# What is Recursion?

**Recursion** is a programming technique where a function solves a problem by calling itself with a smaller or simpler input. It's especially powerful when:

* The problem can be broken down into **smaller, similar subproblems**.
* The overall solution depends on **combining solutions of these subproblems**.

Recursion simplifies complex logic by expressing a problem in terms of itself. It is particularly useful for:

* **Divide-and-conquer** problems.
* **Tree or graph traversals**.
* **Dynamic programming** and **mathematical modelling**.

# Example: Financial Forecasting

In financial forecasting, you may want to predict the future value of an investment based on historical data and a consistent growth rate.

Instead of using loops, recursion can:

* Model the compounding process naturally.
* Emphasize how future values depend on previous ones.
* Simplify the code for clearer logic and maintenance.

# Time and Space Complexity

* **Time Complexity**: O(n)  
  One recursive call per period (e.g., per year or quarter).
* **Space Complexity**: O(n)  
  Due to the recursive call stack (one frame per level of recursion).

# Optimizing Recursive Solutions

**Memorisation**

If your recursive logic involves **repeated calculations of overlapping subproblems** (like in Fibonacci numbers or subset sum problems), **memorization** helps by storing intermediate results.

This:

* Avoids recomputation.
* Reduces time complexity significantly.
* Converts exponential time solutions into polynomial time in many cases.