Question: Why can't a struct inherit from another struct or class in C#?
Answer: In C#, structs are value types, and value types are designed to be lightweight and stored directly on the stack. Allowing inheritance between structs or from classes would add complexity and overhead that defeats this purpose. Instead, structs can only inherit from .`System.ValueType` (implicitly) and can implement interfaces

---

Question: How do access modifiers impact the scope and visibility of a class member?
Answer: Access modifiers determine who can use a class member. For example, `public` allows access from anywhere, `private` restricts access to within the same class, `protected` allows access from the same class and its subclasses, and `internal` limits access to the same assembly. Choosing the right modifier controls how much of the code is .exposed

---

Question: Why is encapsulation critical in software design?
Answer: Encapsulation hides the internal details of a class and only exposes what’s necessary through public members. This prevents accidental changes to important data, makes the code easier to maintain, and ensures that objects are used in a controlled, .predictable way

---

Question: What is constructors in structs?
Answer: A constructor in a struct is a special method used to initialize its fields when creating a new instance. Structs in C# can have parameterized constructors, but they cannot .(have a parameterless constructor (the compiler provides a default one automatically

---

Question: How does overriding methods like ToString() improve code readability?
Answer: By overriding `ToString()`, you can control how an object is represented as text. This makes it easier to understand the output during debugging, logging, or displaying data .to the user, rather than relying on the default type name

---

Question: How does memory allocation differ for structs and classes in C#?
Answer: Structs are value types, so they are usually stored on the stack and copied when passed around. Classes are reference types, so they are stored on the heap, and variables hold references (pointers) to them. This difference affects performance and how data is .shared or copied between methods