**Exercise 1: The Basics – Modeling a 2D Point**

**Objective:** Create a simple struct from scratch and understand value type behavior.

- **Task:** Create a struct named Point2D.
- **Requirements:**
  - Add two public readonly fields: double X and double Y.
  - Create a constructor to initialize these values.
  - Add a method GetDistance() that calculates the distance from the origin (0,0) using the formula: $\sqrt{X^2 + Y^2}$.
- **Bonus:** Try to change a value of a Point2D instance after it's created and see what happens (it should fail if you used readonly).

---

**Exercise 2: Class to Struct – Memory Optimization**

**Objective:** Identify why a class might be better as a struct and perform the conversion.

**The Original Code:**

C#

```
public class ColorRGB
{
    public byte R { get; set; }
    public byte G { get; set; }
    public byte B { get; set; }

    public ColorRGB(byte r, byte g, byte b)
    {
        R = r; G = g; B = b;
    }
}
```

- **Task:** Convert ColorRGB into a struct.
- **Why?** Since a color is just three bytes, allocating it on the heap as a class object adds unnecessary overhead.
- **Requirement:** Make the struct **immutable** (use readonly struct).

---

**Exercise 3: The "Ref" Factor – Working with Large Structs**

**Objective:** Learn how to pass structs efficiently to avoid copying data.

- **Task:** Create a struct called LargeData that contains 8 double fields (e.g., Val1 through Val8).
- **Requirement:** * Write a method CalculateSum(in LargeData data) that returns the sum of all fields.
    - **The Twist:** Use the in keyword in the parameter to ensure the struct is passed by reference without being copied, keeping it read-only.

---

**Exercise 4: Refactoring a "Data Bag"**

**Objective:** Convert a complex class used for mathematical constants into a struct.

**The Original Code:**

```C#
public class SphereMetrics
{
    public double Radius;
    public double PI = 3.14159;

    public double GetVolume() => (4.0/3.0) * PI * Math.Pow(Radius, 3);
}
```

- **Task:** Convert SphereMetrics to a struct.
- **Requirement:** * Initialize Radius through a constructor.
    - Change PI to a const inside the struct.
    - Ensure that if you copy the struct to a new variable and change the radius of the copy, the original remains unchanged.

---

**Exercise 5: Structs with Properties and Logic**

**Objective:** Implementing logic within a value type to handle "Time."

- **Task:** Create a struct called TimeSlot.
- **Requirements:**
    - Fields: int Hours and int Minutes.
    - Add a property TotalMinutes that calculates (Hours * 60) + Minutes.
    - Override the .ToString() method to display the time as HH:MM.
    - **Validation:** Add logic in the constructor to ensure Minutes stays between 0 and 59.