Debugging and creating asynchronous programs

Step 1: Introduction to Async and Await

Understand the async and await keywords and their role in making programs responsive by allowing tasks to run asynchronously. You'll create a small application using the Visual Studio Code console application you created at the start of the course. Your application will run an asynchronous method.

Remove any existing code in the Program.cs file of your console application and create all the code in each step in that file.

Step 2: Implementing an Asynchronous Method

Create a simple asynchronous method that simulates a time-consuming operation using Task.Delay.

Instructions:

1. In the Program.cs file, create a class called Program.
2. Inside the Program class, create a method called PerformLongOperationAsync.
3. Use await Task.Delay to simulate a delay within the method.

Step 3: Calling the Async Method in Main

Call the asynchronous method from the Main method, ensuring the program waits for its completion before exiting.

Instructions:

1. Below the PerformLongOperationAsync method, create a Main method.
2. In the Main method, call the PerformLongOperationAsync method using Task.Run.
3. Ensure the program waits for the async method to complete.
4. To check your answer, run the Visual Studio Code console application. If you receive an error when you run the code, go to the reading on the next page to compare your code to the correct answer.

Step 4: Simulating Debugging with Console Statements

Use Console.WriteLine statements to simulate breakpoints and observe the flow of the program.

Instructions:

1. Add a Console.WriteLine statement after the await in the PerformLongOperationAsync method.
2. Use these statements to understand the program's flow.
3. To check your answer, run the Visual Studio Code console application. If you receive an error when you run the code, go to the reading on the next page to compare your code to the correct answer.

Step 5: Handling Potential Errors

Add error handling to the async method to make the code more robust.

Instructions:

1. Modify the PerformLongOperationAsync method to include a try-catch block.
2. Catch any exceptions that might occur during execution and print an error message.
3. To check your answer, run the Visual Studio Code console application. If you receive an error when you run the code, go to the reading on the next page to compare your code to the correct answer.

Code:

```
class Program
{
    // Step 2: Asynchronous method simulating a long operation
    static async Task PerformLongOperationAsync()
    {
        try
        {
            Console.WriteLine("Operation started...");
            await Task.Delay(3000); // Simulate delay (3 seconds)
            Console.WriteLine("Operation completed after delay.");
        }
        catch (Exception ex)
        {
            Console.WriteLine($"An error occurred: {ex.Message}");
        }
    }
    // Step 3: Main method - starting point of the app
    static void Main(string[] args)
    {
        // Run the async method and wait for it to complete
        Task task = Task.Run(() => PerformLongOperationAsync());
        task.Wait(); // Ensures the Main thread waits
        Console.WriteLine("Main method completed.");
    }
}
```