1. What's the difference between compiled and interpreted languages and in this way what about Csharp?

Compiled languages are translated into machine code before the program runs. This translation happens only once using a compiler, and then the result is a standalone executable. These languages usually run faster. Examples include C and C++.

Interpreted languages are executed line by line at runtime by an interpreter. They are usually slower but more flexible and easier to debug. Examples include Python and JavaScript.

C# is considered a hybrid. It is compiled into Intermediate Language (IL) using the C# compiler, and then the .NET runtime uses a Just-In-Time (JIT) compiler to convert IL into machine code at runtime. This gives C# a balance between performance and flexibility.

---

2. Compare between implicit, explicit, Convert and parse casting

- **Implicit Casting**: Happens automatically by the compiler when converting from a smaller type to a larger type, with no risk of data loss. Example: from `int` to `long`.
- **Explicit Casting**: Requires manual conversion when there's a possibility of data loss. You must use casting syntax. Example: from `double` to `int`.
- **Convert**: A method from the `System.Convert` class that converts between different types, such as from string to int. It is safe and throws exceptions if conversion fails.
- **Parse**: A method that converts strings into specific value types like `int.Parse("123")`. It only works on strings and throws an exception if the string is not in a valid format.

Bonus

5. What is meant by "C# is managed code"?

Managed code refers to the code that runs under the control of the .NET Common Language Runtime (CLR). In C#, the CLR manages memory allocation, garbage collection, security, type checking, and exception handling.
This means the programmer doesn't need to manually handle low-level memory management tasks, making development safer and more reliable.

So, when we say **C# is managed code**, we mean that the .NET runtime (CLR) is responsible for managing the execution of the program.

6. What is meant by “struct is considered like class before”?

In earlier versions of programming languages, **structs (structures)** were mainly used to group small sets of related variables — similar to records — and didn’t support advanced features like inheritance or methods.

In C#, **structs are similar to classes in many ways**: they can have constructors, methods, properties, and fields. However, they are different in one major aspect — **structs are value types** while **classes are reference types**.

When we say “**struct is considered like class before**,” we usually mean that structs were originally seen as simpler, limited versions of classes, but in C#, structs have grown more powerful while still behaving differently in terms of memory and performance.