Recursion – Introduction to Recursion

1. Introduction

Recursion is a programming technique where a **function calls itself** to solve a problem.

It is mainly used to solve problems that can be **broken down into smaller sub-problems of the same type**.

Recursion is widely used in **DSA, algorithms, and problem solving**, especially for problems involving repetition and hierarchical structures.

---

2. What is Recursion?

Recursion occurs when:

- A function calls itself directly or indirectly
- Each call solves a smaller version of the original problem
- The process stops when a **base condition** is reached

Without a base condition, recursion leads to **infinite calls**.

---

3. Why Do We Use Recursion?

Recursion is used because:

- It simplifies complex problems
- Makes code shorter and more readable
- Naturally fits problems like trees and graphs
- Helps understand divide-and-conquer algorithms
- Commonly used in algorithm design

---

4. Key Components of Recursion

Every recursive solution must have **two essential parts**:

1️⃣ Base Case

- The condition where recursion stops
- Prevents infinite function calls

2️⃣ Recursive Case

- The part where the function calls itself
- Breaks the problem into smaller sub-problems

---

5. Basic Idea Behind Recursion

The idea is:

- Solve a big problem by solving smaller versions of the same problem
- Each recursive call reduces the problem size
- When the base case is reached, results are returned back

---

6. Logic of Recursion (Plain English)

1. Check if the base condition is satisfied
2. If yes, return the result
3. If not, call the same function with a smaller input
4. Repeat until base condition is met
5. Combine results while returning back

---

7. Example Explanation (Conceptual)

To print numbers from 1 to 5 using recursion:

- Print 1

- Ask recursion to print remaining numbers
- Stop when number becomes greater than 5

This avoids using loops and uses function calls instead.

---

8. How Recursion Works Internally

- Each function call is stored in the **call stack**
- New calls are added on top of the stack
- Once base case is reached, calls start returning
- Stack unwinds until the first call finishes

---

9. Types of Recursion

1️⃣ Direct Recursion

A function calls itself directly.

2️⃣ Indirect Recursion

A function calls another function, which in turn calls the first function.

---

10. Advantages of Recursion

- Clean and elegant code
- Easier to solve complex problems
- Reduces need for loops
- Best suited for tree and graph problems

---

11. Limitations of Recursion

- Uses extra memory (call stack)
- Can cause stack overflow
- Slower compared to loops in some cases

- Harder to debug for beginners

---

12. When to Use Recursion?

Use recursion when:

- Problem has repetitive structure
- Solution can be defined in terms of itself
- Input size reduces at every step
- Problem involves hierarchical data

Avoid recursion when:

- Performance is critical
- Iterative solution is simpler

---

13. Real-World Applications

- Tree traversal
- Graph traversal
- Factorial calculation
- Fibonacci series
- Divide and conquer algorithms (Merge Sort, Quick Sort)

---

14. Time and Space Considerations

- Time complexity depends on number of recursive calls
- Space complexity depends on recursion depth
- Deep recursion may cause stack overflow

---

15. Summary

- Recursion is a function calling itself

- Requires base case and recursive case
- Simplifies complex problems
- Uses call stack memory
- Important concept in DSA
