1. The Classic Factorial

The Goal: Write a function GetFactorial(int n) that returns the product of all positive integers less than or equal to n.

- **Base Case:** If n = 1, return 1.
- **Recursive Step:** n * \text{GetFactorial}(n - 1).
- **Challenge:** How does your code handle an input of 0?

2. Sum of Digits

The Goal: Create a method that takes a multi-digit integer (e.g., 1234) and returns the sum of its digits (1 + 2 + 3 + 4 = 10).

- **Hint:** Use the modulo operator % to get the last digit and integer division / to remove it.
- **Logic:** (n % 10) + SumDigits(n / 10).

3. String Reversal

The Goal: Write a recursive function ReverseString(string str) that flips a string without using any built-in C# Array/LINQ reverse methods.

- **Visualization:**
- **Base Case:** If the string is empty or has a length of 1, return the string.
- **Recursive Step:** Take the last character and append the result of ReverseString on the remaining substring.

4. The Fibonacci Sequence

The Goal: Generate the nth number in the Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, 13 ...).

- **Formula:** F(n) = F(n-1) + F(n-2)
- **Observation:** Notice how many redundant calculations occur. For a bonus, try to think about how you might "save" results to make it faster (memoization).

5. Power Function (x^n)

The Goal: Write a recursive method Power(int baseNum, int exponent) that calculates the result of raising a number to a power without using Math.Pow().

- **Base Case:** Any number raised to the power of 0 is 1 (x^0 = 1).
- **Recursive Step:** Multiply the baseNum by the result of Power(baseNum, exponent - 1).