**Week 1: Design Patterns and Principles**

**Question 1 :Exercise:1 Implementing the Singleton Pattern**

**Logger.cs**

using System;

public class Logger

{

    private static Logger instance;

    private static readonly object lockObj = new object();

    private Logger()

    {

        Console.WriteLine("Logger initialized.");

    }

    public static Logger GetInstance()

    {

        if (instance == null)

        {

            lock (lockObj)

            {

                if (instance == null)

                {

                    instance = new Logger();

                }

            }

        }

        return instance;

    }

    public void Log(string message)

    {

        Console.WriteLine($"[LOG] {message}");

    }

}

class Program

{

    static void Main(string[] args)

    {

        Logger logger1 = Logger.GetInstance();

        logger1.Log("This is the first log message.");

        Logger logger2 = Logger.GetInstance();

        logger2.Log("This is the second log message.");

        if (logger1 == logger2)

        {

            Console.WriteLine("Both logger instances are the same. Singleton works!");

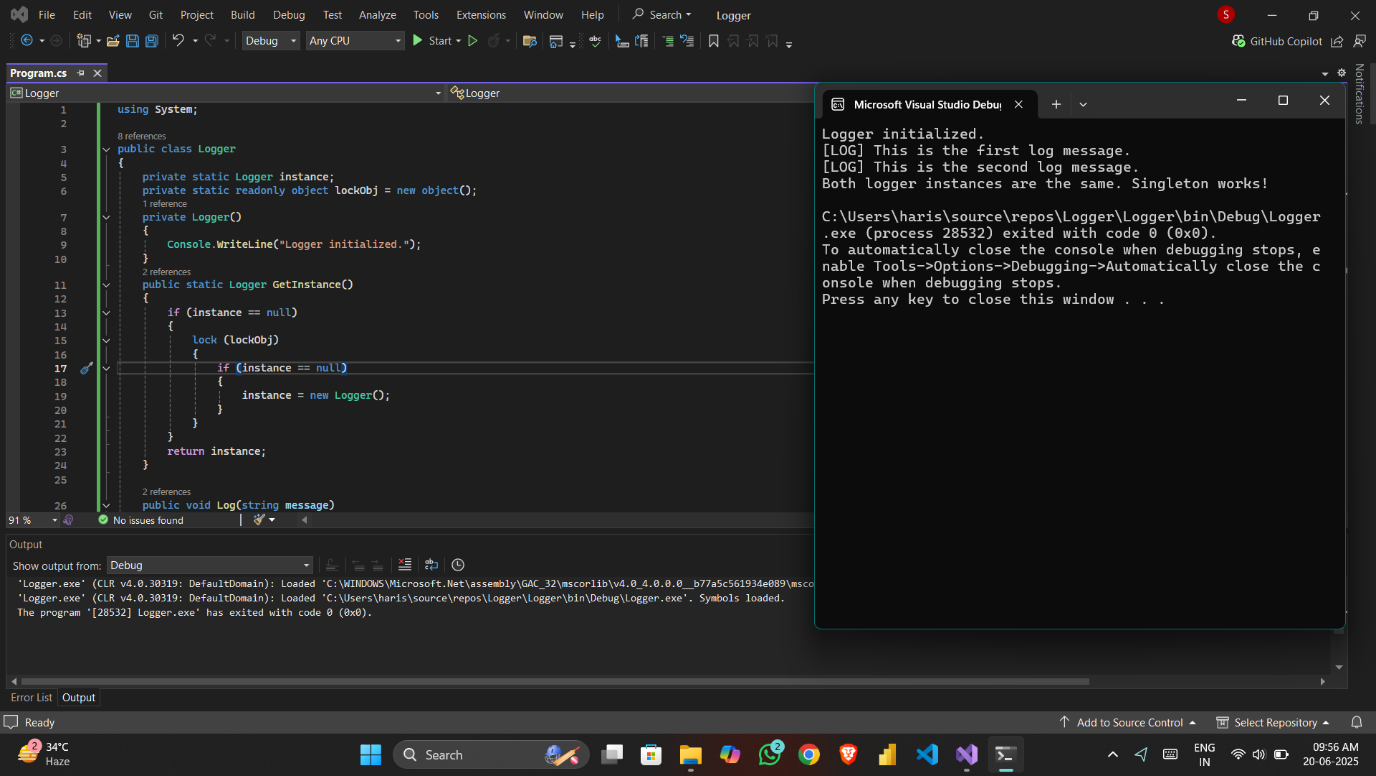
        }

        else

        {

            Console.WriteLine("Different instances. Singleton failed.");

        }

**Output:** 

**Question 2 : Exercise:2 Implementing the Factory Method Pattern**

**Code:**

**using System;**

**public interface IDocument**

**{**

**void Open();**

**}**

**public class WordDocument : IDocument**

**{**

**public void Open()**

**{**

**Console.WriteLine("Opening a Word Document.");**

**}**

**}**

**public class PdfDocument : IDocument**

**{**

**public void Open()**

**{**

**Console.WriteLine("Opening a PDF Document.");**

**}**

**}**

**public class ExcelDocument : IDocument**

**{**

**public void Open()**

**{**

**Console.WriteLine("Opening an Excel Document.");**

**}**

**}**

**public abstract class DocumentFactory**

**{**

**public abstract IDocument CreateDocument();**

**}**

**public class WordDocumentFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new WordDocument();**

**}**

**}**

**public class PdfDocumentFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new PdfDocument();**

**}**

**}**

**public class ExcelDocumentFactory : DocumentFactory**

**{**

**public override IDocument CreateDocument()**

**{**

**return new ExcelDocument();**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**DocumentFactory wordFactory = new WordDocumentFactory();**

**IDocument word = wordFactory.CreateDocument();**

**word.Open();**

**DocumentFactory pdfFactory = new PdfDocumentFactory();**

**IDocument pdf = pdfFactory.CreateDocument();**

**pdf.Open();**

**DocumentFactory excelFactory = new ExcelDocumentFactory();**

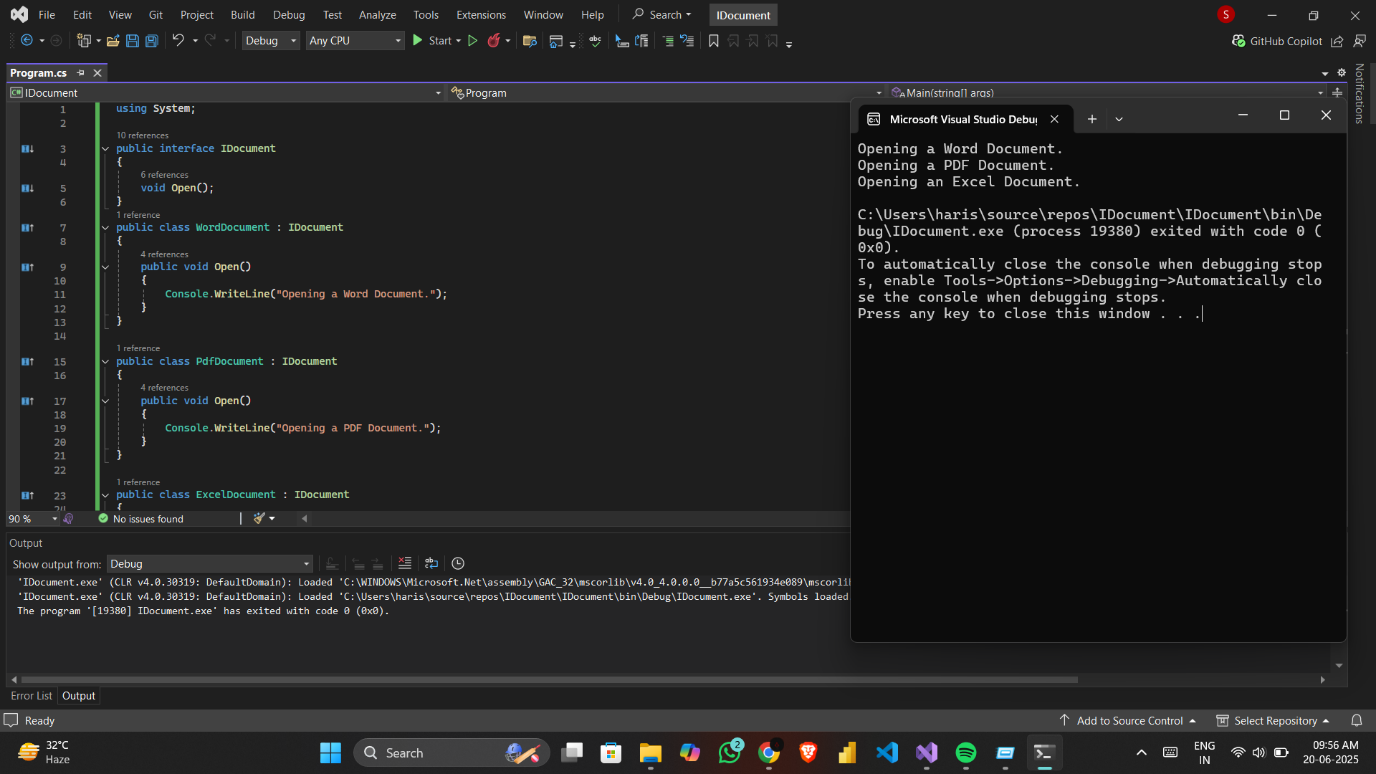
**IDocument excel = excelFactory.CreateDocument();**

**excel.Open();**

**}**

**}**

**Output:**

****

**Algorithms and Data Structures**

**Question 3: Exercise: 2 E-commerce Platform Search Function**

**Code:**

**Program.cs:**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**namespace ECommerceSearch**

**{**

**public class Product**

**{**

**public int ProductId { get; set; }**

**public string Name { get; set; }**

**public string Category { get; set; }**

**public double Price { get; set; }**

**public Product(int id, string name, string category, double price)**

**{**

**ProductId = id;**

**Name = name;**

**Category = category;**

**Price = price;**

**}**

**public void Display()**

**{**

**Console.WriteLine($"ID: {ProductId}, Name: {Name}, Category: {Category}, Price: ${Price}");**

**}**

**}**

**class Program**

**{**

**static void Main(string[] args)**

**{**

**List<Product> products = new List<Product>()**

**{**

**new Product(101, "banana leaf", "utensils", 999.99),**

**new Product(102, "oculus quest 4", "Electronics", 899.99),**

**new Product(103, "asus tuf gaming", "Computers", 1299.50),**

**new Product(104, "bata shoe", "Fashion", 149.99),**

**new Product(105, "Apple Watch", "Accessories", 399.99)**

**};**

**Console.Write("Enter product name to search: ");**

**string searchQuery = Console.ReadLine().Trim().ToLower();**

**var results = products**

**.Where(p => p.Name.ToLower().Contains(searchQuery))**

**.ToList();**

**if (results.Count > 0)**

**{**

**Console.WriteLine("\n Search Results:");**

**foreach (var product in results)**

**{**

**product.Display();**

**}**

**}**

**else**

**{**

**Console.WriteLine("\n No products found matching your search.");**

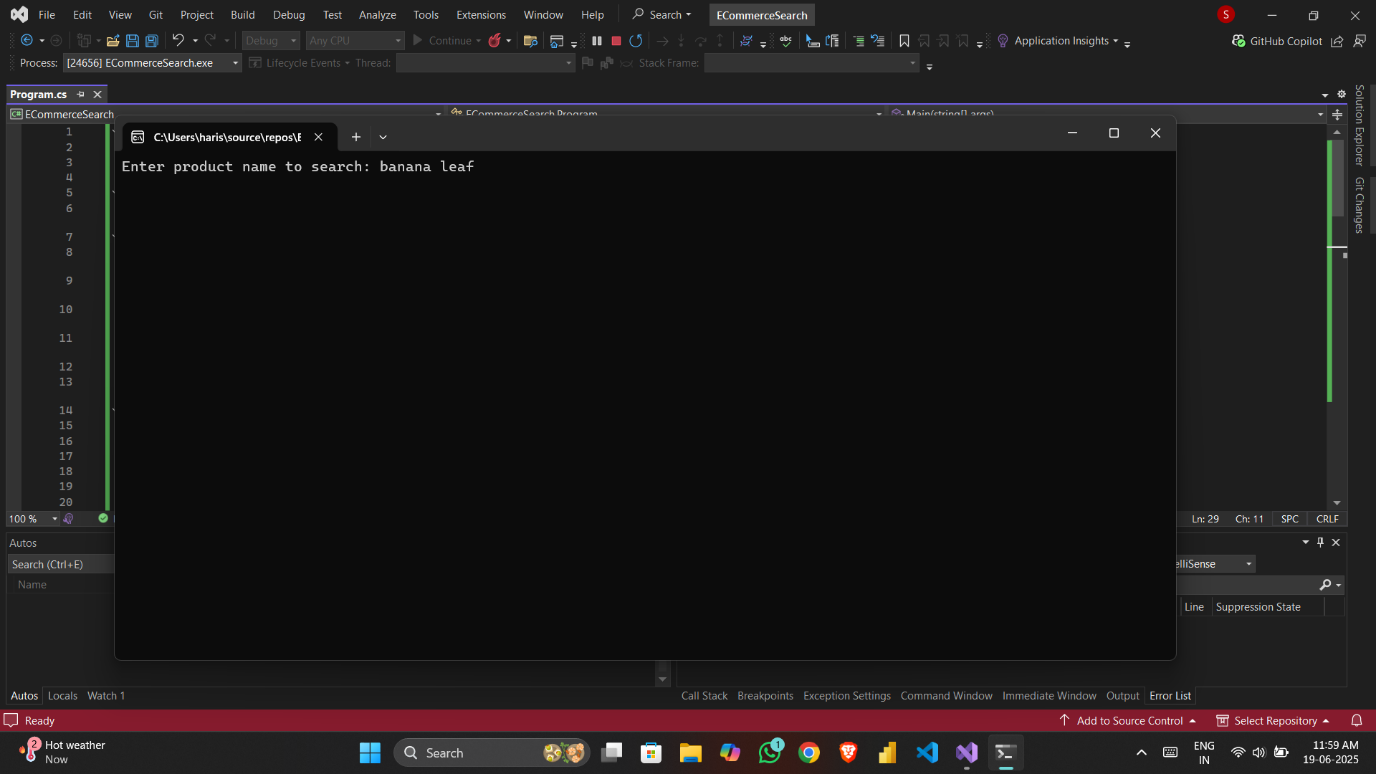
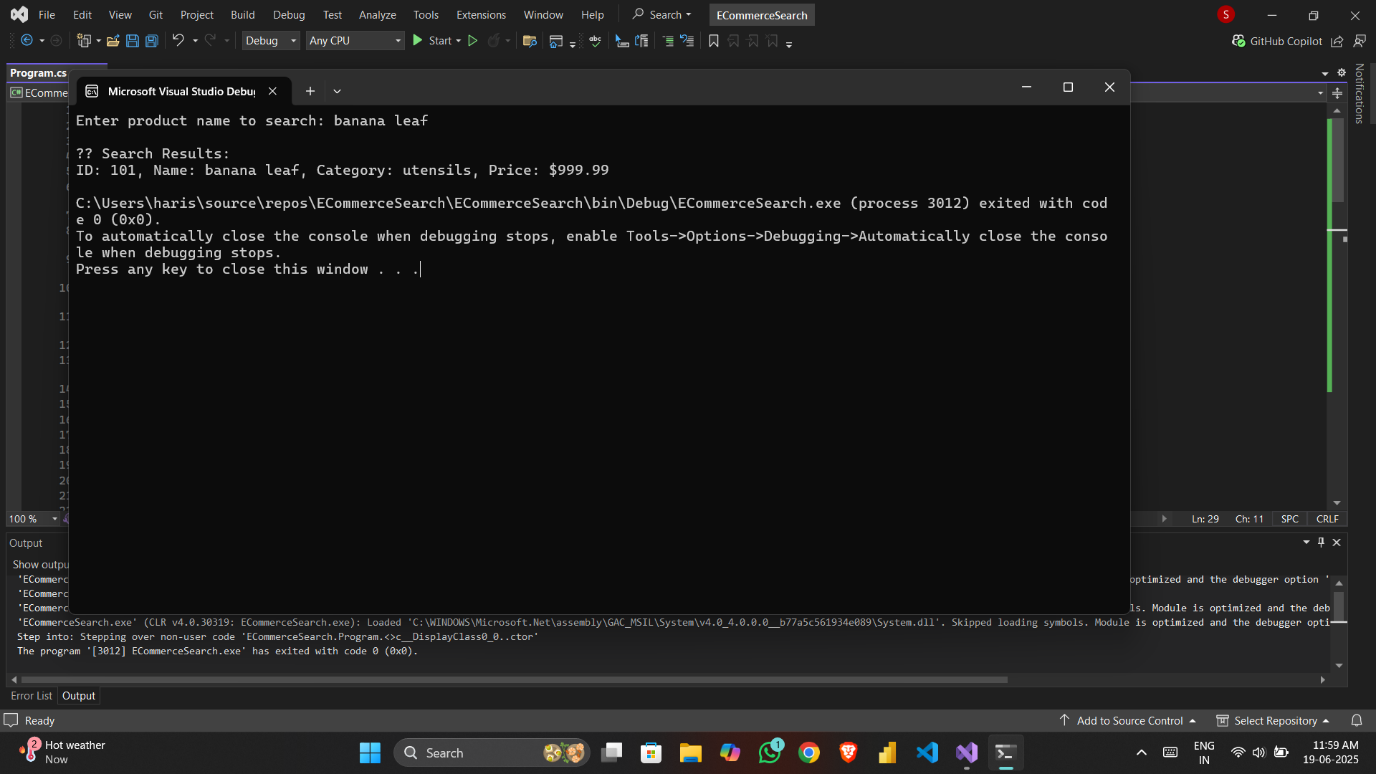
**}**

**}**

**}**

**}**

**Output:**



**Question 4: Exercise: 7 Financial Forecasting**

**CODE:**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**namespace FinancialForecasting**

**{**

**class Program**

**{**

**static double CalculateMovingAverage(List<double> incomeData, int period)**

**{**

**if (incomeData.Count < period)**

**{**

**Console.WriteLine("Not enough data to forecast.");**

**return 0;**

**}**

**var recentMonths = incomeData.Skip(incomeData.Count - period).ToList();**

**double sum = recentMonths.Sum();**

**return sum / period;**

**}**

**static void Main(string[] args)**

**{**

**List<double> monthlyIncome = new List<double>()**

**{**

**120.5, 130.0, 110.75, 145.0, 155.25, 160.0, 170.5, 165.75, 180.0, 175.25, 190.0, 200.5**

**};**

**Console.Write("Enter the number of months for moving average ");**

**int period = int.Parse(Console.ReadLine());**

**double prediction = CalculateMovingAverage(monthlyIncome, period);**

**if (prediction > 0)**

**{**

**Console.WriteLine($"\nPredicted income for next month: ₹{prediction:0.00}K");**

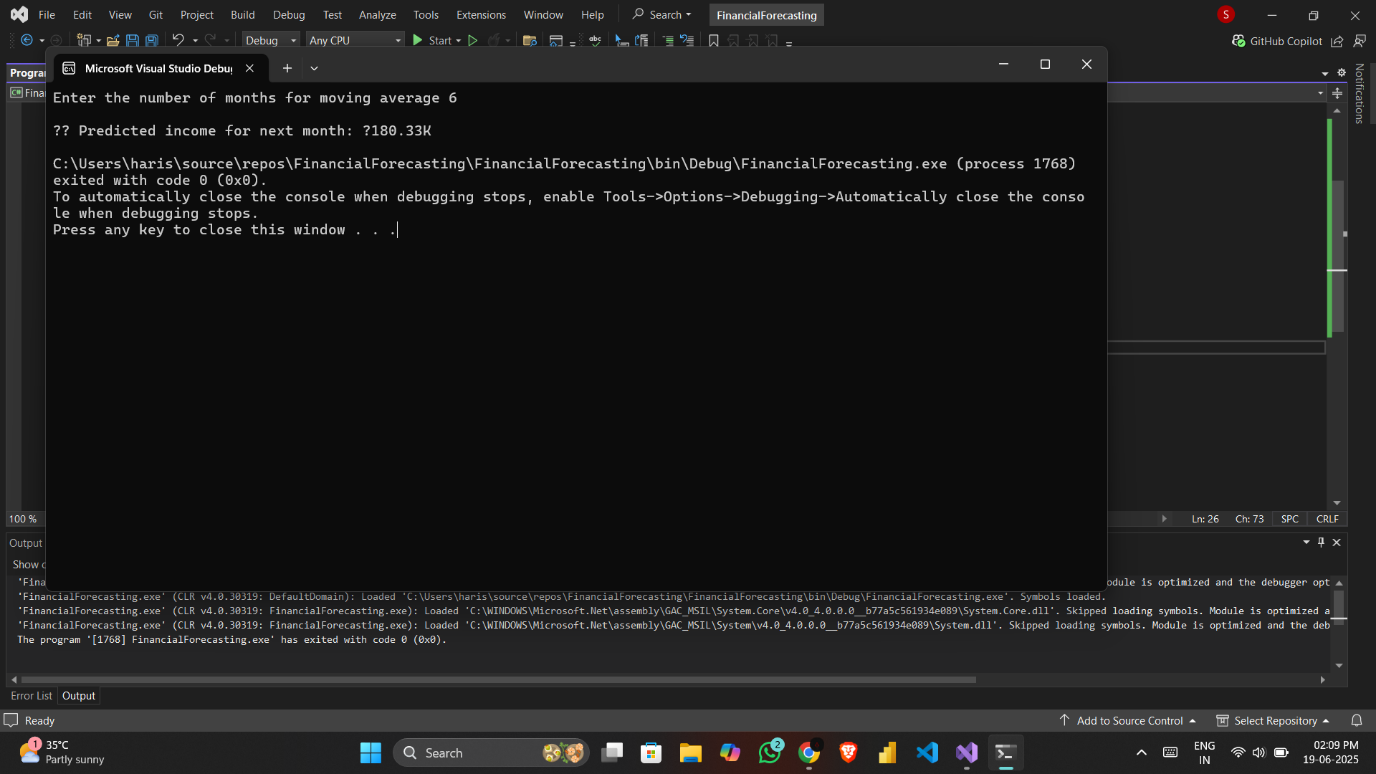
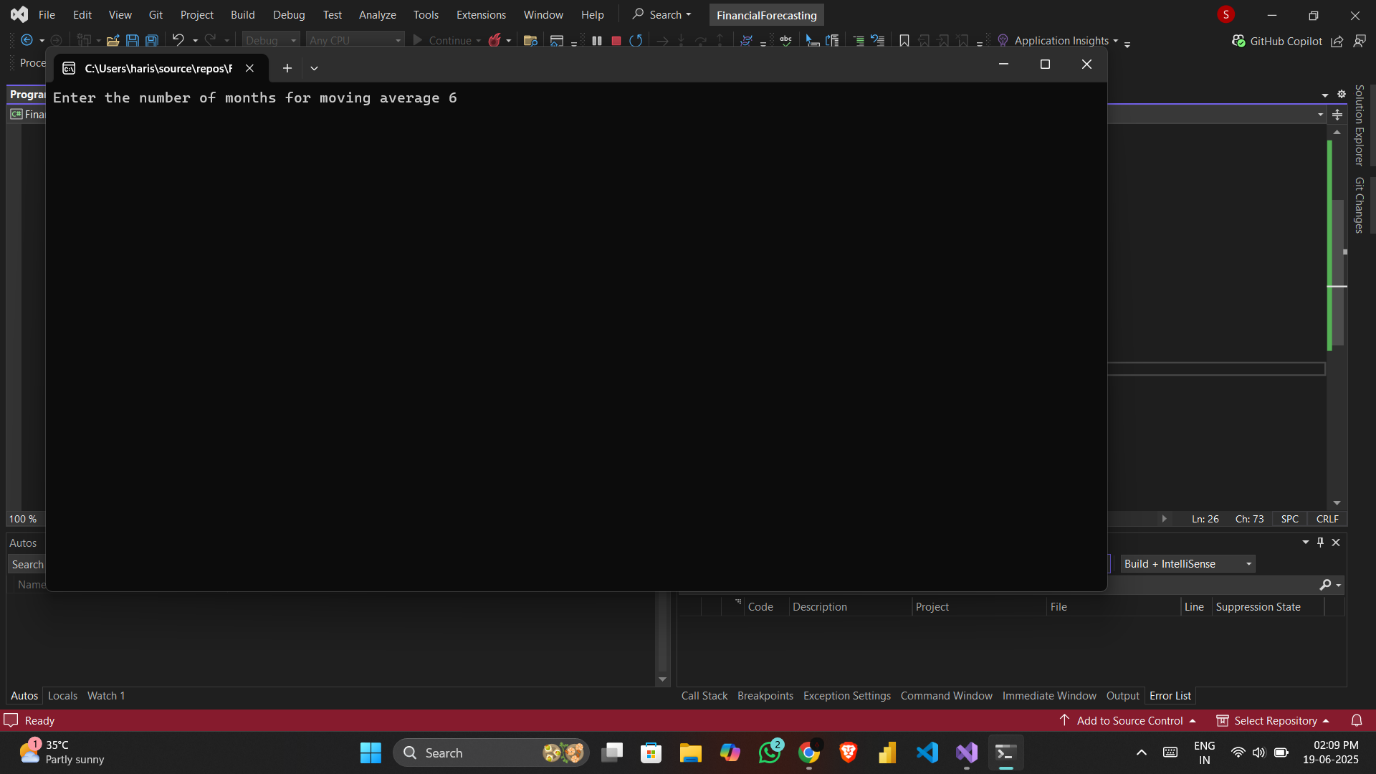
**}**

**}**

**}**

**}**

**Output:**

****