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

Compiled Languages:

- Source code translated entirely into machine code *before* running by a **compiler**.
- **Pros:** Very fast execution, errors caught early (compile-time).
- **Cons:** Less portable (platform-specific), harder to debug runtime logic errors.
- **Examples:** C, C++.

Interpreted Languages:

- Source code read and executed line-by-line *at runtime* by an **interpreter**.
- **Pros:** More portable (same code runs on different platforms), easier to debug (errors reported as they occur), faster development.
- **Cons:** Generally slower execution (translation overhead).
- **Examples:** Python, JavaScript.

C# (Hybrid Approach):

- **First Compilation:** C# code is compiled into an **Intermediate Language (IL)** (not machine code). This IL is platform-independent.
- **Runtime Execution (JIT Compilation):** When the program runs, the **Common Language Runtime (CLR)** uses a **Just-In-Time (JIT)**

compiler to translate the IL into native machine code *on-the-fly* for the specific machine.

- **Benefits:** Achieves both **platform independence** (like interpreted languages) and **near-native performance** (like compiled languages), plus offers runtime optimizations and memory management (Garbage Collection).

2-Compare between implicit, explicit, Convert and parse casting?

- **Implicit Casting:** This is an **automatic** conversion the compiler does for you when there's **no risk of data loss**. Think of it as "widening" a type, like fitting a small number into a larger container (e.g., int to long). It's safe and needs no special syntax.
- **Explicit Casting:** You have to **manually** tell the compiler to do this conversion using (). It's for "narrowing" a type (e.g., long to int), where **data loss is possible**, or for converting between related types where the compiler isn't sure it'll work (e.g., object to string). This is on you, the programmer, to get right, or you'll face errors!
- **Convert Class (e.g., Convert.ToInt32()):** These are **static methods** designed for **general-purpose conversions** between different basic data types, often including converting strings to numbers and vice-versa. They're pretty robust, can handle null inputs gracefully, but will throw exceptions if the data format is wrong or the value is too big.
- **Parse Methods (e.g., int.Parse()):** These are also **static methods**, but they're specific to a data type (like int, double, DateTime). Their main job is to turn a **string into that specific type**. They're

very strict about the string's format and will throw an error if it's not perfect or if the input string is null.

Self:

Garbage collector : Customize >> Self

Customizing a **Garbage Collector (GC)** isn't about building one from scratch, but rather **fine-tuning its existing behavior** to optimize your application's memory usage and performance. This is crucial for balancing speed, responsiveness, and memory footprint.

what meant by Csharp is managed code?

What "Managed Code" Means for C#

When we say C# produces "managed code," it means your code runs under the strict control and supervision of the **Common Language Runtime (CLR)**. The CLR acts as a protective and helpful manager for your application, providing key services:

- **Type Safety:** The CLR enforces strict rules to ensure your code accesses memory correctly and uses data types appropriately, reducing runtime errors.
- **Exception Handling:** Provides a robust system for dealing with errors gracefully.
- **Security:** Offers a layer of security by controlling what your code can access on the system.

- **Language Interoperability:** Allows seamless communication between different .NET languages.
- **Platform Independence:** Because C# compiles to an Intermediate Language (IL) that the CLR then executes, it can run on any platform where a compatible CLR is available.

What is the meant by struct is considered like class before?

- **struct is a Value Type:** Copies its data directly, stored on the stack or inline. Changes to a copy don't affect the original. Cannot inherit.
- **class is a Reference Type:** Stores a reference (pointer) to data on the heap. Copies of the reference still point to the *same* object. Supports inheritance.