1. Define and Invoke a Simple Delegate

Goal: Create a custom delegate type and use it to point to a method.

- **Task:** Define a delegate named LogHandler that takes a string message and returns void.
- **Implementation:** Create a method that prints a message to the console. Instantiate your delegate, point it to that method, and invoke it.

---

2. Implement an Action Timer

Goal: Use the built-in Action delegate to execute code after a delay.

- **Task:** Create a class or method called SimpleTimer.
- **Implementation:** It should take two parameters: an int (seconds) and an Action (the callback). Use Task.Delay or a simple loop to wait for the specified time, then trigger the Action.

Note: Action is a built-in delegate for methods that return void.

---

3. Use a Lambda as an Action

Goal: Simplify your code by passing logic directly without defining a named method.

- **Task:** Create a method ProcessData that accepts an Action<string>.
- **Implementation:** Call ProcessData, but instead of passing a method name, pass a **Lambda expression** (e.g., msg => Console.WriteLine(msg.ToUpper())) that transforms a string to uppercase and prints it.

---

4. Try Action Multicasting

Goal: Chain multiple methods to a single delegate instance.

- **Task:** Create an Action<string> called multiLogger.
- **Implementation:** 1. Assign a method that writes to the Console. 2. Use the += operator to add a second method that simulates "writing to a file" (just another Console print). 3. Invoke the delegate once and observe how both methods execute in sequence.

---

5. Pass in a Func to Validate a Number

Goal: Use the Func<T, TResult> delegate to return a value (in this case, a boolean).

- **Task:** Create a method called FilterNumbers that takes a List<int> and a Func<int, bool> validator.
- **Implementation:** Inside the method, iterate through the list. Use the Func to check if each number meets a criteria (e.g., "Is it even?" or "Is it greater than 10?"). Return a new list containing only the numbers that passed.