1. Function (Can be implemented)

- In programming, a **function** is a block of code designed to perform a specific task and can be executed when called.
- **Can be implemented** means the function actually contains executable code (a body) that defines what it does when called.
- In C#, a function consists of:
    - **Declaration (signature)** → defines the function name, return type, and parameters.
    - **Implementation (body)** → the code inside { } that runs when the function is called.
- Example:

```
// Declaration + Implementation
int Add(int a, int b)
{
   return a + b;
}
```

- In abstract classes or interfaces (C# 8+), you may see "can be implemented" meaning the method may have a default implementation.

---

2. Upcasting vs Downcasting

Upcasting

- Converting a derived class object into a base class reference.
- **Safe**: No explicit cast required and no data is lost.
- Useful for **polymorphism**.

- Example:

```
class Animal { }

class Dog : Animal { }

Dog dog = new Dog();

Animal animal = dog; // Upcasting
```

Downcasting

- Converting a base class reference into a derived class reference.
- **Unsafe**: Requires explicit cast and can throw InvalidCastException if the object is not actually of the target type.
- Example:

```
Animal animal = new Dog();

Dog dog2 = (Dog)animal; // Downcasting
```

---

3. Reference Type Passing

- **Reference types** (like classes) store references (pointers) to objects in memory.
- **Passing by value (default)**: The reference itself is copied, but both references point to the same object. Modifying the object inside the method changes the original.
- **Passing by reference (ref)**: The method can change the reference itself to point to a new object.
- Example:

```
class Person { public string Name; }

void ChangeName(Person p)

{

   p.Name = "Ahmed"; // changes the same object

}

void ReplaceObject(ref Person p)

{    p = new Person { Name = "Ali" }; // changes the reference}
```

✅ **Summary Table:**

Concept	Description
Function (can be implemented)	A function with an actual body containing executable code
Upcasting	Derived → Base, safe, implicit
Downcasting	Base → Derived, unsafe, explicit
Reference type passing	Passing object references, can modify original object