C# Assignment Day02

Part02

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

Compiled vs Interpreted Languages

Compiled Languages:

- The source code is **translated into machine code** (binary) **before execution**.
- The compiler produces an **executable file**.
- Execution is usually **faster** because the computer runs native machine code.
- **Examples:** C, C++, Go.

Interpreted Languages:

- The source code is **executed line by line** by an interpreter **at runtime**.
- No separate executable file is produced.
- Usually **slower**, but easier to debug and more flexible.
- **Examples:** Python, JavaScript, Ruby.

Key difference:

- **Compilation = ahead-of-time translation** → machine code.
- **Interpretation = runtime translation** → executed directly.

Where C# fits

C# is a bit **hybrid**:

1. **C# source code (.cs) → compiled** by the **C# compiler (csc)** into **Intermediate Language (IL)**, also called **CIL (Common Intermediate Language)**.
2. The IL is **not directly machine code**; it runs on the **.NET Common Language Runtime (CLR)**.
3. At runtime, the CLR uses a **Just-In-Time (JIT) compiler** to convert IL to **machine code** for the computer.

3- Compare between implicit, explicit, Convert and parse casting.

Implicit Casting (Type Conversion)

- **Definition:** Automatic type conversion by the compiler.
- **Rules:** Works only when **no data will be lost** (safe conversion).
- **Example:** int → double

int a = 10;

double b = a; // implicit cast

Console.WriteLine(b); // 10.0

- **Key points:**
    - No special syntax needed.
    - Safe, always succeeds.
    - Works from smaller type → larger type

Explicit Casting (Type Casting)

- **Definition:** Manual conversion using **cast operator** ()
- **Rules:** Required when **data might be lost** (unsafe conversion).
- **Example:** double → int

double d = 9.78;

int i = (int)d; // explicit cast

Console.WriteLine(i); // 9

- **Key points:**
    - You must specify the target type in parentheses.
    - May truncate or lose data.
    - Can throw runtime errors if incompatible types.

Convert Class

- **Definition:** Provides methods to **convert between different types safely**.
- **Example:** Convert.ToInt32, Convert.ToDouble

string str = "123";

int num = Convert.ToInt32(str);

Console.WriteLine(num); // 123

- **Key points:**
  - Can handle **null and some invalid inputs** more gracefully.
  - Converts **between compatible types**, not just numeric.
  - Throws exceptions for invalid format (FormatException) or overflow (OverflowException).

Parse Method

- **Definition:** Converts a **string** to a specific type using methods like int.Parse, double.Parse.
- **Example:**

string str = "45";

int num = int.Parse(str);

Console.WriteLine(num); // 45

- **Key points:**
  - Only works for strings.
  - Throws FormatException if string is not a valid number.
  - No handling of null (throws ArgumentNullException).