1. Default Constructor

```
public class vehicle
{
public void makesound()
{
Console.WriteLine("Vehicle Makes Sound"); }
 }
public class program
 {
public static void Main(string[] args)
{
vehicle car = new vehicle();
car.makesound();
}
 }
```

2. Parametrized constructor

```
public class person
 {
private string name;
private int age;
//parametrized constructor public
person(string name, int age)
{
this.name=name;
this.age=age;
}
}
```

3. Destructor

```
public class Vehicle
 {
   public void MakeSound()
   {
 Console.WriteLine("Vehicle
Makes Sound");
}
   ~Vehicle()
   {
Console.WriteLine("Vehicle object
removed.");
   }
}
```

4. Arrays

```
public static void Main (string[] args)
{

int [] array1 = new int[] { 5,6,7,8,9};
int [] array2 = new int[5];

Console.WriteLine("Length of 1st array:" +
array1.Length);
Array.Sort(array1);
PrintArray(array1);

Array.Reverse(array1);
PrintArray(array1);

Array.Copy(array1,array2, array1.Length);
}
```

5. Base Class

```
public class Animal
{
public void Eat()
{ Console.WriteLine("Animal Eats");
}

Public Class Dog : Animal
{
Public void Eat()
{
base.Eat();
Console.Writeline("Dog eats");
}
```

6. Exception Handling

```
Public Class Program
{
public static void main(string[] args)
        {
try
{  int i=20;    int result =i/0;   }
catch( DivideByZeroException ex)
{Console. WriteLine("Error. Attempted  divide

by Zero"); }
catch (Exception ex)
{  Console. WriteLine($" Error:{ex.Message}");
} // This will handle any  other exception type
       } }
```

1. Properties

```
public class person
{
private string name;
private int age;

//property for accessing name field
public string Name
{
get { return name;}
set { name = value;}
}
//property for accessing age field
public int Age
{
get { return age;}
set { age= value;}
}
}
```

2. Indexers

```
public class MyCollection
{
private string[] data = new string[3];
// indexer
public string this[int index]
{
get { return data[index]; }
set { data[index] = value; }
}
}
Public class Program
{
public static void Main(string[] args)
      {
MyCollection collection = new MyCollection();
collection[0]="Ankit";
Console.WriteLine(collection[0]);
      }
}
```

3. Multiple Inheritance (Interface)

```
public interface IHumanwalk
{ void walk(); }
public interface IHumanswim
{ void swim(); }

public class Human : IHumanwalk , IHumanswim
   {
public void walk()
{ Console.WriteLine("Human Walks"); }

public void swim()
{ Console.WriteLine("Human Swims"); }
   }
Public class Program
   {
Public Static void Main ( string[] args) {
Human ankit = new Human();
ankit.walk(); ankit.swim();
 }
   }
```

4. Method Overriding

```
public class Animal {
public virtual void MakeSound()
{
Console.WriteLine("Animal MakeSound");
} }
public class Dog : Animal
{
public override void MakeSound()
 { Console.WriteLine("Dog makes sound"); }
}
public class Program
{
public Static void Main (string [] args)
   {
Animal animal1 = new Animal();
animal1.MakeSound();
Dog dog1 = new Dog(); dog1.MakeSound();
   }
}
```

5. Polymerphism

```
public class Animal {
public virtual void MakeSound()
 { Console.WriteLine("Animal Makes Sound");
}
}
public class Dog : Animal
{
public override void MakeSound()
 {
Console.WriteLine("Dog make sound"); }
}
public class Program
{
public Static void Main (string [] args)
   {
Animal animal1 = new Dog();
animal1.MakeSound(); // Dogmake--
   }
}
```

1. Struct

```
public struct Point
{
Public int X;
Public int Y;
}
public class Program
{
public static void main (string[] args)
   {

Point point1;
point1.X=10;
point1.Y=20;
Console.WriteLine("Coordinates: ({0}, {1}) ",
point1.X, point1.Y );

   }
}
```

2. Enum

```
public enum DaysOfWeek
{
Sunday,
Monday,
Tuesday,
Wednesday,
Thursday,
Friday,
Saturday
}
public class Program
{
public static void Main(string[] args)
   {
DaysOfWeek today = DaysOfWeek.Monday;
Console.WriteLine($"Today is {today}. ");
   }
}
```

3. Abstract Class

```
public abstract class Animal
{
public abstract void MakeSound()
public class Dog : Animal { public override void
MakeSound()
{
Console.WriteLine("Dog Barks");
}
public class Program
{
Public Static void Main(string[] args)
{
Animal animal1 = new Dog(); animal1.MakeSound();
} }
```

4. Sealed Class

```
public Sealed class Animal
{
public abstract void MakeSound()
{
Console.WriteLine("Animal makes sound");
}
public class Dog : Animal   // error
{
}
public class program
{
public static void Main(string[] args)
{
Animal animal1 = new Animal();
animal1.MakeSound();
}
}
```

5. DELEGATES

```
//Declaration
public delegate void SimpleDelegate();
class DelegateTest
{
static void Main (string[] args)
   { //Installation
SimpleDelegate d = MyFunc; // Invocation d();

  } public static void MyFunc()
                {
Console.WriteLine("I was called by a delegate");
      }
}
```

1. Events

```
public class Button
{
   public event EventHandler Click;

   public void ClickButton()
   {
 Console.WriteLine("Button clicked!");
 Click?.Invoke(this, EventArgs.Empty);
   }
}

public class Program
{
   public static void Main(string[] args)
   {
      Button button1 = new Button();

button1.Click += (s, e) =>
Console.WriteLine("Button was clicked!");

button1.ClickButton();
   }
}
```

2. Partial Class

File 1: Part1.cs

```
public partial class Animal
{
public void Dog()
   {
Console.WriteLine("Dog Barks");
   }
}
```

File 2: Part2.cs

```
public partial class Animal
{
public void Cat()
   {
Console.WriteLine("Cat Meows");
   }
}
```

File 3 : Program.cs

```
public class Program
{
public static void Main(string[] args)
   {
Animal animal1 = new Animal();
animal1.Dog(); animal1.Cat();
   }
}
```

3. Collection & Generics

(Note: generics ma list matra rakhne)

```
using System;
using System.Collections.Generic;
public class program
{
public static void main(string[] args)
      {
//list example
List<string> names = new List<string> ();
names.Add("Ankit");
names.Add("Kushal");
names.Add("Sulab");
Console.WriteLine("List of names:");
foreach (string name in names)
{
Console.WriteLine(name);
}
//dictionary example
Dictionary<string,int> ages = new
Dictionary<string,int>{
ages.Add("Ankit", 22);
ages.Add("Kushal",22);
ages.Add("Sulab",23);
      }
}
```

4. LINQ & Lamda expressions

```
using System
using System.Collections.Generic
public class Program
{
public static void main(string[] args)
   {

List <int> numbers = new List <int> {1,7,3,9,5,2};

// LINQ Query to filter even numbers
var evenNumbers =
numbers.Where(num => num % 2 == 0);

Console.WriteLine("Even numbers:");
foreach( var num in evenNumbers)
{
Console.WriteLine(num);
}

   }
}
```

5. Asynchronous Programming

```
public class program
{
public static async Task Main(string[] args)
   {
Console.WriteLine("Start");

//call the asyncronous method & use await to
wait for it to complete

await DoSomethingAsync();
Console.WriteLine("End");
   }

public static async Task DoSomethingAsync()
   {
Console.WriteLine("Async Method Started");

//simulate an asynchronous operation (eg. a
delay) using Task.Delay

await Task.Delay(2000);  // pause for 2 sec
Console.WriteLine("Astnc Method end");
   }
}
```

1. Roles

```
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
 public class HomeController : Controller
      {
 [Authorize(Roles = "Admin")]
public IActionResult AdminDashBoard()
 {
Content("Welcome to the Admin Dashboard!");
}
      }
```

2. Claims & Policies

```
Services.AddAuthorization( options =>
{
options.Addpolicy("AdultOnly", policy =>
policy.RequireClaim("Age","18","19","20")
);
});
```

In this example, the "AdultOnly" Policy requires the user to have an "Age" claim of 18, 19 or 20 to access the associated resources or actions.

3. Securing Action Method in Controller

```
public class HomeController : Controller
{
[Authorize]
public IActionResult About()
{
ViewData["Message"]= "This is my about page";
return View();
}
```

4. XSS (vulnerable code)

```
<div> @foreach(var comment in Model.Comments)
{
<p> @ comment.Content </p>
}
</div>
```

5. SQL INJECTION (vulnerable code)

```
pubic bool ValidateUser(string username, string
passwpord)
 {
string query= " SELECT COUNT(*) FROM USERS
WHERE Username=' "+username+" ' AND Password= '
"+password+" '  ";
```

6. Model Binding

consider a simple model class for user registration

```
public class user
{
public string Username {get; set;}
public string Email {get; set;}
public int Age {get; set;}
}
```

7. Model Validation

```
public class user
{
[Required(ErrorMessage="Username required")]
public string Username {get; set;}

[EmailAddress(ErrorMessage="Invalid  email")]
public string Email {get; set;}

[Range(18,99,ErrorMessage="Age must be between 18 and 99")]
public int Age {get; set;}

}
```

8. SQL INJECTION

```
using System; using System.Data.SqlClient;
class Program
 {
static void Main()
{
Console.Write("Enter a username: "); string username = Console.ReadLine();

 string connectionString = "your_connection_string";
 string query = $"SELECT * FROM Users WHERE Username = '{username}'";

using (SqlConnection connection = new SqlConnection(connectionString))
{
 connection.Open();
SqlCommand command = new SqlCommand(query, connection);
 SqlDataReader reader = command.ExecuteReader();

while (reader.Read())
 {
string userId = reader["UserId"].ToString(); Console.WriteLine("User ID: " +
userId);

} } }
```

1. Cookies

Reading Cookie:

```
//Read cookie from IHttpContextAccessor
string cookieValueFromContext = httpContextAccessor.HttpContext.
                    Request.Cookies["key"];
//Read cookie from Request object
string cookieValueFromReq = Request.Cookies["key"];
```

Writing Cookie:

```
public void SetCookie(string key, string value, int? expireTime)
{
  CookieOptions option = new CookieOptions();
  if (expireTime.HasValue)
      option.Expires = DateTime.Now.AddMinutes(expireTime.Value);
  else
      option.Expires = DateTime.Now.AddMilliseconds(10);

      Response.Cookies.Append(key, value, option);
}
```

Remove Cookie

```
Response.Cookies.Delete(key);
```

2. Query Strings

```
public IActionResult GetQueryString(string name, int age){
    User newUser = new User()
    {
        Name = name;
        Age = age;
    };
    return View(newUser);
}
```

Now we can invoke this method by passing query string parameters:

/welcome/getquerystring?name=John&age=31

3. Hidden fields

```
[HttpGet]
public IActionResult SetHiddenFieldValue(){
    User newUser = new User(){
        Id=101, Name="John", Age=31
    };
    return View(newUser);
}
[HttpPost]
public IActionResult SetHiddenFieldValue(IFormCollection keyValues){
    var id = keyValues["Id"];
    return View();
}
```

4. Http Context

Example: Example to check request processing time using HttpContext class.

⇒This example check the uses of the HttpContext class. In the global.aspx page we know that a BeginRequest() and EndRequest() is executed every time before any Http request. In those events we will set a value to the context object and will detect the request processing time.

```
protected void Application_BeginRequest(object sender, EventArgs e)
{
    HttpContext.Current.Items.Add("BeginTime", DateTime.Now.
    ToLongTimeString());
}
protected void Application_EndRequest(object sender, EventArgs e)
{
  TimeSpan diff = Convert.ToDateTime(DateTime.Now.ToLongTimeString()) -
      Convert.ToDateTime(HttpContext.Current.Items["BeginTime"].ToString());
}
```

5. Session State

Example: Program to demonstrate how to set and read a value from a session.

Controller: HomeController.cs

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
namespace SessionDemo.Controllers
{
 public class HomeController: Controller
 {public IActionResult Index()
   {
    HttpContext.Session.SetString("uname", "Roshan");
    HttpContext.Session.SetString("pwd", "1234567");
    return (RedirectToAction("Get"));
   }
  public IActionResult Get()
  {
   string uname = HttpContext.Session.GetString("uname").ToString();
   string pwd = HttpContext.Session.GetString("pwd").ToString();
   ViewBag.username = uname;
   ViewBag.password = pwd;
   return View();
  }
 }
}
```

View: Get.cshtml

```
<html>
<body>
    Username: @ViewBag.username; <br/>
    Password: @ViewBag.password;
</body>
</html>
```

6. TempData

Controller: HomeController.cs

```
using Microsoft.AspNetCore.Mvc;
namespace TempDataDemo.Controllers
{
 public class HomeController: Controller
 {
  public IActionResult First()
  {
   TempData["uname"] = "Roshan"; //This will continue for the ne
                                  request until it is read
   return (RedirectToAction("Second"));
  }
  public IActionResult Second()
  {
   return View();
  }
  public IActionResult Third()
  {
   return View();
  }
 }
}
```

View: Second.cshtml

```
<html>
<body>
  @*Username: <h1>@TempData["uname"] </h1>
    @{ TempData.Keep();

     }*@

    @{
      var nam = TempData.Peek("uname");
     }
    Username: <h1>@nam </h1>
    @Html.ActionLink("Click me", "Third");
</body>
</html>
```

7. Form Validation: Name validation in using react

```
import React, { useState } from 'react';
const FormComponent = () => {
  const [name, setName] = useState('');
  const [error, setError] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();

    if (!name.trim()) {
      setError('Name is required');
      return;
    }
    console.log('Form submitted:', { name });
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Name"
        value={name}
        onChange={(e) => setName(e.target.value)}
      />
      {error && <div
className="error">{error}</div>}
      <button type="submit">Submit</button>
    </form>
  );
};
export default FormComponent;
```

ADO. Net application to read data from existing table MOVIE(id, name, genre), where genre "comedy".

```
using System;
using System.Data;
using System.Data.SqlClient;
namespace MovieApp
{
  class Program  {
    static void Main(string[] args   {
      string connectionString = "Your_Connection_String"; // Replace with your actual connection string
      try
      {
        using (SqlConnection connection = new SqlConnection(connectionString))  {
          connection.Open();
          string query = "SELECT * FROM MOVIE WHERE genre = 'comedy'";
          SqlCommand command = new SqlCommand(query, connection);
          SqlDataReader reader = command.ExecuteReader();

          while (reader.Read())  {
            int id = Convert.ToInt32(reader["id"]);
            string name = reader["name"].ToString();
            string genre = reader["genre"].ToString();
            Console.WriteLine($"ID: {id}, Name: {name}, Genre: {genre}");
          }
          reader.Close();}  }
      catch (Exception ex) {  Console.WriteLine("Error: " + ex.Message);  }    Console.ReadKey();  } } }
```

method to insert record (3, "John , 12000")to table Employee having fields EmployeeId(int), Name varchar(200), Salary(int) using Entity Framework.

```
using System;
using Microsoft.EntityFrameworkCore;
// Define the Employee model class
class Employee {
  public int EmployeeId { get; set; }
  public string Name { get; set; }
  public int Salary { get; set; }  }
class Program {
  static void Main()  {
    // Create a new instance of DbContextOptions with the connection string
    var options = new DbContextOptionsBuilder<EmployeeDbContext>()
    .UseSqlServer("your_connection_string")
    .Options;
  // Create a new instance of EmployeeDbContext with the options
    using (var dbContext = new EmployeeDbContext(options))
    {
      // Create a new Employee record
      var newEmployee = new Employee
      { EmployeeId = 3, Name = "John",  Salary = 12000
      };
      // Add the new Employee record to the context
      dbContext.Add(newEmployee);   dbContext.SaveChanges();
Console.WriteLine("Record inserted successfully.");  } } }
```