Q?

Question: What is the shortcut to comment and uncomment a selected block of code in Visual Studio?

* Ctrl + K, Ctrl + C & Ctrl + K, Ctrl + U

Question: Explain the difference between a runtime error and a logical error with examples.

* A **runtime error** occurs **while the program is executing**, and it usually causes the program to **crash** or behave unexpectedly. These errors aren't caught at compile time
* Ex:
* int x = 10;
* int y = 0;
* int result = x / y; // Runtime Error: Division by zero
* A **logical error** is trickier: it doesn’t crash the program or throw errors. Instead, your program runs but it gives **incorrect results** because your logic is flawed.
* Ex:
* int radius = 5;
* double area = 2 \* 3.14 \* radius; // Logical Error: Should be π \* r \* r

Question: Why is it important to follow naming conventions such as PascalCase in C#?

* Readability & Professionalism
* Consistency Across Teams
* Tooling & Best Practices
* Maintainability & Debugging

Question: Explain the difference between value types and reference types in terms of memory allocation.

* **Value Types**

**Stored in:**

-The **stack** memory.

**Behavior:**

-When you assign a value type to a variable or pass it to a method, a **copy** of

the value is created.

-Changes made to one variable **do not affect** others.

-> int, double, bool, char, struct, enum

### Reference Types

**Stored in:**

* The actual object is stored in the **heap** memory.
* The **reference** (like an address) is stored in the stack.

**Behavior:**

* When you assign a reference type, you're copying the **reference**, not the object itself.
* Multiple variables can **point to the same object**, and changes via one reference affect the others.

**Common Examples:**

* class, array, string, interface, delegate

Question: What will be the output of the following code? Explain why: int a = 2, b = 7; Console.WriteLine(a % b);

* the output = 2 why: because a < b a %b == a

Question: How does the && (logical AND) operator differ from the & (bitwise AND) operator?

### Logical AND Operator

**Use Case:** Boolean conditions in control statements (if, while, etc.)

**Behavior:**

* Evaluates **left condition first**
* If it's false, it **short-circuits** (skips evaluating the right condition)
* Used for decision-making logic

### & – Bitwise AND Operator

**Use Case:** Performs bit-by-bit comparison of integers.

**Behavior:**

* Evaluates **both sides** always
* Combines each bit of operands using AND logic

Question: Why is explicit casting required when converting a double to an int?

#### 1. ****Data Loss Risk****

* A double can hold decimal places, but an int can only hold whole numbers.
* If C# allowed an implicit cast, it could silently **truncate** values like 89.99 to 89 — causing **unintended behavior** in your code.

#### 2. ****Type Safety in C#****

* C# is a **strongly typed language**, so it forces you to be **explicit** when there’s risk.
* Narrowing conversions (like going from double → int) can change value or precision. C# wants you to **acknowledge that risk**.

#### 3. ****Developer Intent****

* By requiring an explicit cast (int)myDouble, C# ensures:
  + You’ve thought about the conversion.
  + You know whether truncation is acceptable.
  + You’re not accidentally introducing logic bugs or rounding surprises.

Question: What exception might occur if the input is invalid and how can you handle it?

### Exception: FormatException

This happens when the input string passed to int.Parse() **isn’t a valid integer format**. For example:

### How to Handle It

Option 1: Use int.TryParse() for a Safe, Exception-Free Conversion

Option 2: Wrap int.Parse() in a try-catch Block

Question: Given the code below, what is the value of x after execution? Explain why int x = 5; int y = ++x + x++;

* Initial value: x = 5

**Evaluate** ++x: This is the prefix increment

* x is incremented to 6, **then used**
* So ++x evaluates to 6

**Evaluate** x++: This is the postfix increment

* The current value of x is **used first**, then incremented
* x is now 6, so x++ evaluates to 6, then x becomes 7
* **Final values**:
  + x = 7
  + y = 12