

Functions & Pass by Value/Ref

Write once, use anywhere. The power of reuse. ■

■ What You Will Learn

- What a function is and why it exists
- Return types, parameters, and void
- Pass by Value — Java's default behaviour
- How objects behave differently (Pass by Reference)
- Method overloading — same name, different inputs

■ Intuition First

A function is a **named recipe**. Instead of writing the recipe every time you want to cook, you just call its name. *makePizza()*. Done. Functions save you from repeating yourself and make your code clean.

■ Think Like This

A TV remote has buttons. Each button (function) does one specific job. You press 'Volume Up' — you don't need to know how the TV increases volume internally. That's the power of functions: call and forget the details.

■ Visual: Anatomy of a Function

```
returnType functionName (parameters)
■■■■■ ■■■■■■■■■■ ■■■■■■■■■■
■ ■ ■
int add (int a, int b)
(what it (label/name) (inputs it needs)
gives back)
CALLING IT:
int result = add(3, 7);
■ ■
a b ← values passed in
■
returns 10 ← stored in result
```

■■ Pass by Value — Visual Proof

```
int x = 10;
changeMe(x); // x is COPIED into function
println(x); // still prints 10!
void changeMe(int n) {
  n = 99; // only the COPY changes
} // original x is untouched
MEMORY VIEW:
main: [ x = 10 ] ← original, never touched
function:[ n = 10 ] → [ n = 99 ] ← only copy changed
```

■ Code Implementation

```
public class Functions {
  // Function that RETURNS a value
  static int add(int a, int b) {
    return a + b;
  }
  // Function that returns NOTHING (void)
  static void greet(String name) {
    System.out.println("Hello, " + name + "!");
  }
  // Overloaded: same name, different params
  static double add(double a, double b) {
    return a + b;
  }
  public static void main(String[] args) {
    int sum = add(3, 7); // calls int version
    System.out.println(sum); // 10
    greet("Rahul"); // Hello, Rahul!
    double s2 = add(1.5, 2.5); // calls double version
  }
}
```

■ Time & Space Complexity

■ TIME COMPLEXITY

Depends

■ SPACE COMPLEXITY

O(1) per call

A function's complexity depends on what's inside it. The function call itself is O(1). Stack space is O(n) for recursive calls (next topic!).

■ Common Mistakes

■ Forgetting return

If your function has a return type (like int), you **MUST** return something on all code paths. Java will refuse to compile otherwise.

■ Expecting pass-by-reference for primitives

Java **ALWAYS** copies primitives. If you pass an int and modify it inside, the original doesn't change. Use arrays or objects if you need to modify from inside.

■ Practice Problems

#	Problem	Difficulty
1	Write a function to check if a number is even or odd	Easy
2	Write functions for area of circle, rectangle, triangle	Easy
3	Write an overloaded max() for 2 and 3 numbers	Easy
4	Write a function that returns factorial of n	Medium
5	Write a function isPalindrome(String s) returning boolean	Medium

■ Quick Revision Box

- ✓ Function = named block of reusable code
- ✓ void = function returns nothing
- ✓ return type = what the function gives back
- ✓ Parameters = input variables inside ()
- ✓ Java passes COPIES of primitives (pass by value)
- ✓ Arrays/Objects — the reference is copied (can modify contents)
- ✓ Overloading = same name, different parameter list

■ Visual Thinking

■ Imagine It Like This

Think of functions as vending machine slots. You insert input (coins/params), the machine does its thing, and out comes the result. Pass by value is like photocopying your ID — the machine keeps the copy, your original ID stays with you. Animation: show a tube receiving input and ejecting output.