

# Recursion Basics

The function that calls itself. Mind = blown. ■

## ■ What You Will Learn

- What recursion is and why it's useful
- Base case — the stopping condition (CRITICAL!)
- Recursive case — breaking the problem down
- How the call stack works during recursion
- Factorial and Fibonacci as classic examples

## ■ Intuition First

Recursion is when a function **calls itself** with a smaller version of the problem. It's like standing between two mirrors — you see reflections of reflections... but it must stop somewhere. That's the **base case**.

### ■ Think Like This

Imagine Russian nesting dolls (Matryoshka). To open the outermost, you open the next one inside, and the next... until you hit the smallest doll that doesn't open. That smallest doll is your base case.

## ■ Call Stack Visualized — factorial(4)

```
factorial(4)
→ 4 * factorial(3)
→ 3 * factorial(2)
→ 2 * factorial(1)
→ 1 * factorial(0)
→ BASE CASE: return 1
← returns 1*1 = 1
← returns 2*1 = 2
← returns 3*2 = 6
← returns 4*6 = 24
Stack grows DOWN as calls are made →
Stack unwinds UP as calls return ←
```

## ■ Two Parts Every Recursive Function MUST Have



### ■ Missing or wrong base case → `StackOverflowError`

If your base case is never reached, the function calls itself forever until Java crashes with `StackOverflowError`. Always verify your base case is reachable.

### ■ Not making the problem smaller

If recursive(n) calls recursive(n) (same input), you'll loop forever. Always ensure each call reduces the problem: recursive(n-1) or recursive(n/2).

## ■ Pro Tips

### ■ Pro Tip — Print and trace

Add `System.out.println("Calling with n=" + n);` at the start of your function. Watch the calls happen. This is the best way to understand recursion.

## ■ Practice Problems

| # | Problem                                            | Difficulty |
|---|----------------------------------------------------|------------|
| 1 | Write recursive sum of digits of a number          | Easy       |
| 2 | Write recursive reverse of a string                | Easy       |
| 3 | Write recursive power: <code>pow(base, exp)</code> | Easy       |
| 4 | Count occurrences of a char in string recursively  | Medium     |
| 5 | Tower of Hanoi (3 pegs, n discs)                   | Medium     |

## ■ Quick Revision Box

- ✓ Recursion = function calling itself
- ✓ MUST have a base case (stopping condition)
- ✓ Recursive case reduces the problem size
- ✓ Each call creates a new stack frame
- ✓ Stack unwinds (returns) in reverse order
- ✓ Factorial:  $O(n)$  time,  $O(n)$  space
- ✓ Naïve Fibonacci:  $O(2^n)$  — very slow for large n

## ■ Visual Thinking

### ■ Imagine It Like This

Think of recursion as a stack of plates. Each function call adds a plate on top. When a base case is hit, you start removing plates from the top (returning results). Animation: Show plates stacking up with each recursive call, then popping off one by one as returns bubble back up.