1. Function

A **function** is a block of code that performs a specific task when called. It helps organize code, reduce repetition, and make programs easier to maintain.
A function can be implemented in various ways depending on the programming language.

Key Points:

- **Definition**: A set of instructions grouped together under a name.
- **Syntax Example (C#)**:

```
int Add(int a, int b)

{

   return a + b;

}
```

- **Benefits**:
    1. **Reusability** – Write once, use multiple times.
    2. **Readability** – Code becomes easier to understand.
    3. **Modularity** – Each function handles one responsibility.

Types of Functions:

1. **Built-in functions** – Already provided by the language (e.g., Console.WriteLine() in C#).
2. **User-defined functions** – Written by the programmer for specific needs.

---

2. Upcasting vs Downcasting

In **object-oriented programming**, casting is the process of converting one type into another.
When working with classes and inheritance, **upcasting** and **downcasting** refer to converting between base and derived class references.

Upcasting:

- **Definition**: Converting a derived class reference into a base class reference.
- **Safe** – No data loss for accessible members; happens automatically in most languages.
- **Example**:

```
class Animal { }

class Dog : Animal { }

Dog d = new Dog();

Animal a = d; // Upcasting
```

Downcasting:

- **Definition**: Converting a base class reference back into a derived class reference.
- **Risky** – Can cause runtime errors if the object is not actually of the derived type.
- **Example**:

```
Animal a = new Dog();

Dog d = (Dog)a; // Downcasting
```

Summary Table:

Feature	**Upcasting**	**Downcasting**
Direction	Derived → Base	Base → Derived
Safety	Safe	Risky
Type Check	Implicit or explicit	Always explicit
Common Use	Polymorphism	Access specific derived features

3. Ref Type Passing

In C#, **ref** is a keyword that allows passing variables **by reference** instead of by value.

How It Works:

- Normally, when a variable is passed to a method, a **copy** of its value is sent.
- Using ref, the method works directly on the original variable, meaning changes inside the method affect the original value.

Rules:

1. The variable must be **initialized** before passing.
2. Both the method definition and the method call must include the ref keyword.

Example:

```
void Increase(ref int number)
{
   number += 10;
}
int myNum = 5;
Increase(ref myNum);
// myNum is now 15
```

When to Use:

- When you want the method to modify the caller's variable.
- For performance optimization when passing large data structures (avoids copying).