**Understanding Recursion**

**Recursion** is a programming technique where a function calls itself to solve smaller instances of a problem. It's particularly useful for problems that can be broken down into **smaller, repetitive subproblems**, such as in financial forecasting, where future values often depend on the previous year’s data.

Recursion helps write **cleaner and more intuitive code** for such scenarios, especially when there’s a mathematical relationship, like compound interest or growth patterns.

However, recursion must be used carefully, as it can lead to **stack overflow** or **performance issues** if not optimized. This is where techniques like **memoization** or **converting to iterative solutions** come in handy.

ANALYSIS

Time Complexity of Recursion

The time complexity of this recursive algorithm is **O(n)**, where *n* is the number of years into the future. This is because the function makes one recursive call for each year until it reaches the base case (years == 0). Although this is acceptable for small inputs, for larger values of *n*, recursion can lead to **deep call stacks** and **repeated calculations** if subproblems overlap. This is inefficient and can be optimized.

To avoid excessive computation, we can:

* **Use memoization** to store previously computed values and reuse them.
* **Convert the recursive approach to an iterative one**, which reduces the function call overhead and improves performance.
* In financial forecasting, since the formula is exponential (P \* (1 + r)^n), it's more efficient to use Java’s built-in Math.pow() function instead of recursive multiplication.