**Recursion**

When a method calls itself again and again to solve a problem it is known as recursion. It is used to solve complex problems by breaking them down into simpler subproblems which are easy to tackle.

**Why Use Recursion?**

* Simplifies problems that have repeating patterns, like:
  + Financial growth (compounding)
  + Tree traversal
  + Factorials, Fibonacci numbers
* Often more readable than iterative loops for such cases

**My Solution**

Let’s assume a compound growth model, where each year the value grows by a certain growth rate:

futureValue = presentValue × (1 + growthRate)^n

We'll implement this recursively.

**Explanation:**

* Each recursive call reduces years by 1.
* When years == 0, the recursion stops and returns the base presentValue.
* Recursively multiplies by (1 + growthRate) at each step.
* This calculates:  
  FV = PV × (1 + r)^years

**Time and Space Complexity**

The time and space complexity of my solution is O(n)

One big problem of recursion is that it requires a large amount of space to store all the function call and temporary variable created for each call.

**Optimization:**

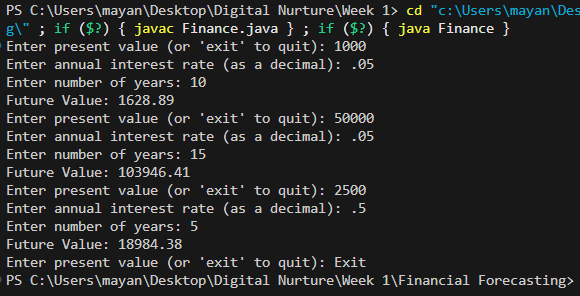
Recursive calls can lead to:

* **Stack overflow** for large n
* **Repeated computation** (in more complex recursion, e.g., Fibonacci)

**Optimized Solution (Iterative or Tail Recursion)**

* Iterative Approach (better for production use)
* Use Memoization or Iteration for complex cases with overlapping subproblems (like predicting with historical patterns, trends, or cycles).

**Sample Output**

****