DAY 18 : Assignment
By
Vihar D.

Assignment 1

What are the uses of XML ?

Answer :

- XML is a simple text based format for representing structured information.
- It is a data transfer mechanism used for transferring data to different platforms, therefore it is not platform dependent.
- Users can define their own tags based on their requirements.
- It requires high storage capacities due to redundancy of its syntax, hence transferring of high capacities of data costs a lot.

Assignment 2

Write all the points discussed about XML in the session.

Answer :

- XML is short for eXtensible Markup Language.
- It is case sensitive.
- It is not platform dependent.
- It consists mostly of user defined tags.
- It has only one root tag for the data entry.
- The syntax is redundant as compared to other text based transmission formats like JSON.

Assignment 3

Create a simple XML to illustrate : 1. Tag based XML with 10 employees 2. Attribute base XML.

Answer :

1. Tag Based XML :

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```xml
▼<Employees>
  ▼<Employee>
     <ID>1</ID>
     <Name>Vihar</Name>
     <Salary>10000</Salary>
   </Employee>
  ▼<Employee>
     <ID>2</ID>
     <Name>Sarath</Name>
     <Salary>20000</Salary>
   </Employee>
  ▼<Employee>
     <ID>3</ID>
     <Name>Lokesh</Name>
     <Salary>30000</Salary>
   </Employee>
  ▼<Employee>
     <ID>4</ID>
     <Name>Manoj</Name>
     <Salary>40000</Salary>
   </Employee>
  ▼<Employee>
     <ID>5</ID>
     <Name>Prudhvi</Name>
     <Salary>50000</Salary>
   </Employee>
  ▼<Employee>
     <ID>6</ID>
     <Name>Bhanu</Name>
     <Salary>50000</Salary>
   </Employee>
  ▼<Employee>
     <ID>7</ID>
     <Name>Praveen</Name>
     <Salary>60000</Salary>
   </Employee>
  ▼<Employee>
     <ID>8</ID>
     <Name>Sudheer</Name>
     <Salary>65000</Salary>
   </Employee>
  ▼<Employee>
     <ID>9</ID>
     <Name>Vamsi</Name>
     <Salary>70000</Salary>
   </Employee>
  ▼<Employee>
     <ID>10</ID>
     <Name>Siva</Name>
     <Salary>80000</Salary>
   </Employee>
 </Employees>
```

2. Attribute Based XML :

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
▼<Employees>
   <Employee ID="1" Name="Vihar" Salary="10000"/>
   <Employee ID="2" Name="Sarath" Salary="15000"/>
   <Employee ID="3" Name="Lokesh" Salary="20000"/>
   <Employee ID="4" Name="Manoj" Salary="25000"/>
   <Employee ID="5" Name="Prudhvi" Salary="30000"/>
   <Employee ID="6" Name="Bhanu" Salary="35000"/>
   <Employee ID="7" Name="Praveen" Salary="40000"/>
   <Employee ID="8" Name="Sudheer" Salary="45000"/>
   <Employee ID="9" Name="Vamsi" Salary="50000"/>
   <Employee ID="10" Name="Siva" Salary="55000"/>
 </Employees>
```

Assignment 4

Convert the above XML to JSON (JavaScript Object Notation) and display the JSON data.

Answer :

XML to JSON format :


```
[
  {
    "@ID": "1",
    "@Name": "Vihar",
    "@Salary": "10000"
  },
  {
    "@ID": "2",
    "@Name": "Sarath",
    "@Salary": "15000"
  },
  {
    "@ID": "3",
    "@Name": "Lokesh",
    "@Salary": "20000"
  },
  {
    "@ID": "4",
    "@Name": "Manoj",
    "@Salary": "25000"
  },
  {
    "@ID": "5",
    "@Name": "Prudhvi",
    "@Salary": "30000"
  },
  {
    "@ID": "6",
    "@Name": "Bhanu",
    "@Salary": "35000"
  },
  {
    "@ID": "7",
    "@Name": "Praveen",
```

```
    "@Salary": "40000"
  },
  {
    "@ID": "8",
    "@Name": "Sudheer",
    "@Salary": "45000"
  },
  {
    "@ID": "9",
    "@Name": "Vamsi",
    "@Salary": "50000"
  },
  {
    "@ID": "10",
    "@Name": "Siva",
    "@Salary": "55000"
  }
]
```


Assignment 5
Research and write all the benefits of JSON over XML.
Answer :
• JSON files occupy less file space as compared to XML. • It is faster than XML since it is designed for data transmission. • JSON encoding is crisp hence uses less file space for transferring it.

Assignment 6

For the below requirement , create a layered architecture project with separate class library for business logic.

Create console application
Create desktop application

<u>Business Requirements</u> :

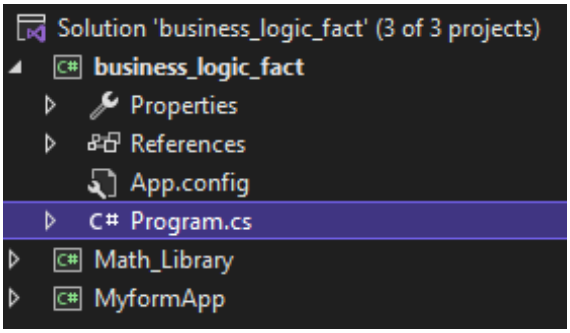
Find factorial of a number :

If 0 = 1,
any positive number (upto 7) = factorial answer
> 7 = -999 (as answer)
> 0 = -9999 (as answer)

Paste the screenshots of the output and the project (solution explorer).

Answer :

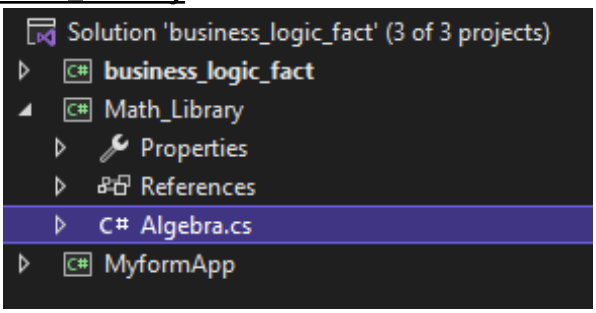
<u>Console App</u> :




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Math_Library;     //Class library

namespace business_logic_fact
{
    internal class Program
    {
        static void Main(string[] args)
        {
            int n;
```

```
        Console.WriteLine("\n Enter number :");
        n = Convert.ToInt32(Console.ReadLine());

        Console.WriteLine("\n Factorial is : {0}", Algebra.Factorial(n));

        Console.ReadLine();
    }
  }
}
```

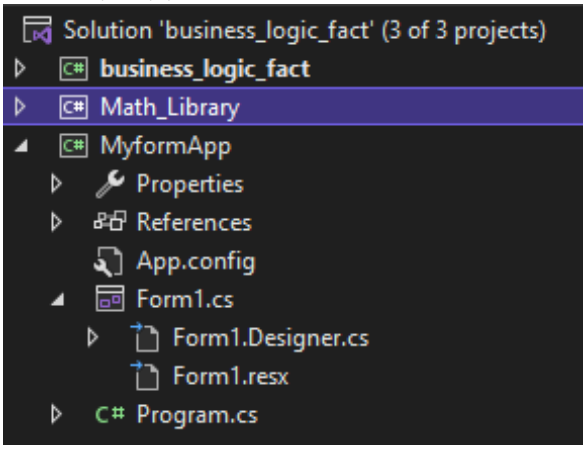
Math_Library :




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Math_Library
{
  public class Algebra
  {
    public static int Factorial(int n)
    {
      int fact = 1;
      if (n == 0)
        return 1;
      else if (n > 7)
        return -999;
      else if (n < 0)
        return - 9999;
      else
      {
```

```
            for (int i = 1; i <= n; i++)
            {
                fact *= i;
            }
            return fact;
        }
    }
  }
}
```

Desktop Application :

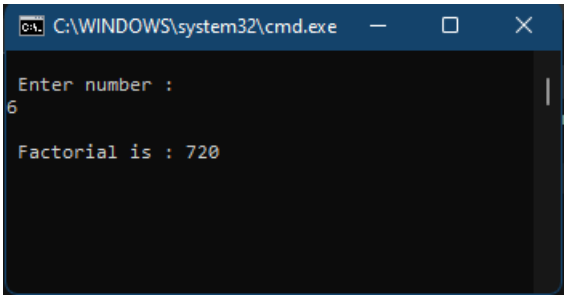


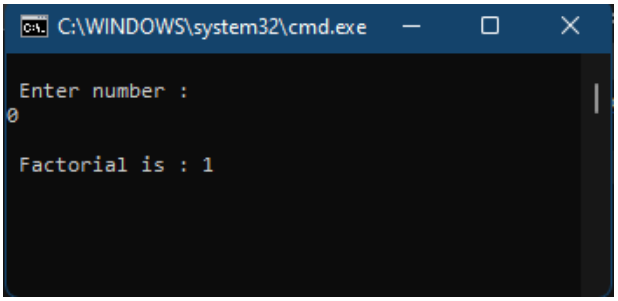

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

using Math_Library;

namespace MyformApp
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
```

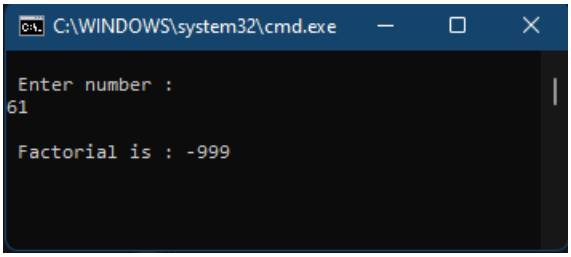
```
    }

    private void button1_Click(object sender, EventArgs e)
    {
        int n = Convert.ToInt32(textBox1.Text);
        int result = Algebra.Factorial(n);

        textBox2.Text = result.ToString();
    }
  }
}
```

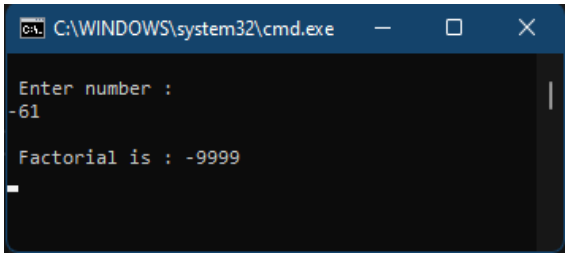
Output :

Console Application Outputs :

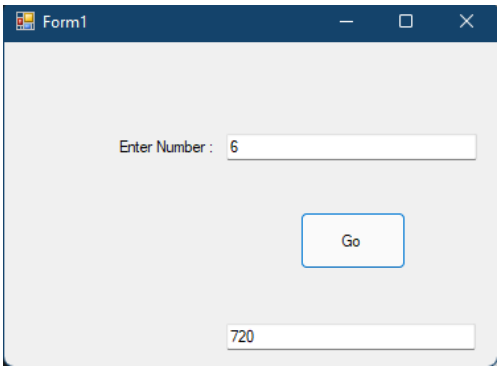


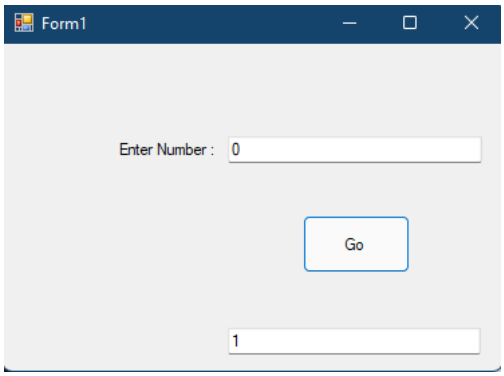


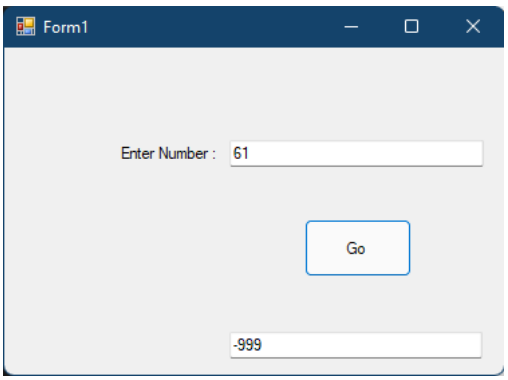


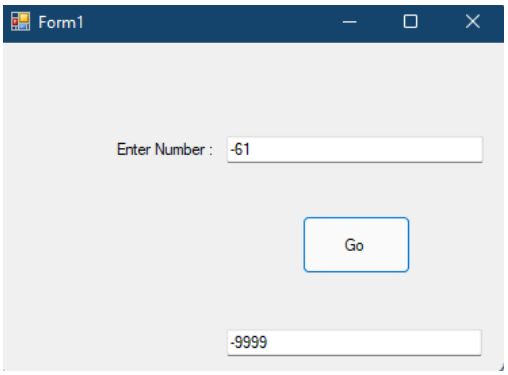


Desktop Application Outputs :






Form1
Enter Number :
61
Go
-999


Form1
Enter Number :
-61
Go
-9999

Assignment 7

For the above method, implement TDD (Test Driven Development) and write 4 test cases and paste the code
Also, paste the screenshots of all the failing test cases and make them pass with a screenshot of it.

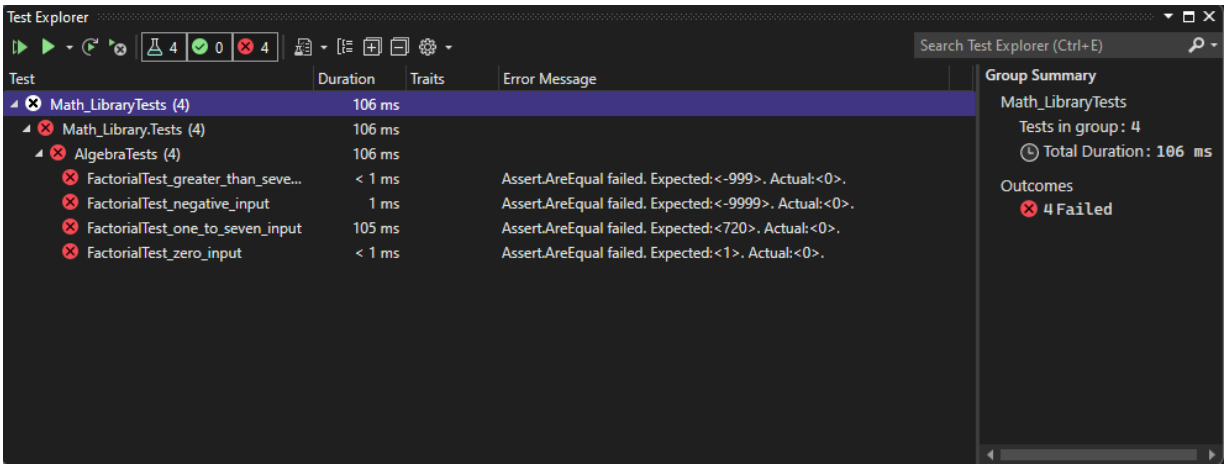
Answer :

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Math_Library;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Math_Library.Tests
{
    [TestClass()]
    public class AlgebraTests
    {
        [TestMethod()]
        public void FactorialTest_zero_input()
        {
            //Arrange
            int n = 0;
            int expected = 1;

            //Act
            int actual = Algebra.Factorial(n);

            //Assert
            Assert.AreEqual(expected, actual);

            //Assert.Fail();
        }
        [TestMethod()]
        public void FactorialTest_one_to_seven_input()
        {
            //Arrange
            int n = 6;
            int expected = 720;
```

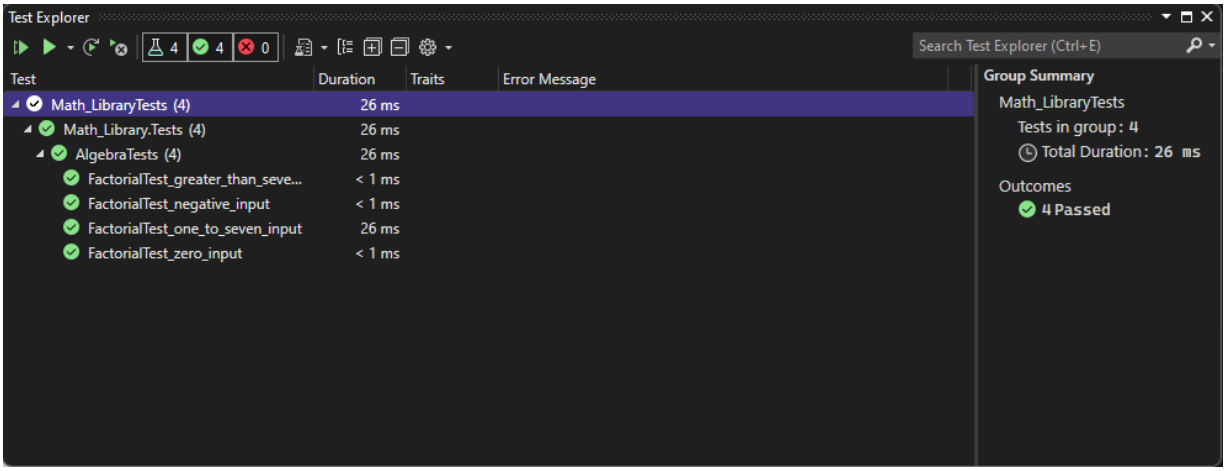
```
        //Act
        int actual = Algebra.Factorial(n);

        //Assert
        Assert.AreEqual(expected, actual);
    }
    [TestMethod()]
    public void FactorialTest_negative_input()
    {
        //Arrange
        int n = -61;
        int expected = -9999;

        //Act
        int actual = Algebra.Factorial(n);

        //Assert
        Assert.AreEqual(expected, actual);
    }
    [TestMethod()]
    public void FactorialTest_greater_than_seven_input()
    {
        //Arrange
        int n = 61;
        int expected = -999;

        //Act
        int actual = Algebra.Factorial(n);

        //Assert
        Assert.AreEqual(expected, actual);
    }
  }
}
```

Output :

All test cases failed.



All test cases succeeded.



Assignment 8

Add one more method to check if the number is palindrome or not in the above Algebra class and write test cases for the same.

Answer :

Algebra Library :

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Math_Library
{
    public class Algebra
    {
        public static int Factorial(int n)
        {
            int fact = 1;
            if (n == 0)
                return 1;
            else if (n > 7)
                return -999;
            else if (n < 0)
                return - 9999;
            else
            {
                for (int i = 1; i <= n; i++)
                {
                    fact *= i;
                }
                return fact;
            }
            //return 0;
        }
        public static bool IsPalindrome(int n)
        {
            int rev = 0, rem, x;
            x = n;
            while(x > 0)
            {
```

```
            rem = x % 10;
            x /= 10;
            rev = rev * 10 + rem;
        }
        if (n == rev)
            return true;
        else
            return false;
    }
  }
}
```

AlgebraTests :

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using Math_Library;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Math_Library.Tests
{
    [TestClass()]
    public class AlgebraTests
    {
        [TestMethod()]
        public void FactorialTest_zero_input()
        {
            //Arrange
            int n = 0;
            int expected = 1;

            //Act
            int actual = Algebra.Factorial(n);

            //Assert
            Assert.AreEqual(expected, actual);

            //Assert.Fail();
        }
        [TestMethod()]
```

```
public void FactorialTest_one_to_seven_input()
{
    //Arrange
    int n = 6;
    int expected = 720;

    //Act
    int actual = Algebra.Factorial(n);

    //Assert
    Assert.AreEqual(expected, actual);
}
[TestMethod()]
public void FactorialTest_negative_input()
{
    //Arrange
    int n = -61;
    int expected = -9999;

    //Act
    int actual = Algebra.Factorial(n);

    //Assert
    Assert.AreEqual(expected, actual);
}
[TestMethod()]
public void FactorialTest_greater_than_seven_input()
{
    //Arrange
    int n = 61;
    int expected = -999;

    //Act
    int actual = Algebra.Factorial(n);

    //Assert
    Assert.AreEqual(expected, actual);
}
[TestMethod()]
public void Palindrome_test_input()
{
```

```
        //Arrange
        int n = 14541;
        bool expected = true;

        //Act
        bool actual = Algebra.IsPalindrome(n);

        //Assert
        Assert.AreEqual(expected, actual);
    }
  }
}
```

Output :