data it requires



Q.3 ii Name spaces:-

 NameSpace is the Logical group of types or we can say namespace is a container (e.g Class, Structures, Interfaces, Enumerations, Delegates etc.), example System.IO logically groups input output related features, System.Data.SqlClient is the logical group of ado.net Connectivity with Sql server related features. In Object Oriented world, many times it is possible that programmers will use the same class name, Qualifying NameSpace with class name can avoid this collision.

**Example :-**

// Namespace Declaration
using System;

// The Namespace
namespace MyNameSpace
{
    // Program start class
    class MyClass
    {
       //Functionality
    }
}

*Namespaces* allow you to create a system to organize your code. A good way to organize your *namespaces* is via a hierarchical system. You put the more general names at the top of the hierarchy and get more specific as you go down. This hierarchical system can be represented by nested *namespaces*. Bellow shows how to create a nested*namespace*. By placing code in different sub-namespaces, you can keep your code organized.

// Namespace Declaration
using System;

// The Namespaces
namespace MyNameSpace
{
    namespace Video
    {
        // Program start class
        class MyVideo
        {
            //Functionality

           public static void play()

           {
              //..
           }

        }
    }

    namespace Audio
    {
        // Program start class
        class MyAudio
        {
            //Functionality
           public static void play()

           {
               //..

           }

        }
    }
}

Now a question arise on our mind that how to call the namespace members so there is two way to call namespacemembers By typing a fully qualified name, Or by implement the *using* directive.

Here i provides an example of how to call *namespace* members with fully qualified names. A fully qualified name contains every language element from the *namespace* name down to the method call. At the top of the listing there is a nested *namespace* *tutorial* within the **MyNameSpace***namespace* with *class* **Video, Audio** and method ***play()***.Here we calls this method with the fully qualified name :

MyNameSpace*.*Video.MyVideo*.play()*

MyNameSpace*.*Audio*.*MyAudio*.play()*
If we call play method in the same namespace then no need to include MynameSpace at the begin e.g. Video*.play()*.

Now implement this by *using* directive

using MyNameSpace.Video;
using MynameSpace.Audio;

public void call()

{

  Video*.play()*

}

If our method would not be static then make instance then call by the object of the class.

Q.4 All Standard Controls:

 Button Controls

ASP.NET provides three types of button control:

* **Button** : It displays text within a rectangular area.
* **Link Button** : It displays text that looks like a hyperlink.
* **Image Button** : It displays an image.

When a user clicks a button, two events are raised: Click and Command.

Basic syntax of button control:

<asp:Button ID="Button1" runat="server" onclick="Button1\_Click" Text="Click" / >

Common properties of the button control:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Text | The text displayed on the button. This is for button and link button controls only. |
| ImageUrl | For image button control only. The image to be displayed for the button. |
| AlternateText | For image button control only. The text to be displayed if the browser cannot display the image. |
| CausesValidation | Determines whether page validation occurs when a user clicks the button. The default is true. |
| CommandName | A string value that is passed to the command event when a user clicks the button. |
| CommandArgument | A string value that is passed to the command event when a user clicks the button. |
| PostBackUrl | The URL of the page that is requested when the user clicks the button. |

Text Boxes and Labels

Text box controls are typically used to accept input from the user. A text box control can accept one or more lines of text depending upon the settings of the TextMode attribute.

Label controls provide an easy way to display text which can be changed from one execution of a page to the next. If you want to display text that does not change, you use the literal text.

Basic syntax of text control:

<asp:TextBox ID="txtstate" runat="server" ></asp:TextBox>

Common Properties of the Text Box and Labels:

|  |  |
| --- | --- |
| **Property** | **Description** |
| TextMode | Specifies the type of text box. SingleLine creates a standard text box, MultiLIne creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine. |
| Text | The text content of the text box. |
| MaxLength | The maximum number of characters that can be entered into the text box. |
| Wrap | It determines whether or not text wraps automatically for multi-line text box; default is true. |
| ReadOnly | Determines whether the user can change the text in the box; default is false, i.e., the user can not change the text. |
| Columns | The width of the text box in characters. The actual width is determined based on the font that is used for the text entry. |
| Rows | The height of a multi-line text box in lines. The default value is 0, means a single line text box. |

The mostly used attribute for a label control is 'Text', which implies the text displayed on the label.

Check Boxes and Radio Buttons

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form, then specify a different group name for each group.

If you want check box or radio button to be selected when the form is initially displayed, set its Checked attribute to true. If the Checked attribute is set to true for multiple radio buttons in a group, then only the last one is considered as true.

Basic syntax of check box:

<asp:CheckBox ID= "chkoption" runat= "Server">

</asp:CheckBox>

Basic syntax of radio button:

<asp:RadioButton ID= "rdboption" runat= "Server">

</asp: RadioButton>

Common properties of check boxes and radio buttons:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Text | The text displayed next to the check box or radio button. |
| Checked | Specifies whether it is selected or not, default is false. |
| GroupName | Name of the group the control belongs to. |

List Controls

ASP.NET provides the following controls

* Drop-down list,
* List box,
* Radio button list,
* Check box list,
* Bulleted list.

These control let a user choose from one or more items from the list. List boxes and drop-down lists contain one or more list items. These lists can be loaded either by code or by the ListItemCollection editor.

Basic syntax of list box control:

<asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True" OnSelectedIndexChanged="ListBox1\_SelectedIndexChanged">

</asp:ListBox>

Basic syntax of drop-down list control:

<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True" OnSelectedIndexChanged="DropDownList1\_SelectedIndexChanged">

</asp:DropDownList>

Common properties of list box and drop-down Lists:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Items | The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection. |
| Rows | Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added. |
| SelectedIndex | The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1. |
| SelectedValue | The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string (""). |
| SelectionMode | Indicates whether a list box allows single selections or multiple selections. |

Common properties of each list item objects:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Text | The text displayed for the item. |
| Selected | Indicates whether the item is selected. |
| Value | A string value associated with the item. |

It is important to notes that:

* To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.
* The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

The ListItemCollection

The ListItemCollection object is a collection of ListItem objects. Each ListItem object represents one item in the list. Items in a ListItemCollection are numbered from 0.

Common properties of ListItemCollection:

|  |  |
| --- | --- |
| **Property** | **Description** |
| Item(integer) | A ListItem object that represents the item at the specified index. |
| Count | The number of items in the collection. |

Common methods of ListItemCollection:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| Add(string) | Adds a new item at the end of the collection and assigns the string parameter to the Text property of the item. |
| Add(ListItem) | Adds a new item at the end of the collection. |
| Insert(integer, string) | Inserts an item at the specified index location in the collection, and assigns string parameter to the text property of the item. |
| Insert(integer, ListItem) | Inserts the item at the specified index location in the collection. |
| Remove(string) | Removes the item with the text value same as the string. |
| Remove(ListItem) | Removes the specified item. |
| RemoveAt(integer) | Removes the item at the specified index as the integer. |
| Clear | Removes all the items of the collection. |
| FindByValue(string) | Returns the item whose value is same as the string. |
| FindByValue(Text) | Returns the item whose text is same as the string. |

Radio Button list and Check Box list

A radio button list presents a list of mutually exclusive options. A check box list presents a list of independent options. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax of radio button list:

<asp:RadioButtonList ID="RadioButtonList1" runat="server" AutoPostBack="True"

 OnSelectedIndexChanged="RadioButtonList1\_SelectedIndexChanged">

</asp:RadioButtonList>

Basic syntax of check box list:

<asp:CheckBoxList ID="CheckBoxList1" runat="server" AutoPostBack="True"

 OnSelectedIndexChanged="CheckBoxList1\_SelectedIndexChanged">

</asp:CheckBoxList>

Common properties of check box and radio button lists:

|  |  |
| --- | --- |
| **Property** | **Description** |
| RepeatLayout | This attribute specifies whether the table tags or the normal html flow to use while formatting the list when it is rendered. The default is Table. |
| RepeatDirection | It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical. |
| RepeatColumns | It specifies the number of columns to use when repeating the controls; default is 0. |

Bulleted lists and Numbered lists

The bulleted list control creates bulleted lists or numbered lists. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

<asp:BulletedList ID="BulletedList1" runat="server">

</asp:BulletedList>

Common properties of the bulleted list:

|  |  |
| --- | --- |
| **Property** | **Description** |
| BulletStyle | This property specifies the style and looks of the bullets, or numbers. |
| RepeatDirection | It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical. |
| RepeatColumns | It specifies the number of columns to use when repeating the controls; default is 0. |

HyperLink Control

The HyperLink control is like the HTML <a> element.

Basic syntax for a hyperlink control:

<asp:HyperLink ID="HyperLink1" runat="server">

 HyperLink

</asp:HyperLink>

It has the following important properties:

|  |  |
| --- | --- |
|  |  |
| ImageUrl | Path of the image to be displayed by the control. |
| NavigateUrl | Target link URL. |
| Text | The text to be displayed as the link. |
| Target | The window or frame which loads the linked page. |

Image Control

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

Basic syntax for an image control:

<asp:Image ID="Image1" runat="server">

It has the following important properties:

|  |  |
| --- | --- |
|  |  |
| AlternateText | Alternate text to be displayed in absence of the image. |
| ImageAlign | Alignment options for the control. |
| ImageUrl | Path of the image to be displayed by the control. |

ii WAP to define the login

**Registration Form**

**Step 1** **Create Table**

Now create a table in a SQL Server database with username, password and email fields. The table looks like this.

1. **create** **table** registrationtab
2. (
3. Username **varchar**(100), Email **varchar**(100), **Password** **varchar**(20)
4. )

**Step 2** **Create stored procedure**

Now create a stored procedure for registration. That means the values will be inserted into the table using the stored procedure. The stored procedure looks like:

1. **create** **procedure**[dbo].[storlogin134]
2. (
3. @username **varchar**(40), @email **varchar**(50), @**password** **varchar**(20)
4. )
5. **as**
6. **insert** **into** registrationtab **values**(@username, @email, @**password**)

**Step 3 Create form for registration**

Now create a form in ASP.Net with the following fields defined in the table. The form looks like the following figure.



Figure1

Now double-click on the register me button and add following code

1. **protected** **void** Buttonregisterme\_Click(object sender, EventArgs e) {
2. string strcon = "Data Source=.;uid=sa;pwd=Password$2;database=master";
3. SqlConnection con = **new** SqlConnection(strcon);
4. SqlCommand com = **new** SqlCommand("storlogin134", con);
5. com.CommandType = CommandType.StoredProcedure;
6. SqlParameter p1 = **new** SqlParameter("username", TextBoxusername.Text);
7. SqlParameter p2 = **new** SqlParameter("email", TextBoxemail.Text);
8. SqlParameter p3 = **new** SqlParameter("password", TextBoxpassword.Text);
9. com.Parameters.Add(p1);
10. com.Parameters.Add(p2);
11. com.Parameters.Add(p3);
12. con.Open();
13. com.ExecuteNonQuery();
14. Labelinfo.Text = "registered successful.";
15. }

Now run the application and enter the username, email and password and then click on the register me button to save the values to the database.



Figure 2

Open the database and check the registrationtab table.

**Login form**

**Step 4** **Create stored procedure**

Now create a stored procedure for login. That means the values will be selected from the table using a stored procedure. The stored procedure looks like

1. **create** **PROCEDURE** CheckUser
2. (
3. @username **as** **varchar**(50), @**password** **as** **varchar**(50)
4. )
5. **AS**
6. **SELECT** \* **FROM** registrationtab **WHERE** username = @username AND **password** = @**password**

Now create a form in ASP.Net with the following fields defined in the table. The form looks like the following figure.



Figure3

**Step 5 Create form for login**

Now create a form in ASP.Net with the username and password field which are defined in the table. The form looks like the following figure.

Figure 4

Now double-click on the login button and add the following code.

1. **protected** **void** Buttonlogin\_Click(object sender, EventArgs e) {
2. string strcon = "Data Source=.;uid=sa;pwd=Password$2;database=master";
3. SqlConnection con = **new** SqlConnection(strcon);
4. SqlCommand com = **new** SqlCommand("CheckUser", con);
5. com.CommandType = CommandType.StoredProcedure;
6. SqlParameter p1 = **new** SqlParameter("username", TextBoxusername.Text);
7. SqlParameter p2 = **new** SqlParameter("password", TextBoxpassword.Text);
8. com.Parameters.Add(p1);
9. com.Parameters.Add(p2);
10. con.Open();
11. SqlDataReader rd = com.ExecuteReader();
12. **if** (rd.HasRows) {
13. rd.Read();
14. Labelinfo.Text = "Login successful.";
15. } **else** {
16. Labelinfo.Text = "Invalid username or password.";
17. }
18. }

Now run the application and enter the username and password and then click on the login button to retrieve the values from the database. Now suppose we enter the wrong username and password, as in,



Figure4

Now enter the correct username and password,

