1. Write a program to print "Hello world, I am student of Bsc.CSIT".

```
using System;
public class HelloWorld {
    public static void Main (string[] args) {
        Console.WriteLine("Hello world, I am student
        of Bsc.CSIT");
    }
}
```

2. Write a program - To check the input year is leap or not.

```
using system;
public class checkleapyear{
    public static void Main(string[] args){
        Console.WriteLine("Enter a year:");
        int year = Convert.ToInt32(Console.Readline());
        if((year%4==0 && year%100!=0) ||(year%400==0)){
            Console.Writeline("{0} is a leapyear",year);
        }
        else{
            Console.Writeline("{0} is not a leap
                year.",year);
        }
}
```

3. Write a program for the given condition with the appropriate approach (loop if, switch,) also explain why did you use the specific approach.

a. Age >= 18, "person can get citizenship & vote"
b. Age >= 16, "person can get citizenship but cannot vote".
c. Age < 16, "Person cannot get citizenship"

```
Using System;
class
public class Program{
  public static void Main(string[] args){
    Console.WriteLine("Enter your age:");
    int age = Convert.ToInt32(Console.ReadLine());
    if age >= 18){
      Console.WriteLine("you can get citizenship & vote");
    }else if( age >= 16 ){
      Console.WriteLine("You can get citizenship but
          cannot vote");
    }
    else{
      Console.WriteLine("Person cannot get citizenship");
    }
  }
}
```

Explanation: In this code, I have used if..else statement to check for different age conditions. If age >= 18, then the person can get citizenship and ~~other~~ vote (like this). I chose to use an if else statement because it is simple and efficient way to handle multiple conditions in a program.

4. write a program to display sum of n natural numbers, factorial and reverse of a number.

```
using system;
public class MathOperations {
    public int SumofNaturalNumbers (int n) {
        int sum = 0;
        for (int i = 1; i <= n; i++) {
            sum += i;
        }
        return sum;
    }
    public int Factorial (int n) {
        int factorial = 1;
        for (int i = 1; i <= n; i++) {
            factorial *= i;
        }
        return factorial;
    }
    public int ReverseNumber (int num) {
        int reversedNum = 0;
        while (num > 0) {
            reversedNum = reversedNum * 10 + num % 10;
            num /= 10;
        }
        return reversedNum;
    }
}
public class Program {
    public static void Main (string[] args) {
        MathOperations obj = new MathOperations();
        int n = 5;
        Console.WriteLine ("sum of natural numbers up to {n} : {obj.SumofNaturalNumbers(n)}");
```

```
int factorialNum = 4;
Console.WriteLine($" Factorial of {factorialNum}:
        {obj.Factorial(factorialNum)} ");

int numToReverse = 12345;
Console.WriteLine($" Reverse of {numToReverse} :
        {obj.ReverseNumber(numToReverse)} ");

    }

}
```

5. Write a program to display even digits and odd digits of any number.

```
public class program{
  public static void Main(string[] args){
    Program obj = new Program();
    int n = Convert.ToInt32(Console.ReadLine());
    obj.sum(n);
  }
  public void sum(int n){
  int evenSum=0,oddsum=0,rem;
  while(n!=0){
     rem = n%10;
     if(rem%2==0){
        evensum = evenSum + rem;
     }
     else{
        oddsum = oddsum+rem;
     }
  }
  Console.WriteLine("Evensum : "+evensum);
  Console.WriteLine($"Oddsum : {oddsum}");
}
}
```

8. Write a program to find smallest element in array and search element in array.

```
public class Array {

    public void Smallest (int [] arr, int e) {
        int smallest = arr[0];
        for ( int i = 1; i < arr.length ; i++) {
            if ( smallest > arr[i] ) {
                smallest = arr[i];
            }
        }
        Console.WriteLine("smallest :" + smallest);
    }
    public void SearchElement (int [] arr, int e) {
        int found = 0;
        for ( int i = 0; i < a.length ; i++) {
            if ( e == arr[i] ) {
                found = 1;
                break;
            }
        }
        if (found == 1) {
            Console.WriteLine(" Element found ");
        } else {
            Console.WriteLine(" Element Not found ");
        }
    }
    public static void Main (string[] args) {
        int arr = { 10, 12, 5, 6, 7 };
        Array objArr = new Array();
        objArr.Smallest ( arr );
        int searchElement = 6;
        objArr.SearchElement ( arr, searchElement);
    }
}
```

7. Write a program, create a class named "vehicle" with attribute members travelled distance, fuelused, no of seats, action to steer. Add two methods to calculate the kilometers that can be travelled with 1 litre fuel, and the amount of fuel needed to travel 100km. Extend the class vehicle by making two new classes Motercycle and Car. Two methods added should work for motorcycle and car class. Two methods added should work for no object creation, the class Car has to set to 2. Create a method called steering which takes parameter called direction, when this method is called, it has to set no of seats to 5 and the motorcycle class has to set to 2. Create the method called steering which takes parameter called direction, when this method is called, it has to set the variable action to steer to "lean over <direction>" for class motercycle and "rotate steering wheel <direction>" for class car. Use concept of oop properties, indexes, constructors whenever applicable.

```
using system;
public class Vehicle {
    private double distanceTravelled;
    private double fuelUsed;
    public int NumSeats { get; set; }
    protected string steerAction;

    public Vehicle() {
        NumSeats = 0;
        steerAction = "";
    }
    public void Travel ( double distance, double fuel) {
        distanceTravelled += distance;
        fuelUsed += fuel;
    }
```

```
public double kmPerLiter(){
    if (fuelUsed == 0){
        return 0;
    }
    return distanceTravelled / fuelUsed;
}

public double fuelPer100km(){
    if (distanceTravelled == 0){
        return 0;
    }
    return fuelUsed / (distanceTravelled / 100);
}

public void steering(String direction){
    steerAction = direction;
}

}
```

8. WAP for exception handling

```
using System;
public class Program {
    public void Compute(int a, int b) {
        try {
            int div = a / b;
            Console.WriteLine("Division is :" + div);
        } catch (DividedBy2ero Exception e) {
            Console.WriteLine(e.Message);
        }
        catch (Exception e) {
            Console.WriteLine("Base exception");
        }
    }
public static void Main(string[] args) {
    Program p = new Program();
            p.Compute(10, 0);
    }
}
```

9. WAP for generic class

```
using System;
public class Student<T>{
    public T data;
    public student(T data){
        this.data = data;
        Console.WriteLine("Data : "+data);
    }
}
public class Test{
    public static void Main(string[] args){
        Student<string> s = new Student<string>("Sita");
        Student<int> s1 = new Student<int>(1);
    }
}
```

10)

Create a simple console application in visual studio (Book Inventery system, Quiz Application, student Information System)

Requirements:

a. use of every concept studied in unit 1 (as far possible)
b. No need to focus on UI, just functionality is needed
c. Console Application
d. choose any one from above

=>

```
using System;
using System.Collections.Generic;
public class student{
    public int Age{get; set;}
    public string Gradelevel{get; set;}
    public string Name{get; set;}

    public student(string name, int age, string gradelevel){
            Name = name;
            Age = age;
            Gradelevel = gradelevel;
        }
    }
    public void Display(){
        Console.WriteLine("Name: " + Name);
        Console.WriteLine("Age: " + Age);

    }
}
class Program{
    public static void Main(string[] args){
    List<student> students = new List<student>();
    while(true){
        Console.WriteLine("1. Add student");
        Console.WriteLine("2. Display All students");
        Console.WriteLine("Enter choice:");
        string choice = Console.ReadLine();
```

```
switch(choice){
    case "1":
        Console.Write("Enter name: ");
        string name = Console.ReadLine();
        Console.Write("Enter age: ");
        int age = Convert.ToInt32(Console.ReadLine());
        Console.Write("Enter grade level: ");
        string gradelevel = Console.ReadLine();
        Student student = new Student(name, age,
            gradelevel);
        Students.Add(student);
        Console.WriteLine("student added successfully");
        break;
    case "2":
        if(students.Count == 0){
            Console.WriteLine("No students added yet");
        }
        else{
            Console.WriteLine("student list:");
            foreach(Student s in students){
                s.Display();
                Console.WriteLine();
            }
        }
        break;
    case "3":
        Console.WriteLine("Good Bye!");
        return;
    default:
        Console.WriteLine("Invalid choice, please
            try again");
        break;
}
Console.WriteLine();
}
}
}
```

11. Write a program to handle a file in C#
a. Create and write to a file.
b. Check wheather file exists.
c. Read a content from a file.

```
using System;
using System.IO;

namespace FileHandlingExample{
    class Program{
        public static void Main(string[] args){
            string fileName = "example.txt";
            // create and write to a file
            WriteToFile(fileName, "Hello, world!");

            // check whether file exists
            if(FileExists(fileName)){
                Console.WriteLine("File exists.");
            }
            else{
                Console.WriteLine("file doesn't exists.");
            }

            // Read content from file
            string content = ReadFromFile(fileName);
            Console.WriteLine($"file content: {content}");
        }

        static void WriteToFile(string fileName, string content)
        {
            using(StreamWriter writer = File.CreateText(fileName)
            ){
                writer.WriteLine(content);
            }

            Console.WriteLine($"content written to {fileName}
                ", "");
        }
```

```
static bool FileExists(string fileName){
    return File.Exists(fileName);
}

static string ReadFromFile(string fileName){
    using (StreamReader reader = File.OpenText(fileName))
    {
        return reader.ReadToEnd();
    }
}
```

12. Write a program in LINQ syntax

a. First create list of strings with string values
b. Get and show all string from list with LINQ
c. Filter and show the list containing the matching string text

```
using system;
using System.Collections.Generic;
using System.linq;

namespace SimpleLINQExample {
  class Program {
    public static void Main(string[] args) {
      List<string> MyList = new List<string>{
          "Aryan","apple","banana","cat","dog"};
      Console.WriteLine("Strings:");
        foreach(string names in MyList){
            Console.WriteLine(names);
        }
      string searchText = "an";
      Console.WriteLine($"In Strings containing '{searchT-
            ext}':");
       var matchingStrings= MyList.Where(names=> names.
          Contains(searchText));
       foreach(string names in matchingstrings){
          Console.WriteLine(names);
       }

    }
}
```

13) Create an ASP.NET core MVC project and perform following task

a) Create a controller Home with 3 action methods:
   i. Index → this should return string
   ii. Details → pass some value and show this into corresponding view page
   iii. Show → use ViewData and ViewBag

b) In your project add new controller Display with one action needed to send id numeric value and string value to your view page. Then using razor syntax display the string value to id number of time

c) Now add model in your project with these property members - FullName, Email, Salary
   i) Add data annotation on model
   ii) Add necessary model validation on each property members

d) On any one of the above controller, create action methods with HttpGet and HttpPost
   i) Show how can you build the model
   ii) Check the model state is valid or not.
   iii) Show validation message in your view page.

step-1: Create a New Asp.NET core MVC project

step-2: Create controllers and Actions

1. Home Controller

```
using Microsoft.AspNetCore.Mvc;
public class HomeController : Controller {
    public IActionResult Index() {
        return Content("This is Index action");
    }
    public IActionResult Details(string value) {
        ViewBag.Message = value;
        return View();
    }
    public IActionResult Show() {
        ViewData["message"] = "This message is from
                               viewData.";
        ViewBag.Message = "This message is from
                           viewBag";
        return View();
    }
}
```

2. Display Controller

```
using Microsoft.AspNetCore.Mvc;
public class DisplayController : Controller {
    public IActionResult DisplayDate(int id, string value) {
        ViewBag.Id = id;
        ViewBag.Value = value;
        return View();
    }
}
```

Step 3 : Create Views

1. Details.cshtml (Views | Home) ;

```
@{
    ViewData["Title"] = "Details";
}
<h2> Details </h2>
<p> Value from action : @ViewBag.Message </p>
```

2. Show.cshtml (Views | Hom) ;

```
@{
    ViewData["Title"] = "Show";
}
<h2> Show </h2>
<p> ViewData Message : @ViewData["Message"] </p>
<p> ViewData Message : @ViewBag.Message </p>
```

3. DisplayData.cshtml (Views | Display) ;

```
@{
    ViewData["Title"] = "Display Data";
}
<h2> Display Data </h2>
@{
    int id = ViewBag.Id;
    String value = ViewBag.Value;
    for (int i=0; i<id; i++) {
        <p> @ value </p>
    }
}
```

Step 4: Create Model and validation

1. Models.cs (Model)

```
using System.ComponentModel.DataAnnotations;
public class Models {
    [Required (ErrorMessage:" full Name is required.")]
     public string FullName {get; set}
     [Required (ErrorMessage:" Email is required.")]
     [EmailAddress (ErrorMessage="Invalid email address.")]
     public string Email {get; set;}
     [Required (ErrorMessage = "salary is required.")]
     [Range(0, double.MaxValue, ErrorMessage:" salary
                must be a positive number.")]
      public decimal Salary {get; set;}
}
```

Step 5: Add Controller Actions for Model Binding validation

1. Home controllers

```
[HttpGet]
public IActionResult ModelBinding() {
     return View();
}
[HttpPost]
public IActionResult ModelBinding (Models model) {
    if (ModelState.IsValid) {
        ViewBag.Message = "Model is valid.";
    }
    else {
        ViewBag.Message = "Model is not valid";
    }
    return View();
}
```

step 6: Create views for Model Binding and validation

1. ModelBinding.cshtml (Views/Home):

```
@{
    ViewData["Title"] = "Model Binding";
}

<h2> Model Binding </h2>
<form method="post">
  <div>
    <input type="
    <label for="FullName"> Full Name: </label>
    <input type="text" id="fullname" name="fullName" />
    <span asp-validation-for="fullName"> </span>
  </div>
  <div>
    <label for="Email"> Email : </label>
    <input type="email" id="Email" name="Email" />
    <span asp-validation-for="Email" e> </span>
  </div>
  <div>
    <label for="salary"> salary : </label>
    <input type="number" id="salary" name="salary" />
    <span asp-validation-for="salary" > </span>
  </div>
  <button type="submit" e> Submit </button>
</form>

<p> @ViewBag.message </p>
```

step 7: Run the Application

14. Write a program that shows the use of ADO.NET in your ASP.NET Core MVC project.

a. Create Database and Table Employee(id, employee name, department, salary, joined date) in sql server.

b. Create a form to get user data input and insert into table.

c. Show data from your table in your list page.

d. Update data through your edit page.

e. Delete a row of data from your given data.

=> Step 1:

First we created a database 'EmployeeDB' and a table named 'Employee' with the columns 'Id', 'Name', 'Development Department', 'salary' and 'Joined Date'

Step 2:

We created a model named 'Employee' to represent the table structure.

```
public class Employee {
    public int Id { get; set; }
    public int Id { get; } string Name { get; set; }
    public string Department { get; set; }
    public DateTime Joined Date { get; set; }
}
```

Step 3:

We created an 'EmployeeController' with actions for listing employee.

```
Using System;
Using System.Data;
using System.Data.SqlClient;
Using Microsoft.AspNetCore.Mvc;
Using Ems.Controllers;
```

```
namespace EMS.Controllers{
public class EmployeeController : Controller {
    private readonly string connectionString=" MY-CONNECTION ]
                    String";
    public IActionResult Index() {
        using (SqlConnection connection= new SqlConnection(
                                connectionString)) {

            connection.Open();
            string query=" SELECT * FROM Employee ";
            SqlCommand command= new SqlCommand(
                                query, connection);
            SqlDataReader reader= command.ExecuteReader
                                ();
            var employees= new List<Employee>();
            while (reader.Read()) {
                employees.Add(new Employee
                    {
                    Id = Convert.ToInt32(reader["Id"]),
                    Name= reader["Name"].ToString(),
                    Department= reader["Department"].ToString(),
                    Salary = Convert.ToDecimal(reader["salary"]),
                    JoinedDate= Convert.ToDateTime(reader["Joined
                                                    Date"])
                    });
            }
            return View(employee);
        }
    }
}
```

step 4

we created views for listing employees, creating new employee, editing existing employees and confirming deletion

```
@model List<Employee>
    <table>
        <tr>
          <th> Id </th>
          <th> Name </th>
          <th> Department </th>
          <th> Salary </th>
          <th> Joined Date </th>
        </tr>
        @foreach (var employee in Model) {
            <tr>
              <td> @ employee.Id </td>
              <td> @ employee.Name </td>
              <td> @.employee.Department </td>
              <td> @ employee.Salary </td>
              <td> @ employee.JoinedDate .ToShortDate
                    String () </td>
            </tr>
    </table>
```

15

Write a program to show statement management using Session to set session and retrive session value in your Asp.Net core

step1: Configure session in Program.cs

Before: app = builder.Build()

builder.Services.AddDistributedMemoryCache();
builder.Services.AddSession(options => options.IdleTimeout = TimeSpan.FromMinutes(1));

~~builder.Services.AddMvc();~~
~~builder.Services~~

After: app=builder.Build()

app.UseSession();

step2: Creating ~~Cookie Cont~~ SessionController to set and retrive session value

```
Using Microsoft.AspNetCore.Mvc;

namespace statemenagement.Controllers;

public class SessionController:Controller{
    public IActionResult ~~Index()~~ setsession(){
        Http.Context.Session.SetString("Name","Raju");
        HttpContext.Session.SetInt32("Id",106);

        return View();
    }

    public class IAction ViewSession(){
        ViewBag.Name = HttpContext.Session.GetString("Name");
        ViewBag.Id = HttpContext.Session.GetInt32("Id");

        return View();
    }

}
```

method.

step 3: Creating Views:

for

// Session/Setsession.cshtml

```
<p> Session value has been set!! </p>
<a asp-action="Viewsession"> View session </a>
```

// Session/Viewsession.cshtml

```
@{
    ViewData["Title"] = "View session";
    var name = ViewBag.Name;
    var id = ViewBag.Id;
}
<!DOCTYPE html>
<html>
  <body>
    <h1> @ViewData["Title"] </h1>
    <p> Using session to get the information!!! </p>
    <div>
      <p> Name: @name </p>
      <p> Id: @id </p>
    </div>
  </body>
</html>
```

16. Write a program to show statemanagement using ~~session~~ Cookie. To set ~~session~~ cookie and retrive ~~session~~ cookie value in your Asp.Net core program.

step1: program.cs

```
builder.Service.AddHttpContextAccessor();
```

step 2: Create cookieController

```
using Microsoft.AspNetcore.Mvc;
namespare stateManagement.controllers;
public class CookieController : Controller {
    privale IHttpContextAccesser accesser;
    public CookieController(IHttpContextAccesser accesser){
        this.accesser = accesser;
    }
    public IActionResult Writecookie() {
        Response.Cookies.Append("name","John Doe");
        return View();
    }
    public IActionResult Readcookie() {
        string name = accesser.HttpContext.Request.
                      Cookies["name"];
        ViewBag.name = name;
        return View();
    }
}
```

3: Views /

/Cookie/~~ReadCookie.cshtml~~ WriteCookie.cshtml

```
<div>
    <h3> Cookie is set. </h3>
    <a asp-action="ReadCookie"> Read Cookie </a>
</div>
```

/Cookie/~~Write~~ReadCookie.cshtml

```
<h3> Cookie value </h3>
<p> @ViewData["name"] </p>
```

17) Write a sample program to show use of TempData from one action method to action method 2:

1. Controller

```
using Microsoft.AspNetCore.Mvc;
namespace statemanagement.Controllers;
public class TempDataController : Controller {
    public IActionResult Index() {
        TempData["userId"] = 106;
        return View();
    }
    public IActionResult one1() {
        var userId = TempData["userId"] ?? null;
        return View();
    }
    public IActionResult one2() {
        var userId = TempData["userId"] ?? null;
        return View();
    }
}
```

2. Views / TempData

one.cshtml

```
@{
    var userId = TempData["userId"]?.ToString();
}
<h4> Data from one1 </h4>
<p>
    User Id : @userId
</p>
<a asp-action = "one1"> one1 </a>
<a asp-action = "one2"> one2 </a>
```

one2.cshtml

```
@{
    ViewData["Title"] = "one2";
    var userId = TempData["UserId"]?.ToString();
}
<h4>Data from one2</h4>
<p>User Id : @userId</p>
<a asp-action="one1">one1</a>
<a asp-action="one2">one2</a>
```

4)

Write a program in ASP.NET core to process Querystring.

Controller

QuerystringController.cs

```
using Microsoft.AspNetCore.Mvc;
using statement Management.Models
namespace statemanagement.controllers;

public class QuerystringController : Controller {
    public IActionResult UserInfo(string name, Int32 id){
        User newUser = new User(){
            Name = name,
            Id = id
        };
        return View(newUser);
    }
}
```

model.

User.cs

```
namespace stateManagement.models;
public class User {
    public string Name {get; set;}
    public Int32 Id {get; set;}
}
```

Views

Querystring\UserInfo.cshtml

```
@model User
<!DOCTYPE html>
<html>
<body>
    <h4> Set User Information </h4>
    <form method="post" action="@Url.Action("UserInfo",
                                "Querystring")">
        <label for="name"> Name: </label>
```

```
<input type="text" id="name" name="name" />
<br/>
<label for="id">Id:</label>
<input type="number" id="id" />
<button type="submit">submit</button>
</form>
@if(!string.IsNullOrEmpty(Model.Name) && Model.Id!=0){
    <h1>User Info:</h1>
    <p>Name: @Model.Name</p>
    <p>Id: @Model.Id</p>
}
</body>
</html>
```