**Exercise 1: The Basics of Syntax**

**Goal:** Convert a simple Lambda into a Local Function.

You have the following code using a Lambda:

C#

```
Func<int, int, int> multiply = (a, b) => a * b;
Console.WriteLine(multiply(5, 5));
```

**Task:** Rewrite this logic using a **Local Function** inside a Main method. Bonus points if you can explain why the local function might be slightly more readable in a large method.

---

**Exercise 2: The Recursion Challenge**

**Goal:** Understand why Local Functions handle recursion better than Lambdas.

**Task:** Write a method called CalculateFactorial(int n). Inside it, implement the factorial logic using:

1. A **Local Function**.
2. A **Lambda Expression**.

**Hint:** You'll notice the Lambda requires a "self-reference" trick (assigning it to null first or using a specific delegate type) that the Local Function doesn't need.

---

**Exercise 3: Performance & Variable Capture**

**Goal:** Observe how closures work.

When a Lambda captures a local variable, the compiler creates a class to hold that state (the "heap allocation"). Local functions can often avoid this via ref struct optimizations in newer C# versions.

**Task:** Create a method Counter().

- Inside, define an integer count = 0.
- Create a **Local Function** that increments and prints count.
- Create a **Lambda** that does the same.
- **Question:** If you mark the local function as static, what happens to its ability to access count?

---

**Exercise 4: Ref and Out Parameters**

**Goal:** Explore parameter flexibility.

Lambdas have strict limitations when it comes to ref and out parameters because they are essentially objects.

**Task:**

1. Try to create a Func or Action lambda that takes an out int result parameter.
2. Now, create a **Local Function** that takes an out int result parameter.
3. Observe which one the compiler actually allows.

---

**Exercise 5: The "Define After Use" Test**

**Goal:** Understand scoping and hoisting.

**Task:** In a new method, try to call a Local Function *before* the line where it is defined. Then, try to call a Lambda expression *before* the line where it is defined.

C#

```
void Discovery() {
    // Try calling both here
    // ... logic ...

    // Define both here
}
```

**Question:** Which one allows "hoisting" (calling it before the definition), and why is this useful for keeping the "main" logic of a method at the top?