1. Nullable Types (Safety & Logic)

The goal here is to master the ? syntax and the null-coalescing operators.

1. **The Basic Guard:** Create a method PrintLength(string? input) that prints the length of the string if it's not null, or "String is null" if it is. Use the null-conditional operator ?..
2. **The Default Value:** Write a function that takes a double? representing a temperature. If the value is null, return a default of 20.0. Use the null-coalescing operator ??.
3. **Nullable Logic:** Declare a bool? variable. Write a program that prints "Yes" if true, "No" if false, and "Maybe" if null.
4. **The Assignment Shortcut:** Create a list of strings. Write a line of code that initializes the list only if it is currently null using the null-coalescing assignment operator ??=.
5. **Refactoring Challenge:** Take a nested object structure (e.g., Company -> Department -> Manager -> Name). Write a single line of code to retrieve the Manager's Name safely, ensuring no NullReferenceException occurs if any step in the chain is null.

---

2. Span<T> and Memory Efficiency

Span is all about working with "windows" of data without creating expensive copies on the heap.

1. **The Slice:** Create an array of integers from 1 to 10. Create a Span<int> that points only to the middle four elements (4, 5, 6, 7) and change the 5 to a 50. Observe what happens to the original array.
2. **Stack Allocation:** Use stackalloc to create a Span<int> of size 5 on the stack. Fill it with the squares of its indices and print them.
3. **String Parsing (ReadOnlySpan):** Given a string like "2026-02-26", use ReadOnlySpan<char> and the Slice method to extract the year, month, and day as integers without calling string.Substring().
4. **The Searcher:** Write a function FindIndex(ReadOnlySpan<int> data, int target) that returns the index of the target. Pass it a slice of a larger array to see how it handles offsets.
5. **Performance Comparison:** Write a method that sums every second element in a large array using a loop. Then, write a version using Span<T> and Slice to see if you can achieve the same result with cleaner syntax.

---

3. Bitwise Operators (The Low Level)

Think of these as the "surgical tools" for manipulating individual bits.

1. **Even or Odd:** Write a function that checks if an integer is even or odd using only the bitwise AND (&) operator. (Hint: Look at the least significant bit).
2. **The Toggle:** Given an integer representing a set of flags, use the XOR (^) operator to toggle the 3rd bit (value of 4) from 0 to 1 or 1 to 0.
3. **Bit Packing:** You have two 4-bit numbers (0-15). Pack them into a single 8-bit byte using left-shift (<<) and OR (|). Then, write the code to "unpack" them back into two variables.
4. **Power of Two:** Write a method that returns true if a given integer is a power of two using bitwise logic. (Hint: Compare n and n - 1).
5. **The Mask:** Create a "mask" to extract the last 4 bits of any integer. Use the AND (&) operator with your mask to return only those bits.