🧠 What is Recursion?

> Recursion is like a **loop made by a function calling itself**. It's a powerful problem-solving tool where a complex problem is broken into smaller, simpler sub-problems until it reaches a "base case" (a simple, solvable problem).

Example: Imagine opening Russian dolls inside each other. You keep opening one until you find the smallest doll (base case). Recursion does the same thing with data or tasks!

🔄 How Does Recursion Work?

Recursion has two essential parts:

1. **Base Case**: The simplest form of the problem, which stops the recursion.
2. **Recursive Case**: The function calls itself with a smaller/transformed version of the problem.

Think of it like climbing stairs:

- Each step is a sub-problem.
- You stop when you reach the top (base case).

Key Rule: Every recursive function must have a base case. Without it, the function will call itself infinitely (like an infinite loop)!

🧱 Example 1: Calculating Factorial

Factorial (n!) means multiplying all numbers from 1 to *n* (e.g., 5! = 5×4×3×2×1).

Java Code:

```
int factorial(int n) {
    if (n == 0) { // Base case: 0! = 1
        return 1;
    } else {      // Recursive case: n! = n × (n-1)!
        return n * factorial(n - 1);
    }
}
```

How It Works:

1. **Call factorial(3):**
   - Checks if 3 == 0? No → calls `factorial(2)`.
2. **Call factorial(2):**
   - Calls `factorial(1)`.
3. **Call factorial(1):**
   - Calls `factorial(0)`.
4. **Base case hit!** returns 1.
5. Now multiply back: 1×1=1 → 2×1=2 → 3×2=6.

Self-Check Questions:

- What would happen if the base case (`n == 0`) was removed?
- How does the function "remember" previous calculations?

🧱 Example 2: Sum of Numbers

Problem: Add all numbers from 1 to *n*.

Java Code:

```
int sum(int n) {
    if (n == 1) { // Base case: Sum of 1 is 1
        return 1;
    } else {      // Recursive case: n + sum(n-1)
        return n + sum(n - 1);
    }
}
```

How It Works:

- **sum(4)** → 4 + sum(3) → 4 + (3 + sum(2)) → ... → 1 + 1 (base case).

Self-Check Questions:

- What's the base case for sum(n)?
- How does recursion reduce the problem size each time?

⚠️ Common Mistakes & Tips

- **Forgetting the base case:** Always define when to stop!
- **Incorrect recursion:** Ensure the problem size reduces each call (e.g., `n-1`, not `n`).
- **Stack Overflow:** Deep recursion can exhaust memory. Use iteration for very large inputs.

Tips:

- Visualize recursion like a tree: Each call splits into new branches.
- Practice with small inputs first (e.g., factorial(3) instead of factorial(100)).

🌟 Real-Life Analogies

- **Directory Search:** Traversing folders recursively (e.g., a folder inside a folder).
- **Mirror Reflection:** A mirror reflecting images infinitely (though recursion stops at the base case!).
- **Cooking Recipe:** Follow steps recursively (e.g., making dough requires flour, which requires grinding grains, etc.).

🎯 Quick Recap

- Recursion is a function calling *itself* to solve a problem.
- Needs a **base case** to avoid infinite loops.
- Common use cases: factorial, sum, tree traversal.
- Java example: `factorial(n)` or `sum(n)`.

Emoji Checklist:

- 🎯 Base case is your safety net!
- 🤔 Did you understand the call stack?
- 🚀 Practice with small examples.

Final Thought: "Recursion is like teaching a child to ride a bike by riding it with them until they confidently do it alone!" 🚲✨