Functions

Functions:

In Java programming, suppose you are writing a program that requires multiple uses of addition (sum) logic. Instead of writing the sum logic repeatedly, you can create a function that performs the addition and accepts variables as parameters. This concept of functions allows us to encapsulate the sum logic into a reusable function.

Now, whenever we need to perform addition, we can simply call the function and pass the required variables as arguments, eliminating the need to rewrite the addition logic.

Uses of Function

- Code Reusability: Functions allow you to write a piece of code once and use it multiple times throughout your program. This not only saves time and effort but also improves the maintainability of your code. If you need to make a change or fix a bug, you only have to do it in one place (the function definition) rather than searching through the entire program.

- Code Organization: Functions help in structuring your code logically. By separating different tasks into functions, you can improve the readability and maintainability of your codebase. Functions provide a natural way to divide your code into meaningful blocks and make it easier to navigate and understand.

How to define a Function?

In Java, functions are defined using the keyword "public" or "private", followed by the return type, the function name, and any parameters or arguments that it takes. For example, a function that adds two numbers might look like this:

Syntax:

```
public int add(int num1, int num2) {
    int sum = num1 + num2;
    return sum;
}
```

Pass by value in java

- For primitive data types (e.g., int, float, boolean), Java uses pass by value.
- When you pass a primitive to a method, you're passing a copy of the actual value.
- Changes made to the parameter within the method do not affect the original value outside the method.

```java
public void modifyValue(int num) {
    num = 42; // This only modifies the local copy of 'num'
}

public static void main(String[] args) {
    int x = 10;
    modifyValue(x);
    System.out.println(x); // Output: 10 (unchanged)
}
```