Recursion Notes

- Function calls itself directly or indirectly.
- Helps in solving bigger/complex problems in a simple way.
- Solve a problem by breaking it down into smaller subproblems.

Indirect Recursion: A function calls another function, which eventually calls the
original function.

- Space complexity is not constant because of recursive calls.
- Can convert a recursive solution into iteration and vice versa.

Tail Recursion: A type of recursion where the recursive call is the last operation
performed before returning the result, making it more optimised for compilers
that support tail call optimisation (TCO).

How to Understand and Approach a Problem
1. Identify if you can break down the problem into smaller subproblems.
2. Prepare a recursion tree and study it.
3. Identify base cases to stop recursion.
4. Ensure overlapping subproblems do not lead to redundant calculations.
5. Think about time and space complexity.

Stack Behavior:
- Fun(n--) passes the value of n first and then subtracts (can cause stack overflow).
- Fun(--n) works, subtracts first, and then passes the value.

Advantages of Recursion
- Provides a clean and simple way to solve problems like tree traversal, backtracking,
  and divide & conquer algorithms.
- Reduces code complexity for problems that can naturally be divided into subproblems.

Disadvantages of Recursion
- High memory usage due to function call stack.
- Can be slower due to repeated function calls compared to an iterative approach.
- Might lead to stack overflow if the base case is not defined properly.

When to Use Recursion
- When the problem exhibits overlapping subproblems and optimal substructure.
- When an iterative solution is less intuitive (e.g., tree and graph traversal,
  dynamic programming, backtracking problems like N-Queens, Sudoku solver, etc.).