Tarea 1

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, "ProyectoDelegadosEventosLambdas").
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.

Ejercicio 1: Delegados

Crea un programa que utilice delegados para realizar una operación matemática simple (suma, resta, multiplicación, etc.). Define un delegado que pueda apuntar a diferentes métodos que realicen estas operaciones.

```
using System;
public delegate int OperacionMatematica(int a, int b);
class Program
{
  static void Main()
  {
    OperacionMatematica operacion = Suma;
    Console.WriteLine("Suma: " + operacion(5, 3));
    operacion = Resta;
    Console.WriteLine("Resta: " + operacion(5, 3));
  }
  static int Suma(int a, int b)
  {
    return a + b;
  }
```

```
    static int Resta(int a, int b)
    {
        return a - b;
    }
}
```

Ejercicio 2: Eventos

Crea una clase que contenga un evento. Lanza el evento cuando se produce una acción y suscríbete a ese evento en el programa principal.

```
using System;
class MiClase
{
    public event EventHandler MiEvento;
    public void RealizarAccion()
    {
        Console.WriteLine("Realizando alguna acción...");
        OnMiEvento();
    }
    protected virtual void OnMiEvento()
    {
        MiEvento?.Invoke(this, EventArgs.Empty);
    }
}
class Program
{
    static void Main()
    {
        MiClase miObjeto = new MiClase();
        miObjeto.MiEvento += ManejarEvento;
        miObjeto.RealizarAccion();
    }
    static void ManejarEvento(object sender, EventArgs e)
    {
        Console.WriteLine("Evento manejado con éxito.");
    }
}
```

Ejercicio 3: Lambdas

Utiliza lambdas para simplificar la definición de funciones y métodos en tu programa. Por ejemplo, modifica el Ejercicio 1 para utilizar lambdas en lugar de métodos separados.

```
using System;
```

```
public delegate int OperacionMatematica(int a, int b);
class Program
{
  static void Main()
  {
    OperacionMatematica operacion = (a, b) => a + b;
    Console.WriteLine("Suma: " + operacion(5, 3));

    operacion = (a, b) => a - b;
    Console.WriteLine("Resta: " + operacion(5, 3));
  }
}
```

Estos ejercicios te permitirán practicar el uso de delegados, eventos y lambdas en C# dentro del entorno de Visual Studio Code.

Tarea 2

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

Instalación de requisitos previos

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

Creación de una aplicación de consola de .NET Core:

5. Abre Visual Studio Code.
6. Crea una carpeta para tu proyecto (por ejemplo, "ProyectoLinQ").
7. Abre el terminal en Visual Studio Code (puedes hacerlo seleccionando Ver > Terminal en el menú principal).
8. 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

Ejercicio 1: Consulta Simple con LINQ

Crea una lista de números y utiliza LINQ para encontrar los números pares.

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

class Program
{
    static void Main()
    {
        List<int> numeros = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

        var numerosPares = from num in numeros
                           where num % 2 == 0
                           select num;

        Console.WriteLine("Números Pares:");
        foreach (var numero in numerosPares)
        {
            Console.Write(numero + " ");
        }
    }
}
```

Ejercicio 2: Consulta con Proyección

Crea una lista de estudiantes con nombres y edades, y utiliza LINQ para proyectar solo los nombres de aquellos que son mayores de 18 años.

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

class Estudiante
{
    public string Nombre { get; set; }
```

```
    public int Edad { get; set; }
}

class Program
{
    static void Main()
    {
        List<Estudiante> estudiantes = new List<Estudiante>
        {
            new Estudiante { Nombre = "Juan", Edad = 20 },
            new Estudiante { Nombre = "Ana", Edad = 22 },
            new Estudiante { Nombre = "Pedro", Edad = 17 },
            new Estudiante { Nombre = "María", Edad = 25 }
        };

        var nombresMayoresDe18 = from estudiante in estudiantes
                                 where estudiante.Edad > 18
                                 select estudiante.Nombre;

        Console.WriteLine("Nombres de Estudiantes Mayores de 18 años:");
        foreach (var nombre in nombresMayoresDe18)
        {
            Console.Write(nombre + " ");
        }
    }
}
```

Ejercicio 3: Consulta con Ordenamiento y Filtrado

Crea una lista de productos con nombres, precios y categorías, y utiliza LINQ para obtener los productos de una categoría específica, ordenados por precio de forma ascendente.

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

class Producto
{
    public string Nombre { get; set; }
```

```
    public double Precio { get; set; }
    public string Categoria { get; set; }
}

class Program
{
    static void Main()
    {
        List<Producto> productos = new List<Producto>
        {
            new Producto { Nombre = "Laptop", Precio = 1200, Categoria = "Electrónicos" },
            new Producto { Nombre = "Teléfono", Precio = 800, Categoria = "Electrónicos" },
            new Producto { Nombre = "Libro", Precio = 20, Categoria = "Libros" },
            new Producto { Nombre = "Cámara", Precio = 500, Categoria = "Electrónicos" },
            new Producto { Nombre = "Bicicleta", Precio = 300, Categoria = "Deportes" }
        };

        string categoriaElegida = "Electrónicos";

        var productosFiltrados = from producto in productos
                                 where producto.Categoria == categoriaElegida
                                 orderby producto.Precio
                                 select producto;

        Console.WriteLine($"Productos de la categoría '{categoriaElegida}' ordenados por precio:");
        foreach (var producto in productosFiltrados)
        {
            Console.WriteLine($"{producto.Nombre} - ${producto.Precio}");
        }
    }
}
```

Estos ejercicios te ayudarán a practicar el uso de expresiones LINQ y consultas en colecciones utilizando C# en Visual Studio Code.

Tarea 3

Ejercicio 1: Tarea Básica Asíncrona

Crea un programa que ejecute una tarea asíncrona simple. Puedes utilizar el método **Task.Run** para simular una operación asíncrona.

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        Console.WriteLine("Iniciando programa...");
        await MiTareaAsincrona();
        Console.WriteLine("Programa completado.");
    }

    static async Task MiTareaAsincrona()
    {
        await Task.Run(() =>
        {
            Console.WriteLine("Realizando tarea asíncrona...");
            // Simula una operación asíncrona
            Task.Delay(2000).Wait();
        });
    }
}
```

Ejercicio 2: Múltiples Tareas Concurrentes

Crea un programa que ejecute varias tareas de forma concurrente y espere a que todas se completen antes de continuar.

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

```
class Program
{
    static async Task Main()
    {
        Console.WriteLine("Iniciando programa...");

        List<Task> tareas = new List<Task>
        {
            RealizarTareaAsync("Tarea 1"),
            RealizarTareaAsync("Tarea 2"),
            RealizarTareaAsync("Tarea 3")
        };

        await Task.WhenAll(tareas);
        Console.WriteLine("Programa completado.");
    }

    static async Task RealizarTareaAsync(string nombreTarea)
    {
        await Task.Run(() =>
        {
            Console.WriteLine($"Realizando {nombreTarea}...");
            Task.Delay(2000).Wait();
        });
    }
}
```

Ejercicio 3: Manejo de Errores Asíncronos

Crea un programa que maneje errores en operaciones asíncronas utilizando un bloque **try-catch**.

```
using System;
using System.Threading.Tasks;

class Program
{
    static async Task Main()
    {
        try
```

```
        {
            Console.WriteLine("Iniciando programa...");
            await RealizarOperacionAsync();
            Console.WriteLine("Programa completado.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"Error: {ex.Message}");
        }
    }

    static async Task RealizarOperacionAsync()
    {
        await Task.Run(() =>
        {
            Console.WriteLine("Realizando operación asíncrona con error...");
            // Simula una operación que lanza una excepción
            throw new Exception("Algo salió mal.");
        });
    }
}
```

Estos ejercicios te proporcionarán práctica en el uso de programación asíncrona y el manejo de tareas en C# en un entorno de Visual Studio Code.