Tarea 1

Iteradores sincrónicos y asincrónicos:

Los iteradores sincrónicos y asincrónicos te permiten recorrer colecciones de datos de manera eficiente. Por ejemplo, imagina que estás trabajando en un proyecto de IoT con sensores que generan un flujo constante de datos. Puedes usar un iterador para tomar muestras de cada enésimo elemento de esos datos.

Ejemplo de iterador en C#:

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main()
    {
        foreach (var number in EvenSequence(5, 18))
        {
            Console.Write(number + " ");
        }
        // Salida: 6 8 10 12 14 16 18
        Console.ReadKey();
    }

    private static IEnumerable<int> EvenSequence(int firstNumber, int lastNumber)
    {
        // Devuelve números pares en el rango.
        for (int number = firstNumber; number <= lastNumber; number++)
        {
            if (number % 2 == 0)
            {
                yield return number;
            }
        }
    }
}
```

En este ejemplo, el método **EvenSequence** es un iterador que devuelve números pares en un rango específico. La palabra clave **yield** se utiliza para generar los valores uno por uno.

Tarea 2

Indexador básico

Supongamos que tienes una clase llamada **TempRecord** que almacena las temperaturas en grados Fahrenheit registradas en diferentes momentos durante un período de 24 horas. Puedes implementar un indexador en esta clase para acceder a las temperaturas como si fueran elementos de una matriz:

```
class TempRecord
{
    private float[] temps = new float[10]; // Almacena las temperaturas

    // Indexador: permite acceder a las temperaturas como si fueran elementos de una matriz
    public float this[int index]
    {
        get { return temps[index]; }
        set { temps[index] = value; }
    }
}
```

```
// Uso del indexador
var tempRecord = new TempRecord();
tempRecord[4] = 72.5f; // Asigna una temperatura
float temperature = tempRecord[4]; // Recupera la temperatura
```

El uso del indexador simplifica la sintaxis para los clientes y hace que la clase sea más intuitiva.

Indexador en una interfaz

También puedes declarar indexadores en interfaces. Por ejemplo:

```
public interface IIndexInterface
{
    // Indexador de solo lectura
    int this[int index] { get; }
}
```

```
// Implementación de la interfaz
class IndexerClass : IIndexInterface
{
    private int[] arr = new int[100];

    public int this[int index]
    {
        get => arr[index];
        set => arr[index] = value;
    }
}

// Uso del indexador
var indexerInstance = new IndexerClass();
indexerInstance[0] = 42; // Asigna un valor
int value = indexerInstance[0]; // Recupera el valor
```

En este ejemplo, **IIndexInterface** define un indexador de solo lectura. La clase **IndexerClass** implementa esta interfaz y proporciona un indexador que permite acceder a los valores almacenados en un arreglo interno.

Recuerda que los indexadores son una herramienta poderosa para trabajar con colecciones de datos.

Tarea 3

Los árboles de expresión en C# son una herramienta poderosa para representar y manipular expresiones de código en forma de estructuras de datos en forma de árbol.

Creación de un árbol de expresión simple

Supongamos que queremos representar la expresión matemática: (x * x) + (y * y) utilizando árboles de expresión. Aquí está cómo puedes hacerlo:

```
using System;
using System.Linq.Expressions;

class Program
{
    static void Main()
    {
        // Crear parámetros para x e y
        var xParameter = Expression.Parameter(typeof(double), "x");
        var yParameter = Expression.Parameter(typeof(double), "y");

        // Crear nodos para las operaciones
        var xSquared = Expression.Multiply(xParameter, xParameter);
        var ySquared = Expression.Multiply(yParameter, yParameter);
        var sum = Expression.Add(xSquared, ySquared);

        // Crear una expresión lambda
        var distanceCalc = Expression.Lambda<Func<double, double, double>>(sum, xParameter, yParameter);

        // Compilar y ejecutar la expresión
        var calculateDistance = distanceCalc.Compile();
        double result = calculateDistance(3.0, 4.0); // Debería ser 5.0 (teorema de Pitágoras)
        Console.WriteLine($"Distancia: {result}");
    }
}
```

En este ejemplo, creamos un árbol de expresión que representa la distancia entre dos puntos en un plano cartesiano utilizando el teorema de Pitágoras.

Tarea 4

LINQ (Language Integrated Query) es una poderosa característica de .NET que permite consultar y manipular datos de manera elegante y eficiente. Aquí tienes algunos ejemplos prácticos de uso de LINQ en Visual Studio:

Consulta de números pares

Supongamos que tenemos una lista de números y queremos obtener solo los números pares. Podemos usar LINQ para filtrarlos:

```
using System;
using System.Linq;

class Program
{
    static void Main()
    {
        int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        var evenNumbers = numbers.Where(n => n % 2 == 0);

        Console.WriteLine("Números pares:");
        foreach (var num in evenNumbers)
        {
            Console.Write(num + " ");
        }
    }
}
```

En este ejemplo, utilizamos el método Where para filtrar los números pares de la lista.

Consulta de cadenas en una lista

Supongamos que tenemos una lista de nombres y queremos obtener aquellos que comienzan con la letra “A”:

```
using System;
using System.Linq;
using System.Collections.Generic;
```

```
class Program
{
    static void Main()
    {
        List<string> names = new List<string>
        {
            "Alice", "Bob", "Anna", "Alex", "John"
        };

        var filteredNames = names.Where(name => name.StartsWith("A"));

        Console.WriteLine("Nombres que comienzan con 'A':");
        foreach (var name in filteredNames)
        {
            Console.WriteLine(name);
        }
    }
}
```

Aquí utilizamos el método Where junto con StartsWith para filtrar los nombres que comienzan con "A".

Consulta de objetos personalizados

Supongamos que tenemos una lista de objetos Person con propiedades como Name y Age. Queremos obtener las personas mayores de 30 años:

```
using System;
using System.Linq;
using System.Collections.Generic;

class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}

class Program
{
    static void Main()
```

```
    {
        List<Person> people = new List<Person>
        {
            new Person { Name = "Alice", Age = 25 },
            new Person { Name = "Bob", Age = 40 },
            new Person { Name = "Carol", Age = 32 }
        };

        var olderPeople = people.Where(p => p.Age > 30);

        Console.WriteLine("Personas mayores de 30 años:");
        foreach (var person in olderPeople)
        {
            Console.WriteLine($"{person.Name}, {person.Age} años");
        }
    }
}
```

Aquí utilizamos el método Where para filtrar las personas mayores de 30 años.

Recuerda que LINQ es una herramienta poderosa para trabajar con colecciones de datos.

Tarea 5

Crear un proyecto en ASP.NET Core utilizando Visual Studio Code

Instalación de requisitos previos

1. Asegúrate de tener instalado Visual Studio Code en tu máquina.
2. También necesitarás el SDK de .NET Core. Puedes descargarlo desde aquí.

Creación de una aplicación de consola de .NET Core:

1. Abre Visual Studio Code.
2. Crea una carpeta para tu proyecto (por ejemplo, "MiProyectoASPNET").
3. Abre el terminal en Visual Studio Code (puedes hacerlo seleccionando Ver > Terminal en el menú principal).
4. Ejecuta el siguiente comando para crear un proyecto de aplicación de consola de .NET Core:
    - dotnet new console --framework net8.0 --use-program-main

Esto creará una estructura básica para tu proyecto en la carpeta que creaste.

Exploración del archivo Program.cs:

1. Abre el archivo Program.cs en la carpeta de tu proyecto.
2. Verás un código similar al siguiente

```
using System;

namespace MiProyectoASPNET
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("¡Hola desde ASP.NET Core!");
        }
    }
}
```

Este es el punto de entrada de tu aplicación. El método Main es donde comienza la ejecución.

Ejecución de la aplicación

Desde el terminal, ejecuta el siguiente comando para compilar y ejecutar tu aplicación

dotnet run

Deberías ver el mensaje "¡Hola desde ASP.NET Core!" en la consola.

¡Listo! Has creado tu primer proyecto en ASP.NET Core utilizando Visual Studio Code. Ahora puedes comenzar a agregar más funcionalidades y explorar las posibilidades de desarrollo web con ASP.NET Core.