Before dot net (1990-1998) the software platform are ------

1. **VB:** Visual basic also known as Visual Basic Classic is introduced in 1991; it is considered the third generation of event-driven programming languages and integrated development environment (IDE). VB derived from the BASIC programming language and considered to event-driven and object-oriented.

Visual Basic use for ----

1. Rapid application development (RAD) of graphical user interface (GUI) applications.
2. Access to databases using Data Access Objects, Remote Data Objects, or ActiveX Data Objects.
3. Creation of ActiveX controls and objects.
4. **VC++:** Visual C++ was Microsoft's implementation of a professional Windows hosted IDE for developing Windows software. Visual C++ 1.0 was the first release of Visual C++, released in 1993 for 16-bit development.
5. **ASP:** ASP stands for Active Server Pages, which known as classic ASP. Classic ASP is a server-side scripting environment that you can use to create and run dynamic web applications. With ASP, you can combine HTML pages, script commands, and COM components to create interactive web pages that are easy to develop and modify. Classic ASP is the predecessor to ASP.NET, but it is still in wide use today.

# .NET

1. After coming java as platform independent Microsoft coming with .Net framework.
2. .Net will support for multiple programming language.
3. .net framework can be written as VB, C#, Jscript various language.

## Components of .NET Framework

1. CLR (Common Language Runtime)
2. CTS (Common Type System)
3. BCL (Base Class Library)
4. CLS (Common Language Specification)
5. FCL (Framework Class Library)
6. .NET Assemblies
7. XML Web Services
8. Window Services

## Version of .NET:

1. Version- 1.0 (2002):

* Initial release.
* Introduced CLR 1.0
* Use of DLLs as class libraries.
* OOP support for Web development.

1. Version- 1.1 (2003):

* Added **ASP.NET** Control support for Mobile device development.
* Added support for **ADO.NET** classes for Oracle database and ODBC database connectivity.
* IP6 and fixed issue to Code Access Security for ASP.NET.

1. Version- 2.0 (2005):

* New CLR 2.0
* Enhancement of **ASP.NET & ADO.NET,**
* Covariance and Contravariance
* 64-bit support we added.

1. Version- 3.0 (2006):

* Windows Presentation Foundation (WPF),
* Windows Communication Foundation (WCF),
* Windows Workflow Foundation (WWF).

1. Version- 3.5 (2007):

* Build-in-Support **for AJAX,**
* **Language Integrated Query (LINQ),**
* **ASP.NET MVC**

1. Version- 4.0 (2010):

* New CLR 4.0,
* Task Parallel Library (TPL),
* Dynamic Language Runtime (DLR) were a major addition of this release.

1. Version- 4.5 (2012):

* Async support,
* Support for Windows Store app,
* Enhancements of WPF, WCF, WF, MEF, and ASP.NET, and base framework classes, such as support for arrays larger than 2GB on 64-bit platform.

1. Version- 4.5.1 (2013):

* Performance and debugging improvements,
* support for automatic binding redirection, and
* Expanded support for Windows Store application.

1. Version- 4.5.2 (2014) :

* ASP.NET APIs enhancements,
* System DPI support for Windows Forms controls,
* Debugging improvements in this release.

1. Version- 4.6 (2015):

* ASP.NET enhancements,
* ADO.NET always an encrypted feature for SQL Server 2016,
* new **64-bit JIT compiler,**
* Assembly Loader improvements,
* enhancements to Garbage Collector

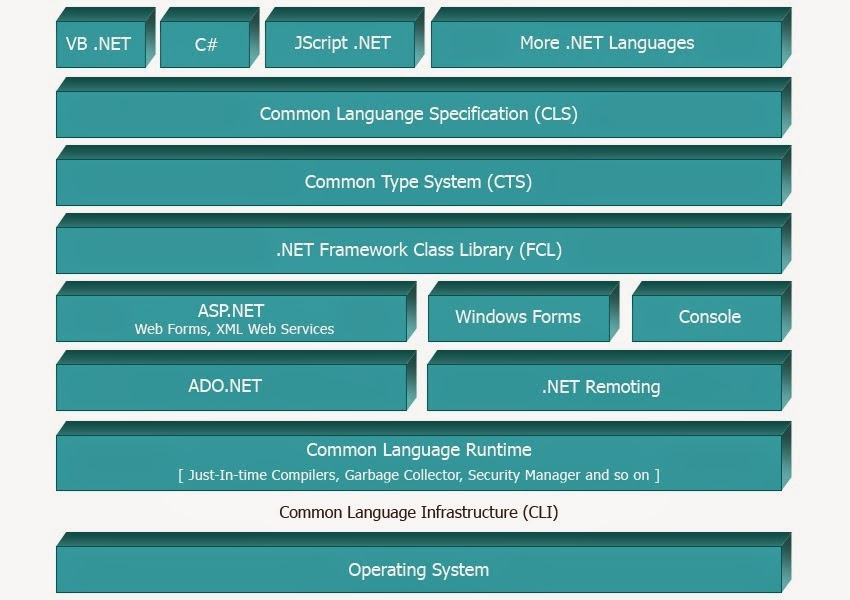
1. Version- 4.7 (2017):

* High DPI support for Windows Forms controls,
* Touch support for WPF in Windows 10,

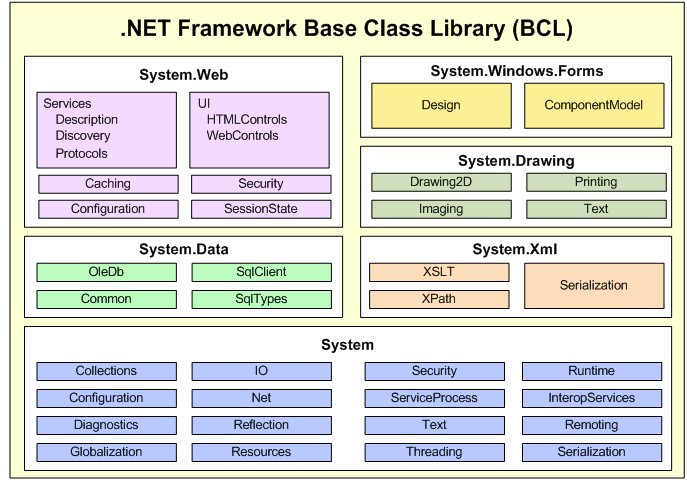
1. Version- 4.8 (2019):

* JIT improvements, Updated ZLib,
* FIPS improvements,
* Malware scanning for Assemblies and Accessibility Enhancements

## .NET framework architecture



## Base Class Library .net Framework

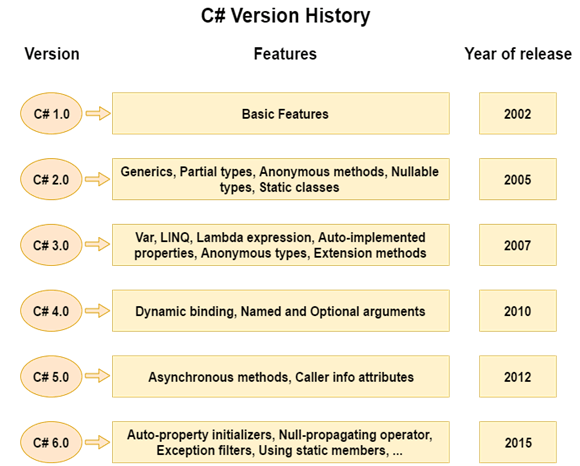


# C#

C#, pronounced as "C-Sharp". It is an object-oriented programming language which provided by Microsoft that runs on .Net Framework. By the help of C# programming language, we can develop different types of secured and robust applications:

* Window applications
* Web applications
* Distributed applications
* Web service applications
* Database applications etc.

**Version:**



#### Variable of C#:

|  |  |
| --- | --- |
| Decimal types | decimal |
| Boolean types | True or false value, as assigned |
| Integral types | int, char, byte, short, long |
| Floating point types | float and double |
| Nullable types | Nullable data types |

#### Data Types:

[](https://www.tutorialsteacher.com/Content/images/csharp/datatypes.png)

Var:

Dynamic:

#### Conversion:

1. **Implicit Conversion:** implicit type conversion, the C# compiler automatically converts one type to another. Smaller types (int) are automatically converted to larger types (double).

Code:

Int a = 5;

Double b = a;

1. **Explicit Conversion**: This type of conversion are done explicitly by users with the pre-defined functions.  Some data is lost during the type conversion. The explicit type conversion called type casting.

Code:

Double a = 5;

int b = (int) a ;

1. **Conversion using Parse ():** we can also use the Parse () method to perform type conversion. Generally, while performing type conversion between non-compatible types like int and string, we use Parse ().

Code:

string a = “100”;

int b = int.Parse(a) ;

float f = float.Parse(a) ;

1. **Conversion using Convert Class:** we can use the Convert class to perform type conversion. The Convert class provides various methods to convert one type to another.

Code:

int a = “100”;

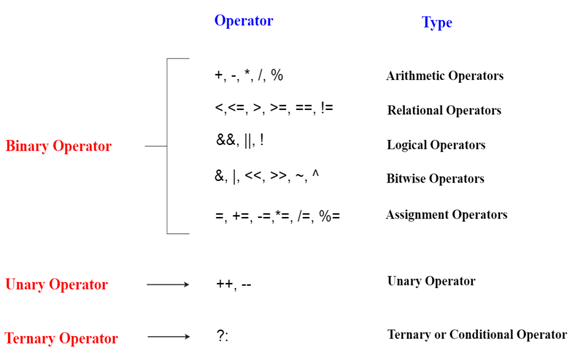
string b = Convert.ToString(a) ;

float f = Convert.ToDouble(a) ;

Method of convert class:

1. ToBoolean()
2. ToChar()
3. ToDouble()
4. ToInt16()
5. ToString()

#### Operator:



#### Namespace:

Namespace is a collection of classes, interface, structs, enum and delegates. The namespace also gives unique names to its classes thereby you can have the same class name in different namespaces.

namespace School

{

class Student

{

}

class Course

{

}

}

#### Output / input:

Output:

1. Concatenation Syntax:

String name = “shuvo”;

Console.WriteLine(“ hello world ”+name);

1. Placeholder Syntax:

String name= “shuvo”;

Int age = 25;

Console.WriteLine(“Your Name {0}, age {1}”, name, age);

Input: Console.ReadLine();

#### Control Statement:

* If-else:
* Switch:
* For loop
* While loop
* Do while loop
* Break
* Continue
* goto:

#### Function:

#### Array:

**Types of Array:**

1. Single Dimensional Array: There are 3 ways to initialize array at the time of declaration.
2. Array initialization.

int[] arr = new int[5]{ 10, 20, 30, 40, 50 };

1. We can omit the size of array.

int[] arr = new int[]{ 10, 20, 30, 40, 50 };

1. We can omit the new operator also.

int[] arr = { 10, 20, 30, 40, 50 };

1. Multidimensional Array: The multidimensional array is also known as rectangular arrays in C#.

Syntax:

int[,] arr=new int[3,3];

int[,,] arr=new int[3,3,3];

code:

int[,] arr = new int[3,3]= { { 1, 2, 3 }, { 4, 5, 6 }, { 7, 8, 9 } };

1. Jagged Array: In C#, jagged array is also known as "array of arrays" because its elements are arrays. The element size of jagged array can be different.

Code : int[][] arr = new int[2][];

**Call By Reference:**

C# provides a ref keyword to pass argument as reference-type. It changes in passed values are permanent and modify the original variable value.

public void Show(ref int val) {

             val \*= val; // Manipulating value

            Console.WriteLine("Value inside the show function "+val);

  }

 static void Main(string[] args) {

            int val = 50;

            Program program = new Program(); // Creating Object

            Console.WriteLine("Value before calling the function "+val);

            program.Show(ref val);

            Console.WriteLine("Value after calling the function " + val);

  }

**Call By Value:**

In C#, value-type parameters are that pass a copy of original value to the function rather than reference. It does not modify the original value.

 public void Show(int val) {

             val \*= val; // Manipulating value

            Console.WriteLine("Value inside the show function "+val);

  }

  static void Main(string[] args) {

            int val = 50;

            Program program = new Program(); // Creating Object

            Console.WriteLine("Value before calling the function "+val);

            program.Show(val);

            Console.WriteLine("Value after calling the function " + val);

    }

**Out Parameter:**

C# provides out keyword to pass arguments as out-type. It is like reference-type, except that it does not require variable to initialize before passing.

1. Static method cannot make object. We called static method by class.

#### Class:

1. Constructor:
2. Destructor:

public class Employee

    {

        public Employee()

        {

            Console.WriteLine("Constructor Invoked");

        }

        ~Employee()

        {

            Console.WriteLine("Destructor Invoked");

        }

    }

#### Object:

#### Properties:

#### Sealed keyword:

1. Sealed class: sealed class is a class that cannot inheritance by other class. But sealed class can be extending other class. Sealed class declared by sealed keyword.

Code:

sealed class Person{

string name;

}

Class User : Person {

// cannot extends Person class

}

1. Sealed method: Sealed keyword is preventing the further override the child override method.

We cannot override the sealed method. Sealed keyword always use in child class method.

Code:

Class A{

Int x = 10;

public void show(){

Console.WriteLine(x);

}

}

Class B : A{

Public sealed override void show(){

Console.WriteLine(x);

}

}

Class C : B{

public override void show(){

Console.WriteLine(x);

}

}

#### Abstraction

#### Namespace

#### File I/O

#### Collection:

Collection represents group of objects. There are 3 ways to work with collections.

1. System.Collections.Generic:

* List
* Stack
* Queue
* LinkedList
* HashSet
* SortedSet
* Dictionary
* SortedDictionary
* SortedList

1. System.Collections classes (Now deprecated)
   * ArrayList
   * Stack
   * Queue
   * Hashtable
2. System.Collections.Concurrent classes

* BlockingCollection
* ConcurrentBag
* ConcurrentStack
* ConcurrentQueue
* ConcurrentDictionary
* Partitioner
* Partitioner
* OrderablePartitioner

1. **Generic:**

#### Enum:

Enum is set of constants. An is a special class that represents a group of constants that we can readonly.

#### Delegates:

The delegate is a reference type data type that defines the method signature. You can define variables of delegate, just like other data type, that can refer to any method with the same signature as the delegate.

Delegate Syntax:

[access\_modifier] delegate [return type] [delegate name] ([parameters])

Code:

public delegate void MyDelegate(string msg);

Action:

Func:

#### Reflection:

#### Multithread:

#### Synchronization:

#### Web service

# XML

* Xml stands for Extensible Markup Language.
* XML is a makeup language much like HTML.
* XML was designed to store and transport data.
* XML was designed to be self-descriptive.
* XML was designed to be both human and machine readable.
* XML is used to structure data which are created by user.
* Xml case sensitive.

Code:

<user>

<name> shuvo </name>

<age> 25 </age>

</user>

**XML Structure:**

<root>  
  <child>  
    <subchild>.....</subchild>  
  </child>  
</root>

### Feature of XML:

1. XML Element:

<user>

<name> shuvo </name>

<age> 25 </age>

</user>

1. **XML attribute:**

* XML elements can have attributes, just like HTML. Attribute values must always be quoted.
* attributes cannot contain multiple values (elements can)
* attributes cannot contain tree structures (elements can)
* attributes are not easily expandable (for future changes)

<person gender="female">

<name> shuvo </name>

</person>

1. **XML Namespace:** XML Namespaces provide a method to avoid element name conflicts.
2. Name conflicts in XML can easily be avoided using a name prefix.

<h:table>  
  <h:tr>  
    <h:td>Apples</h:td>  
  </h:tr>  
</h:table>  
  
<f:table>  
  <f:name>African Coffee Table</f:name>  
</f:table>

1. **xmlns Attribute:** When using prefixes in XML, a namespace for the prefix must be defined. The namespace can be defined by an xmlns attribute in the start tag of an element.

<h:table xmlns:h="http://www.w3.org/TR/html4/">  
  <h:tr>  
    <h:td>Apples</h:td>  
    <h:td>Bananas</h:td>  
  </h:tr>  
</h:table>

# .NET WINFORM

Windows Forms is a Graphical User Interface (GUI) class library, which is bundled in .Net Framework. Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs. It is also termed as the WinForms. The applications which are developed by using Windows Forms or WinForms are known as the Windows Forms Applications that runs on the desktop computer. WinForms can be used only to develop the Windows Forms Applications not web applications.

**Tools of winform:**

1. Textbox:
2. Name:
3. Text:
4. Clear
5. Focus
6. Label:
7. Listbox:
8. Name:
9. Items:
10. count
11. Add:
12. Remove
13. Sorted
14. SelectedItems
15. selectedIndex
16. Clear
17. Button:
18. Combobox:
19. Add
20. Remove()
21. removeAt()
22. Clear();
23. Radio Button:
24. Checked;
25. Text
26. Checkbox:
27. Text;
28. Name
29. CheckListbox:
30. Text;
31. Name
32. checkedIndices
33. count
34. Validation:
35. ErrorValid
36. Regex
37. isMatch
38. Data type conversions:

Parse ():

Convert.int32 ();

**Property of Tools:**

* Apperance:
  1. backColor
  2. BackgroundImage
  3. BackgroundLayout
* Window Style

1. maximizeBox
2. minimizeBox

* Size

1. Startsize

* Font

**Event and Event Handler:**

1. Click
2. Leave
3. Enter
4. KeyPress:
5. KeyChar

**Method:**

1. IsNullOrEmpty
2. IsDigit()
3. Handled

# ADO.NET

ADO.NET stands for **ActiveX Data Objects**, which is Microsoft technology. It is a module of .Net Framework that is used to establish connection between application and data sources. Data sources can be such as SQL Server and XML. ADO.NET consists of classes that can be used to connect, retrieve, insert and delete data.

All the ADO.NET classes are located into **System.Data.dll** and integrated with XML classes located into **System.Xml.dll.**

ADO.NET has two main components that are used for accessing and manipulating data are the .NET Framework

1. Data provider
2. DataSet
3. **Data Providers:**

The Database cannot directly execute our C# code; it only understands SQL. So, if a .NET application needs to retrieve data or to do some insert, update, and delete operations from or to a database, then the .NET application needs to **Connect to the Database**, **prepare an SQL Command, Execute the Command, Retrieve the results and display them in the application**. In addition, this is possible with the help of .NET Data Providers.

Therefore, we can say Data provider is used to connect to the database execute commands and retrieve the record. Each provider exists in a namespace within the **System.Data namespace**, and consists of a number of classes. Every database its own data provider.

1. SQL Server - System.Data.SqlClient
2. OLE DB - System.Data.OleDb
3. Oracle - System.Data.OracleClient
4. EntityClient Provider - System.Data.EntityClient

**Each .NET data provider consists by four-core object:**

1. **Connection** – used to connect to the data source. Example- SQLConnection, OracleConnection, OleDbConnection, OdbcConnection, etc.
2. **Command** – used to execute a command against the data source and retrieve a DataReader or DataSet, or to execute an INSERT, UPDATE, or DELETE command against the data source. Example - SQLCommand, OracleCommand, OleDbCommand, OdbcCommand, etc.
3. **DataReader** – a forward-only, read-only connected resultset . Example - SQLDataReader, OracleDataReader, OleDbDataReader, OdbcDataReader, etc.
4. **DataAdapter** – used to populate a DataSet with data from the data source, and to update the data source. Example - SQLDataAdapter, OracleDataAdapter, OleDbDataAdapter, OdbcDataAdapter, etc.
5. **DataSet:**

The DataSet object in ADO.NET is not Provider-Specific. Once you connect to a database, execute the command, and retrieve the data into the .NET application. The data can then be stored in a DataSet and work independently of the database. So, it is used to access data independently from any data source. The DataSet contains a collection of one or more DataTable objects.

## Data Provider

#### Connection class:

1. The ADO.NET SqlConnection class belongs to System.Data.SqlClient namespace, and is used to establish an open connection to the SQL Server database.
2. The most important point that you need to remember is the connection does not close implicitly even if it goes out of scope. Therefore, it is always recommended and always a good programming practice to close the connection object explicitly by calling the Close() method of the connection object.

**Code:**

Namespace System {

Public sealed classs SqlConnection : DbConnection, ICloneable {

Public SqlConnection ();

Public sqlConnection (string cs);

Public sqlConnection (string cs, SqlCredential C);

BeginTransaction ();

ChangeDatabase (string database);

ChangePassword (string connectionString, string newPassword);

Open ();

Close ();

CreateCommand ();

}

}

**Method and Constructor of SqlConnection Class**:

1. **SqlConnection ():** It is used to initializes a new instance of the SqlConnection class.
2. **SqlConnection(String):** It is used to initialize a new instance of the SqlConnection class and takes connection string as an argument.
3. **SqlConnection (String, SqlCredential):** It is used to initialize a new instance of the SqlConnection class that takes two parameters. First is connection string and second is sql credentials.
4. **BeginTransaction ():** It is used to start a database transaction.
5. **ChangeDatabase(String):** It is used to change the current database for an open SqlConnection.
6. **ChangePassword (String, String):** It changes the SQL Server password for the user indicated in the connection string.
7. **Close ():** It is used to close the connection to the database.
8. **CreateCommand ():** It enlists in the specified transaction as a distributed transaction.
9. **GetSchema ():** It returns schema information for the data source of this SqlConnection.
10. **Open ():** It is used to open a database connection.
11. **ResetStatistics ():** It resets all values if statistics gathering is enabled.

#### Command Class:

This class is used to store and execute SQL statement for SQL Server database. It is a sealed class so that cannot be inherited. The Command object can execute SELECT statements, INSERT, UPDATE, or DELETE statements, stored procedures, or any other statement understood by the database.

Code:

public sealed class SqlCommand :  DbCommand, ICloneable{

SqlCommand();

SqlCommand(String);

SqlCommand(String, SqlConnection);

SqlCommand(String, SqlConnection, SqlTransaction);

Cancel();

Clone();

ExecuteReader();

ExecuteNonQuery();

ExecuteScalar():

}

**Method and Constructor of SqlConnection Class**:

1. **SqlCommand(string cmdText):** It is used to initialize a new instance of the System.Data.SqlClient.SqlCommand class with the text of the query. Here, the cmdText is the text of the query that we want to execute.
2. **SqlCommand(string cmdText, SqlConnection connection):** It is used to initialize a new instance of the System.Data.SqlClient.SqlCommand class with the text of the query and a System.Data.SqlClient.SqlConnection. Here, the cmdText is the text of the query that we want to execute and the parameter connection is the connection to an instance of SQL Server.
3. **SqlCommand(string cmdText, SqlConnection connection, SqlTransaction transaction):** It is used to initialize a new instance of the System.Data.SqlClient.SqlCommand class with the text of the query, a SqlConnection instance, and the SqlTransaction instance. Here, the parameter cmdText is the text of the query. The parameter connection is a SqlConnection that represents the connection to an instance of SQL Server and the parameter transaction is the SqlTransaction in which the SqlCommand executes.
4. **SqlCommand(string cmdText, SqlConnection connection, SqlTransaction transaction, SqlCommandColumnEncryptionSetting columnEncryptionSetting):** It is used to initialize a new instance of the System.Data.SqlClient.SqlCommand class with specified command text, connection, transaction, and encryption settings. We already discussed the first three parameters which are the same as the previous. Here, the fourth parameter i.e. columnEncryptionSetting is the encryption setting.
5. **BeginExecuteNonQuery():** This method initiates the asynchronous execution of the Transact-SQL statement or stored procedure that is described by this System.Data.SqlClient.SqlCommand.
6. **Cancel():** This method tries to cancel the execution of a System.Data.SqlClient.SqlCommand.
7. **Clone():** This method creates a new System.Data.SqlClient.SqlCommand object is a copy of the current instance.
8. **CreateParameter():** This method creates a new instance of system.Data.SqlClient.SqlParameter object.
9. **ExecuteReader():** This method Sends the System.Data.SqlClient.SqlCommand.CommandText to the System.Data.SqlClient.SqlCommand.Connection and builds a System.Data.SqlClient.SqlDataReader.
10. **ExecuteScalar():** This method Executes the query, and returns the first column of the first row in the result set returned by the query. Additional columns or rows are ignored.
11. **ExecuteNonQuery**(): This method executes a Transact-SQL statement against the connection and returns the number of rows affected.
12. **Prepare():** This method creates a prepared version of the command on an instance of SQL Server.
13. **ResetCommandTimeout():** This method resets the CommandTimeout property to its default value.

**Sql Excecute with class:**

#### DataReader Class:

This class is used to read data from SQL Server database. It reads data in forward-only stream of rows from a SQL Server database. it is sealed class so that cannot be inherited. It inherits DbDataReader class and implements IDisposable interface.

**Code:**

public class SqlDataReader :  DbDataReader, IDisposable  {

Connection cn;

Depth;

FieldCount;

HasRows;

IsClosed;

Item[String];

RecordsAffected;

Close();

GetValue(Int32);

GetValues(Object[]);

NextResult();

Read();

}

**SqlDataReader Class Properties:**

1. **Connection:** It gets the System.Data.SqlClient.SqlConnection associated with the System.Data.SqlClient.SqlDataReader.
2. **Depth:** It gets a value that indicates the depth of nesting for the current row.
3. **FieldCount:** It gets the number of columns in the current row.
4. **HasRows:** It gets a value that indicates whether the System.Data.SqlClient.SqlDataReader contains one or more rows.
5. **IsClosed: It** retrieves a Boolean value that indicates whether the specified System.Data.SqlClient.SqlDataReader instance has been closed.
6. **RecordsAffected:** It gets the number of rows changed, inserted, or deleted by the execution of the Transact-SQL statement.
7. **VisibleFieldCount:** It gets the number of fields in the System.Data.SqlClient.SqlDataReader that is not hidden.
8. **Item[String]:** It gets the value of the specified column in its native format given the column name.
9. **Item[Int32]:** It gets the value of the specified column in its native format given the column ordinal.

**ADO.NET SqlDataReader Class Methods in C#:**

1. **Close():** It closes the SqlDataReader object.
2. **GetBoolean(int i):** It gets the value of the specified column as a Boolean. Here, parameter i is the zero-based column ordinal.
3. **GetByte(int i):** It gets the value of the specified column as a byte. Here, parameter i is the zero-based column ordinal.
4. **GetChar(int i):** It gets the value of the specified column as a single character. Here, parameter i is the zero-based column ordinal.
5. **GetDateTime(int i):** It gets the value of the specified column as a System.DateTime object. Here, parameter i is the zero-based column ordinal.
6. **GetDecimal(int i):** It gets the value of the specified column as a System.Decimal object. Here, parameter i is the zero-based column ordinal.
7. **GetDouble(int i):** It gets the value of the specified column as a double-precision floating-point number. Here, parameter i is the zero-based column ordinal.
8. **GetFloat(int i):** It gets the value of the specified column as a single-precision floating-point number. Here, parameter i is the zero-based column ordinal.
9. **GetName(int i):** It gets the name of the specified column. Here, parameter i is the zero-based column ordinal.
10. **GetSchemaTable():** It returns a System.Data.DataTable that describes the column metadata of the System.Data.SqlClient.SqlDataReader
11. **GetValue(int i):** It gets the value of the specified column in its native format. Here, parameter i is the zero-based column ordinal.
12. **GetValues(object[] values):** It Populates an array of objects with the column values of the current row. Here, the parameter values is an array of System.Object into which to copy the attribute columns.
13. **NextResult():** It advances the data reader to the next result when reading the results of batch Transact-SQL statements.
14. **Read():** It Advances the System.Data.SqlClient.SqlDataReader to the next record and returns true if there are more rows; otherwise false.

#### DataAdapter Class:

The DataAdapter works as a bridge between a DataSet and a data source to retrieve data. DataAdapter is a class that represents a set of SQL commands and a database connection. It can be used to fill the DataSet and update the data source.

Code:

public class DataAdapter :  Component,  IDataAdapter  {

DataAdapter();

SqlDataAdapter(SqlCommand selectCommand):

SqlDataAdapter(string sct, string selectConnectionString):

CloneInternals();

Dispose(Boolean);

GetFillParameters();

Update(DataSet);

}

**Constructors SqlDataAdapter class:**

1. **SqlDataAdapter():** Initializes a new instance of the SqlDataAdapter class.
2. **SqlDataAdapter(SqlCommand selectCommand**): Initializes a new instance of the SqlDataAdapter class with the specified SqlCommand. Here, the selectCommand can be a Transact-SQL SELECT statement or a stored procedure.
3. **SqlDataAdapter(string selectCommandText, string selectConnectionString):** Initializes a new instance of the SqlDataAdapter class with the command and a connection string. Here, the selectCommandText can be a Transact-SQL SELECT statement or a stored procedure.
4. **SqlDataAdapter(string selectCommandText, SqlConnection selectConnection):**Initializes a new instance of the SqlDataAdapter class with the command and a connection string.

**Methods SqlDataAdapter class:**

1. **CloneInternals():** It is used to create a copy of this instance of DataAdapter.
2. **Dispose(Boolean):** It is used to release the unmanaged resources used by the DataAdapter.
3. **Fill(DataSet):** It is used to add rows in the DataSet to match those in the data source.
4. **FillSchema(DataSet, SchemaType, String, IDataReader**): It is used to add a DataTable to the specified DataSet.
5. **GetFillParameters():** It is used to get the parameters set by the user when executing an SQL SELECT statement.
6. **ResetFillLoadOption():** It is used to reset FillLoadOption to its default state.
7. **ShouldSerializeAcceptChangesDuringFill():** It determines whether the
8. **ShouldSerializeFillLoadOption():** It determines whether the FillLoadOption property should be persisted or not.
9. ShouldSerializeTableMappings(): It determines whether one or more DataTableMapping objects exist or not.
10. Update(DataSet): It is used to call the respective INSERT, UPDATE, or DELETE statements.

## Data Set

#### DataTable class:

DataTable represents relational data into tabular form. ADO.NET provides a DataTable class to create and use data table independently. It can also be used with DataSet also. Initially, when we create DataTable, it does not have table schema. We can create table schema by adding columns and constraints to the table. After defining table schema, we can add rows to the table. Datatable class store in System.Data namespace.  DataTable is a central object which can be used independently or can be used by other objects such as DataSet and the DataView.

Code:

**public class DataTable :**  MarshalByValueComponent, IListSource, ISupportInitializeNotification, ISerializable, IXmlSerializable {

Columns

Constraints

DataSet

DefaultView

Rows

TableName

DataTable() { };

DataTable(String) { };

DataTable(SerializationInfo, StreamingContext){ };

DataTable(String, String)

AcceptChanges()

Clear()

Clone()

Copy()

Select()

}

**Constructors:**

1. DataTable():
2. DataTable(string tableName):
3. DataTable(SerializationInfo info, StreamingContext context):
4. DataTable(string tableName, string tableNamespace):

**Properties of DataTable:**

1. Columns: It is used to get the collection of columns that belong to this table.
2. Constraints: t is used to get the collection of constraints maintained by this table.
3. DataSet: It is used to get the DataSet to which this table belongs.
4. DefaultView: It is used to get a customized view of the table that may include a filtered view.
5. HasErrors: It is used to get a value indicating whether there are errors in any of the rows in the table of the DataSet.
6. MinimumCapacity: It is used to get or set the initial starting size for this table.
7. PrimaryKey: It is used to get or set an array of columns that function as primary keys for the data table.
8. Rows: It is used to get the collection of rows that belong to this table.
9. TableName: It is used to get or set the name of the DataTable.

**Methods of DataTable:**

1. AcceptChanges(): It is used to commit all the changes made to this table.
2. Clear(): It is used to clear the DataTable of all data.
3. Clone(): It is used to clone the structure of the DataTable.
4. Copy(): It is used to copy both the structure and data of the DataTable.
5. CreateDataReader(): It is used to return a DataTableReader corresponding to the data within this DataTable.
6. CreateInstance(): It is used to create a new instance of DataTable.
7. GetRowType(): It is used to get the row type.
8. GetSchema(): It is used to get the schema of the table.
9. ImportRow(DataRow): It is used to copy a DataRow into a DataTable.
10. Load(IDataReader): It is used to fill a DataTable with values from a data source using the supplied IDataReader.
11. Merge(DataTable, Boolean): It is used to merge the specified DataTable with the current DataTable.
12. NewRow(): It is used to create a new DataRow with the same schema as the table.
13. Select(): It is used to get an array of all DataRow objects.
14. WriteXml(String): It is used to write the current contents of the DataTable as XML using the specified file.

#### DataSet

It is a collection of data tables that contain the data. It is used to fetch data without interacting with a Data Source that's why, it also known as disconnected data access method. It is an in-memory data store that can hold more than one table at the same time. We can use DataRelation object to relate these tables. The DataSet can also be used to read and write data as XML document.

ADO.NET provides a DataSet class that can be used to create DataSet object. It contains constructors and methods to perform data related operations.

**Constructors of DataSet:**

1. **DataSet():** It initializes a new instance of the System.Data.DataSet class..
2. **DataSet(string dataSetName):**
3. **DataSet(SerializationInfo info, StreamingContext context**
4. **DataSet(SerializationInfo info, StreamingContext context, bool ConstructSchema):**

**Properties of DataSet:**

1. **CaseSensitive:**
2. **DefaultViewManager**:
3. **DataSetName**
4. **EnforceConstraints**:
5. **HasErrors**:
6. **IsInitialized**:
7. **Prefix**:
8. **Locale**:
9. **Namespace**:
10. **Site**:
11. **Relations**.
12. **Tables**:

**Methods of DataSet Class:**

1. BeginInit():
2. Clear():
3. Clone():
4. Copy():
5. CreateDataReader():
6. CreateDataReader(params DataTable[] dataTables):
7. EndInit():
8. GetXml():
9. GetXmlSchema():

# ADO.net Connectivity with SQL Server

1. Connect the data base from “Data Sources” Menu.
2. Configure the “App.config” xml file.

<connectionStrings>

<add name =”dbcs” connectionString=”…” providerName =”System.Data.SqlClient”>

</connectionStrings>

1. Now create an object of configurationManager class.

string cs = ConfigurationManager.ConnectionStrings[“dbcs”].connectionString;

SqlConnection con = new SqlConnection(cs);

**Select :**

1. Sql query for select:

String sql = “select \* from customer where cid=@cid”;

1. Create command object :

SqlCommand cmd = new SqlCommand(query, con);

1. Value add in insert query:

cmd.Parameters.AddWithValue(“@cid”,1364);

1. Connection open.

Con.Open();

1. Execute query.

SqlDataReader rd = Cmd.ExecuteReader();

1. Close the connection

Con.close;

**Inset/update/delete:**

1. Sql query:

String query = “insert into customer values(@cid, @cname);”

1. Create Command object of the query.

SqlCommand cmd = new SqlCommand(query, con);

1. Value add in insert query:

Cmd.Parameters.AddWithValue(“@cid”,1364);

Cmd.Parameters.AddWIthValue(“@cname”,”shuvo”);

1. Open the connection

Con.Open();

1. Execute the query.

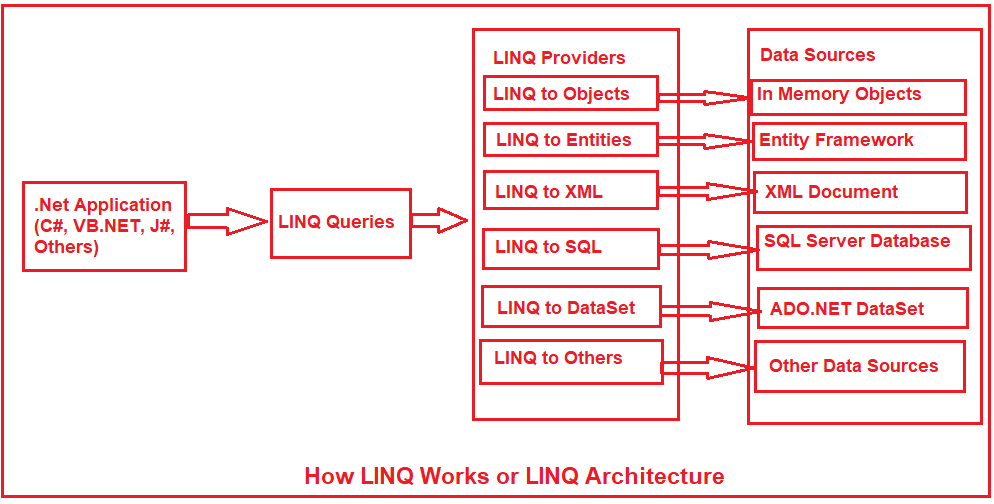
Int a = Cmd.ExecuteNonQuery();

1. Close the connection

Con.close();

# LINQ

1. The LINQ (Language Integrated Query) is a part of a language but not a complete language.
2. It was introduced by Microsoft with .NET Framework 3.5 and C# 3.0 and is available in System.Linq namespace.



## Query Operation

In order to write a LINQ query, we need the following three things

1. Data Source (In-Memory Objects, SQL Server, XML Document, etc)
2. Query
3. Execution of the Query

### Data Source:

We can write LINQ queries for the classes that implement IEnumerable<T> or IQueryable<T> interface.

1. Enumerable: The Enumerable class includes extension methods for the classes that implement IEnumerable<T> interface, for example, all the built-in collection classes implement IEnumerable<T> interface and so we can write LINQ queries to retrieve data from the built-in collections.
2. Queryable: The Queryable class includes extension methods for classes that implement IQueryable<t> interface. The IQueryable<T> interface is used to provide querying capabilities against a specific data source where the type of the data is known. For example, Entity Framework api implements IQueryable<T> interface to support LINQ queries with underlaying databases such as MS SQL Server.

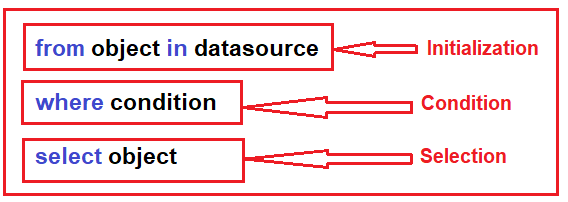
### Query:

A query is nothing but a set of instructions that are applied to a data source (i.e. In-Memory Objects, SQL Server, XML Document) to perform certain operations (i.e. CRUD operations) and then tells the shape of the output from that query. Each query is a combination of three things.

1. Initialization (to work with a particular data source)
2. Condition ( where, filter, sorting condition)
3. Selection (single selection, group selection, or joining)

WRITE A LINQ QUERY

1. **Query Syntax:**



Example:

List<int> integerList = new List<int> () {

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

};

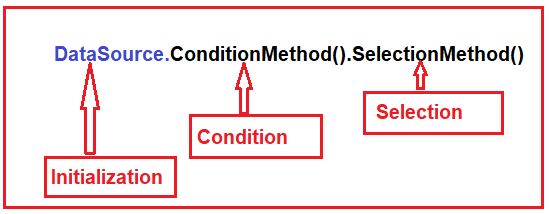
var QuerySyntax = from obj in integerList where obj > 5 select obj;

foreach (var item in QuerySyntax) {

Console.Write(item + " “);

}

1. **Method Syntax:**



Example:

List<int> integerList = new List<int> () {

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

};

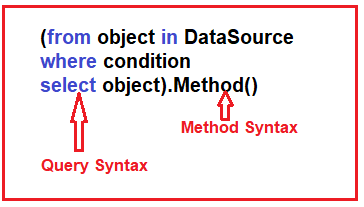
var MethodSyntax = integerList.Where(obj => obj > 5).ToList();

foreach (var item in QuerySyntax) {

Console.Write(item + " ");

}

1. **Mixed Syntax:**



Example:

List<int> integerList = new List<int> () {

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

};

var MethodSyntax = (from obj in integerList where obj > 5 select obj).Sum();

Console.Write("Sum Is: " + MethodSyntax);

# Entity Framework

Entity Framework is an ORM and ORMs are aimed to increase the developer’s productivity by reducing the redundant task of persisting the data used in the applications.

1. Entity Framework can generate the necessary database commands for reading or writing data in the database and execute them for you.
2. If you are querying, you can express your queries against your domain objects using LINQ to entities.
3. Entity Framework will execute the relevant query in the database and then materialize results into instances of your domain objects for you to work within your app.

EF 6 Version History:

| EF Version | Release Year | .NET Framework |
| --- | --- | --- |
| EF 6 | 2013 | .NET 4.0 & .NET 4.5, VS 2012 |
| EF 5 | 2012 | .NET 4.0, VS 2012 |
| EF 4.3 | 2011 | .NET 4.0, VS 2012 |
| EF 4.0 | 2010 | .NET 4.0, VS 2010 |
| EF 1.0 (or 3.5) | 2008 | .NET 3.5 SP1, VS 2008 |

## Context Class

1. Context class is derives from System.Data.Entity.DbContextDbContext.
2. An instance of the context class represents Unit Of Work and Repository patterns wherein it can combine multiple changes under a single database transaction.
3. The context class is used to query or save data to the database.
4. It is also used to configure domain classes, database related mappings, change tracking settings, caching, transaction.

using System.Data.Entity;

public class BankAccContext : DbContext{

public BankAccContext()   {  }

public DbSet<AccHolder> Accountholder { get; set; }

public DbSet<AccNo> Accountnumber { get; set; }

public DbSet<AccType> Accounttype { get; set; }

}

## Entity

An entity is a class that maps to a database table. This class must be included as a DbSet type property in the DbContext class. Entity Framework API maps each entity to a table and each property of an entity to a column in the database.

1. **POCO Entities (Plain Old CLR Object**)
2. A POCO entity is a class that doesn't depend on any framework-specific base class.
3. It is like any other normal .NET CLR class, which is why it is called "Plain Old CLR Objects".
4. These POCO entities (also known as persistence-ignorant objects) support most of the same query, insert, update, and delete behaviors as entity types that are generated by the Entity Data Model.
5. **Dynamic Proxy Entities (POCO Proxy)**
6. A POCO class must be declared with public access.
7. A POCO class must not be sealed (NotInheritable in Visual Basic).
8. A POCO class must not be abstract (MustInherit in Visual Basic).
9. Each navigation property must be declared as public, virtual.
10. Each collection property must be ICollection<T>.
11. The ProxyCreationEnabled option must NOT be false (default is true) in context class.

**An Entity can include two types of properties:**

1. **Scalar Properties:** The type of primitive property is called scalar properties. Each scalar property maps to a column in the database table, which stores the real data.

public class Student{

        public int StudentID { get; set; }

        public string StudentName { get; set; }

        public decimal height { get; set; }

        public float weight { get; set; }

        public Grade Grade { get; set; }

}

1. **Navigation Properties:** Navigation Property represents the relationship with another entity.

We have two types of Navigation property:

**Reference Navigation:** If an entity includes the property of another type of the entity, then it is called as Reference Navigation Property.

**Collection Navigation:** If an entity includes a property of the generic collection of an entity type, then it is called a collection navigation property.

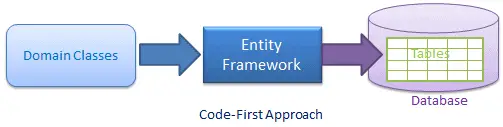
## Entity Framework Approach

The Entity Framework provides three approaches to create an entity model and each one has their own pros and cons.

1. Code First
2. Database First
3. Model First
4. **Code First Approach:**

The Code First approach used when you do not have an existing database for your application. In the code-first approach, you start writing your entities (domain classes) and context class first and then create the database from these classes using migration commands.

Developers who follow the Domain-Driven Design (DDD) principles, prefer to begin with coding their domain classes first and then generate the database required to persist their data.



public class Product{

public int ProductId { get; set; }

public string ProductName { get; set; }

public float Price { get; set; }

}

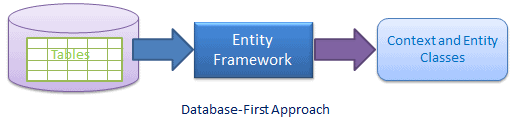
public class IDGContext : DbContext{

public DbSet<Product> Products { get; set; }

}

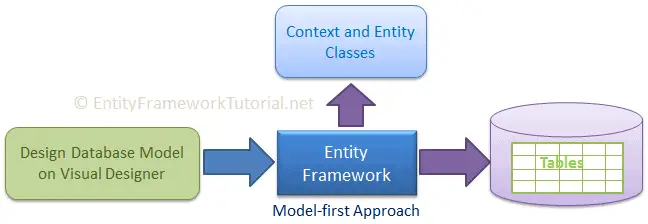
1. **Database First Approach:**

In the database-first development approach, you generate the context and entities for the existing database using EDM wizard integrated in Visual Studio or executing EF commands.

[](https://www.entityframeworktutorial.net/images/EF5/databasefirst.png)

1. **Model First Approach:**

In the model-first approach, you create entities, relationships, and inheritance hierarchies directly on the visual designer integrated in Visual Studio and then generate entities, the context class, and the database script from your visual model.



# ASP

Asp are creating for server slide application. With duration it is develop in two different platforms.

1. Classic ASP
2. ASP.NET

### Classic ASP:

1. ASP stands for Active Server Pages, which is known as classic ASP.
2. Classic ASP is old but make a server-side scripting environment, which create dynamic web applications.
3. With ASP, you can combine HTML pages, script commands, and COM components to create interactive web pages that are easy to develop and modify.
4. Classic ASP is the predecessor to ASP.NET, but it is still in wide use today.
5. An ASP file has the file extension ".asp"
6. An ASP file is just the same as an HTML file
7. An ASP file can contain server scripts in addition to HTML
8. Server scripts in an ASP file are executed on the server
9. Edit, change, add content, or customize any web page
10. Respond to user queries or data submitted from HTML forms
11. Provide web security since ASP code cannot be viewed in a browser

**Language:** classis asp scripting language is VBScript.

**Output**: The Response.Write() method is used by ASP to write output to HTML.

<% Response.Write("Hello World!") %>

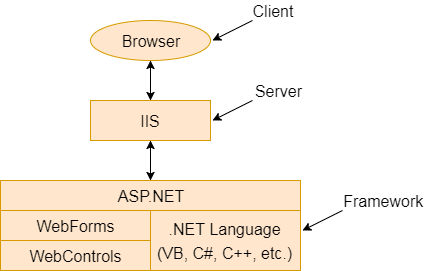
**Variables**: <% Dim x(2,2) %>

### **ASP.NET**

1. Asp.net is a web development platform, which provides a programming model, a comprehensive software infrastructure for web application. It is a server-side Scripting language. It browser independent.
2. Asp.net was released in 2002 as successor to classic ASP.
3. Programming models for creating ASP.NET web sites and web applications.
4. ASP.net web form
5. ASP.NET MVC
6. ASP.NET Web Pages

# ASP.net Web Forms:

1. Web Forms is the oldest ASP.NET programming model, with event driven web pages written as a combination of HTML, server controls, and server code.
2. Web Forms are compiled and executed on the server, which generates the HTML that displays the web pages.
3. In Web form we can use VB or C# as programming language.
4. ASP.NET web forms contain various web pages and GUI applications such as text box, data grid, label, checkbox, hyperlink, etc.
5. It provides flexibility to web pages at run time as well as design time. It also provides a feature to write code in a separate file from the controls
6. Web Forms are made up of two components: the visual portion (the ASPX file), and the code behind the form, which resides in a separate class file.



**Features of Web Forms:**

1. **Server Controls:**

It provides a vast set of server controls. These controls are like objects, and they run when they are requested and rendered to the browser. Some web pages are similar to HTML elements like text-box, button, checkbox, and hyperlink.

Label: < asp:LabelID="Label\_1" runat="server" Text="Label" > </asp:Label>

Text Box: < asp:TextBoxID="Text\_Box" runat="server" > </asp:TextBox>

1. **Default page:**

Default page is initial state of asp.net web form where server is start.

1. **Mater Pages:**

Mater Pages is responsible for the consistent layout of our web applications. It gives a proper appearance and standard to different pages.

1. **Working with data:**
2. **Ispostback:**
3. **QueryString:**
4. **Membership:**
5. **Client Script and Client Frameworks**
6. **State Management:**
7. ViewState:

ViewState[“user”] = UserTextBox.Text;

1. ApplicationState:
2. sessionState:
3. **Reapter Control:**
4. **Error Handling:**
5. **Routing:**

URL routing can be configured to a web application. A request URL is a URL that a user enters in a browser to browse in a specific place.

1. **Security:**

Security always plays a crucial role in software development. ASP.NET provides different configuration options and extensibility points to make our systems more secure.

**Code :**

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

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<style type="text/css">

.auto-style1 {

     width: 100%;

}

</style>

</head>

<body>

<form id="form1" runat="server">

<table class="auto-style1">

 <tr>

   <td> <asp:Label ID="Label1" runat="server" Text="Name"></asp:Label>  </td>

  <td> <asp:TextBox ID="name" runat="server" required="true"></asp:TextBox> </td>

 </tr>

 <tr>

   <td> <asp:Label ID="Label2" runat="server" Text="Password"></asp:Label></td>

   <td><asp:TextBox ID="T2" runat="server" TextMode="Password"></asp:TextBox></td>

 </tr>

 <tr>

   <td> <asp:Label ID="Label5" runat="server" Text="Select Course"></asp:Label>s</td>

    <td>

  <asp:CheckBox ID="CheckBox1" runat="server" Text="J2SEE" />

  <asp:CheckBox ID="CheckBox2" runat="server" Text="J2EE" />

   </td>

  </tr>

<tr>

 <td><asp:Button ID="Bnt" runat="server"  CssClass="btn" OnClick="Click"/> </td>

  </tr>

</table>

</form>

</body>

</html>

# ASP.NET MVC:

**The MVC (Model-View-Controller)** is an application development pattern or design pattern which separates an application into three main components:

1. Model
2. View
3. Controller
4. **Model:**

Model is a part of the application, which implements the logic for the data domain of the application. It is used to retrieve and store model state in a database such as SQL Server database. It also used for business logic separation from the data in the application.

1. **View:**

View is a component that forms the application's user interface. It is uses to create web pages for the application. An example would be an edit view of a Products table that displays text boxes, drop-down lists and check boxes based on the current state of a Product object.

1. **Controller:**

Controller is the component, which handles user interaction. It works with the model and selects the view to render the web page. In an MVC application, the view only displays information whereas the controller handles and responds to the user input and requests.

**Action method:**

* + 1. Action method is the public method in controller class
    2. Are responsible for processing the request.
    3. Action method should be public

# ASP.NET WEB PAGE:

ASP.NET Web Pages is a framework that you can use to create dynamic web pages. It provides fast and lightweight way to combine server code with HTML. It helps to add video, links to the social sites. It also provides other features like you can create beautiful sites that conform to the latest web standards.

## Razor Markup:

1. Razor is a simple markup syntax for embedding server code (C# or VB) into ASP.NET web pages.
2. C# code blocks are enclosed in @{ ... }
3. Inline expressions (variables or functions) start with @

<! DOCTYPE html>  
<html lang="en">  
<head>  
     <meta charset="utf-8" />  
     <title>Web Pages Demo</title>  
</head>  
<body>  
     <h1>Hello Web Pages</h1>  
     <p>The time is @DateTime.Now</p>  
</body>  
</html>

1. **Razor Engine**:
2. Razor engine convert the razor syntax in html format.
3. Complies a view of application when the view is requested for the first time.
4. Deliver the complied view for subsequent request until you make changes to the view.
5. **Razor Variable:**
   * 1. Int
     2. Float
     3. Decimal
     4. Bool
     5. String
6. **Loop:**
7. Forloop:

@for ( var i = 10; i < 21; i++ ){

<p> Line @i </p>

}

1. Foreach:

@foreach (var x in Request.ServerVariables){

<li> @x </li>

}

1. Whileloop:

@{  
 var i = 0;  
 while (i < 5){  
    i += 1;  
    <p>Line @i</p>  
    }  
}

1. **Condition :**

@if (price>=30){  
  <p>The price is high.</p>  
  }  
else if (price>20 && price<30){  
  <p>The price is OK.</p>  
  }  
else{  
   <p>The price is low.</p>  
  }

## Razor Syntax:

1. **ViewData:**
2. ViewData is similar to ViewBag, which transfers data from Controller to View.
3. ViewData is of Dictionary type, whereas ViewBag is of dynamic type.
4. It is valid only during the current request.
5. ViewData is a dictionary, so it contains key-value pairs where each key must be a string.
6. Typecast is needed in ViewData.

string[] name = { "shuvo", "rahiim", "orange" };

ViewData["name"] = name;

@{

foreach(string i in (string[]) ViewData["name"])

{

<li> @i </li>

}

}

1. **ViewBag:**
2. ViewBag is a dynamic property that allows you to pass data from a controller to a view.
3. you can access the data set in ViewBag using the @ symbol with the property name.

ViewBag.Message = " Welcome to our website! " ;

<h1>@ViewBag.Message</h1>

1. **TempData:**
2. TempData is a mechanism that allows you to store data between requests.
3. It is useful when you need to transfer data from one action method to another action method within the same controller or even across different controllers.
4. TempData Value must be typecast Before use.
5. TempData allows passing data from the current request to the subsequent request during request redirection.

TempData["Message"] = "This is a message from Action1.";

string message = TempData["Message"] as string;

1. **Session:**
2. Session is a feature that allows you to store and retrieve user-specific data across multiple requests.
3. It provides a way to maintain stateful information for a particular user during their entire session on the website.

Session["Message"] = "This is a message from Action1.";

string message = Session["Message"];

## Helper Method:

1. It is use only in view page.
2. It used to render html content in a view.
3. Allows generating HTML markup that you can reuse across the web application.

**Method:**

1. **Html.ActionLink():**

This method allows you to generate a hyperlink that points to an action method of a controller class.

**Syntax:**

@Html.ActionLink(linkText, actionName, controllerName, routeValues, htmlAttributes)

**Code :**

@using Microsoft.AspNetCore.Mvc.Rendering

@{

var linkText = "Click here!";

var action = "Index";

var controller = "Home";

var routeValues = new { id = 123 };

var htmlAttributes = new { @class = "btn btn-primary" };

}

@Html.ActionLink(linkText, action, controller, routeValues, htmlAttributes)

1. **Html.Label():**
2. **Html.BeginForm():**
3. **Html.TextBox():**
4. **Html.TextArea():**

## Layout:

In Web Page, we can break the code, in block of content and we can reuse this block separate file. Like headers and footers, in separate files.

1. **Folder:**
2. **Form:**
3. **Object:**

Method:

1. href
2. RenderBody()
3. RenderPage(page)
4. RenderSection(section)
5. Write(object)
6. WriteLiteral

Properties:

1. IsPost
2. Layout
3. Page
4. Request
5. Server
6. Database:
7. Helper:
8. WebGrid:
9. Charts
10. Email
11. Security
12. Publish

# .NET CORE

1. .NET Core is a new version of .NET Framework, which is a free, open-source, general-purpose development platform maintained by Microsoft. It is a cross-platform framework that runs on Windows, macOS, and Linux operating systems.
2. .NET Core Framework can be used to build different types of applications such as mobile, desktop, web, cloud, IoT, machine learning, micro services, game, etc.
3. .NET Core is written from scratch to make it modular, lightweight, fast, and cross-platform Framework. It includes the core features that are required to run a basic .NET Core app. Other features are provided as NuGet packages, which you can add it in your application as needed. In this way, the .NET Core application speed up the performance, reduce the memory footprint and becomes easy to maintain.

**.NET CORE feature:**

* 1. ASP.NET Core MVC
  2. Universal Windows app

**Version of .NET Core:**

1. **.NET 6:**

* 2021
* Visual Studio 2022
* Hot Reload improvements, RyuJIT compiler, and runtime performance boost, and early builds of MAUI, the multi-platform UI support based on Xamarin

1. **.NET 5**

* 2020
* Visual Studio 2019 V 16.8
* Uniform runtime behavior with a single.NET runtime that can be used everywhere.
* A development platform for everything from Linux and Windows to iOS and Android to TV and watches and Web Assembly.

1. **.NET Core 3.1**

* 2019
* Visual Studio 2019 V 16.4
* Issue fixes & improvements of .Net Core 3.0
* Long Term Support (LTS) for three years.

1. **.NET Core 3.0**

* 2019
* Visual Studio 2019 V 16.3
* Improved overall Performance.
* Improved API performance

1. **.NET Core 2.0**

* 2017
* Implements.NET Standard 2.0,
* Supports 6 new distros,
* RyuJIT x86 JIT in Core 2.0,
* Dotnet restore is an implicit command.

1. **.NET Core 1.0**

* 2016
* Visual Studio 2015
* First time release,
* open-source, Cross-platform, has flexible deployment,
* uses ASP.NET Core and UWP

# ASP.NET CORE MVC

## Middleware:

ASP.NET Core introduced a new concept called Middleware. A middleware is nothing but a component (class) which is executed on every request in ASP.NET Core application. In the classic ASP.NET, HttpHandlers and HttpModules were part of request pipeline. Middleware is similar to HttpHandlers and HttpModules where both needs to be configured and executed in each request.

1. **Authentication :**

app.UseAuthentication()

1. **CORS:**
2. **Routing:**

app.UseRouting();

app.MapControllerRoute();

app.Run();

app.Map();

app.MapGet();

app.MapPost();

app.MapDelete();

app.UseEndpoints(endpoints =>

{

endpoints.MapGet(“/Home”, async (context)=>

{

await Context.Response.WriteAsync(“this is Get page “);

});

endpoints.MapPost(“/Home”, async (context)=>

{

await Context.Response.WriteAsync(“this is Post page “);

});

});

1. **Session:**

app.UseSession();

1. **StaticFiles:**

app.UseStaticFiles();

1. **HTTPS:**

app.UseHttpsRedirection();

1. **Diagnostics:** Diagnosticsmiddleware is used for reporting and handling exceptions and errors in ASP.NET Core, and diagnosing Entity Framework Core migrations errors.
2. Developer Exception Page Middleware:

app.UseDeveloperExceptionPage()

1. Exception Handler Middleware:

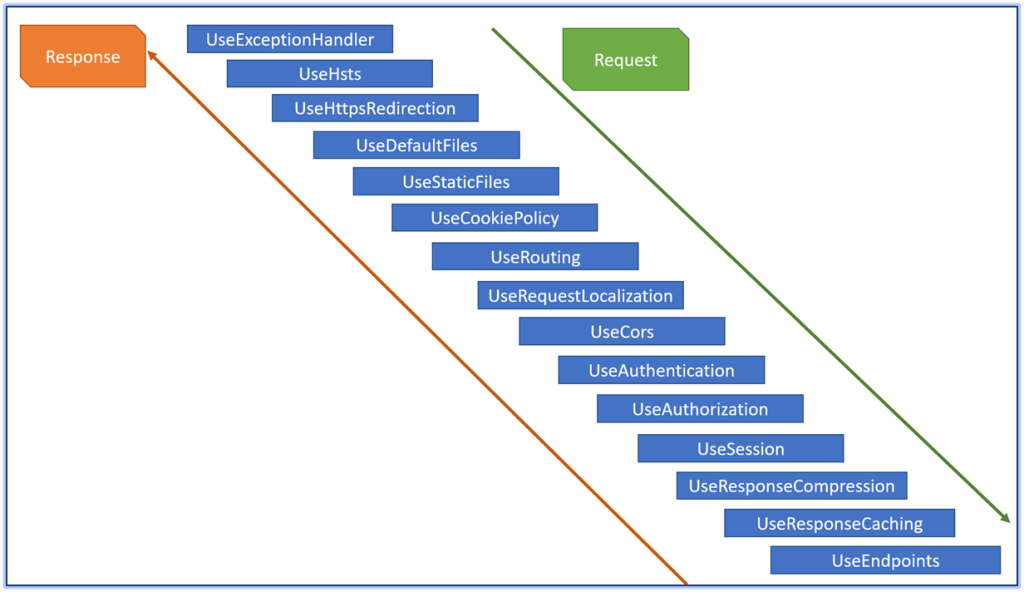
app.UseExceptionHandler()

1. Status Code Pages Middleware

app.UseStatusCodePages()

1. Welcome Page Middleware:

app.UseWelcomePage()



## Routing:

Routing is a pattern matching system that monitors the incoming request and figures out what to do with that request. Typically, it is a way to serve the user request.

Routing = URL + METHOD

There are two main ways to define routes in ASP.NET Core:

1. **Convention Based Routing :**

It creates routes based on a series of conventions, which represent all the possible routes in your system. Convention-based are defined in the Startup.cs file.

routes.MapRoute(

template: "{controller}/{action}/{ id:int?}"

);

1. **Attribute Based Routing:**

It creates routes based on attributes placed on controller or action level. Attribute routing provides us more control over the URLs generation patterns, which helps us in SEO.

[Route("Home")]

public class HomeController : Controller {

[Route("index")]

[Route("")]

[Route("/")]

public IActionResult Index() {

return View();

}

}

1. **Attribute routing token:** We can use token in attribute routes.

[Route("[controller]")]

public class HomeController : Controller {

[Route("[action]")]

public IActionResult Index() {

return View();

}

}

1. **Mixed Routing**: You can use Convention-based Routing and Attribute routing together. Even you should use both together since it is not possible to define attribute route for each and every action or controller. In that case, Convention-based Routing will help you.
2. **Route Constraints:** Route Constraints are used to restrict the type of passed value to an action. For example, if you expect an argument id as an integer type, then you have to restrict it to an integer type by using datatype in the curly brackets as { id : int }.

:int

:bool

:datetime

:decimal

:guid

:length(min,max)

:alpha

:range(min,max)

1. **Optional Parameters**: You can define your route parameter as optional in routes by adding a question mark (?) to the parameter's constraint as given below:

app.UseMvc(routes =>

{

routes.MapRoute(

template: "{controller}/{action}/{ id:int ? }");

});

1. **Default Values:** In addition to route constraints and optional parameters, you can also specify the default values for your route parameters which will be used if values are not provided.

app.UseMvc(routes =>

{

routes.MapRoute(

template: "{controller=Home}/{action=Index}/{id:int?}");

});

## Controller

1. Controller manages the flow of application.
2. A controller is used to defined and group a set of actions.
3. Is responsible for intercepting incoming requests and executing appropriate application code.
4. Controller class should be create with suffix of “Controller”.

**Action Method:**

1. Action method in controller class which are responsible for returning the view or json data.
2. Action method are public method in controller class.

**Data from controller to View:**

1. ViewData:
2. ViewBag:
3. TempData:
4. Stronglytype view:

## View

1. A View provides the user interface of the application.
2. A View is used to display content of an application and accept user inputs.
3. View uses model data to create this UI.
4. **Razor page:**
5. **Comment** :

@\* ……. \*@

1. **Layout property :**

* For add layout property:

Layout = “~/Views/Shared/\_Layout.cshtml”;

* For not add layout property:

Layout = null ;

* Condition layout

If(User.IsInRole(admin)){

Layout = null;

}else{

}

1. **Master page:**

<body>

<header> </header>

<main role="main" class="pb-3">

@RenderBody()

</main>

<footer class="border-top footer text-muted"> </footer>

</body>

1. **\_Layout.cshtml:**
2. \_ViewStart.cshtml :

When we need layout page all the

1. \_ViewImports.cshtml:

## Model

1. Model is C# class, which used for data get and set.
2. We use model with database and collection data.
3. We can insert, update, delete with model.

namespace model.Models{

public class StudentModel {

public int roll { get; set; }

public string name { get; set; }

public int Gender { get; set; }

}

}

**Repository Pattern: -- (21)**

1. Repository Pattern is an abstraction class of data access and the business logic layer of an application.
2. It hides the details of how exactly the data is saved or retrieved from the underlying data source.
3. The details of how the data is stored and retrieved is in the respective repository.

**Strongly Typed View:**

We can transfer data controller to view.

With this we can binds view with model.

We can access model properties on that view.

We can bind view with model multiple class.

Controller:

public IActionResult Index() {

StudentModel std = new List<StudentModel> {

new StudentModel { roll = 1, name = "Test", Gender = 1 },

new StudentModel { roll = 1, name = "SHUVO", Gender = 1 },

};

return View(std);

}

View Page:

@model List<model.Models.StudentModel>

@foreach(var i in Model)

{

<tr>

<td>@i.roll</td>

<td>@i.name</td>

<td>@i.Gender</td>

</tr>

}

Model binding:

# Entity Framework Core

Entity Framework Core is the new version of Entity Framework after EF 6.x. It is open-source, lightweight, extensible and a cross-platform version of Entity Framework data access technology.

Entity Framework is an Object/Relational Mapping (O/RM) framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing & storing the data in the database.

EF Core supports two development approaches

1. **Code-First:**

* In the code-first approach the entity framework core create database objects based on model classes that you create to represent application data.
* Is the most common approach implement in asp.net core mvc .

[key] :

[Column(“Student”, TypeName = “varchar(20)”)]

1. **Database-First.**

**Database provider:**

**DBContext:**

1. DBcontext class is an integral part of entity framework.
2. Dbcontext class we use to interact with database.
3. This class manages the database connection and is used to retrieve and save data in database.
4. An instance of DbContext represent a session with the database
5. Dbcontext is a combination of unit of work.