📅 Day 20: Learning Summary

Aptitude: Number Series

Number series require identifying the mathematical rule governing a sequence of numbers.

Common Patterns to Check

1. **Arithmetic (Difference):** Constant difference or a difference that increases/decreases (e.g., $+2, +4, +6 \dots$).
2. **Geometric (Ratio):** Numbers are multiplied or divided by a constant (e.g., $\times 2, \times 3 \dots$).
3. **Square/Cube Series:** Numbers are squares or cubes (n^2 or n^3), or variations like $(n^2 + 1)$.
4. **Alternating Series:** Two different patterns running in parallel at odd and even positions.
5. **Fibonacci-like:** The next term is the sum of previous terms.
6. **Mixed Operations:** Combinations like $(x \times 2 + 1)$.

Programming: Rotate an Array by K Positions

The goal is to shift the elements of an array to the right by k steps.

Approach 1: Using an Extra Array

- Create a new array and place each element at its new position: `new_arr[(i + k) % n] = arr[i]`.
- **Space:** $O(n)$.

Approach 2: Slicing (Python Specific)

- `arr = arr[-k:] + arr[:-k]`
- Very concise but uses extra space under the hood.

Approach 3: Reversal Algorithm (Most Efficient)

This method allows rotation in-place without extra memory.

1. **Normalize K:** $k = k \pmod n$ (to handle cases where $k >$ array length).
2. **Reverse the whole array.**
3. **Reverse the first k elements.**
4. **Reverse the remaining $n - k$ elements.**

- **Time:** $O(n)$.
- **Space:** $O(1)$.

Concept: Python Decorators

Decorators are a powerful tool that allows you to wrap another function to extend its behavior without permanently modifying it.

The "@" Syntax

A decorator is essentially a function that takes another function as an argument and returns a new function.

```
def my_decorator(func):
    def wrapper():
        print("Something before the function.")
        func()
        print("Something after the function.")
    return wrapper

@my_decorator
def say_hello():
    print("Hello!")
```

Use Cases: Logging, access control (authentication), timing functions for performance, and caching.

C++ Concept: Macros

Macros are pre-processor directives that perform text substitution before the actual compilation begins.

Characteristics

- **Keyword:** Defined using `#define` .
- **No Type Checking:** Since it is just text replacement, the compiler doesn't check types, which can lead to bugs.
- **Speed:** They are slightly faster than functions for very small tasks because there is no function call overhead.
- **Risk:** Always use parentheses in macros to avoid operator precedence issues.
    - *Bad:* `#define SQUARE(x) x * x` (SQUARE(1+1) becomes $1 + 1 \times 1 + 1 = 3$)
    - *Good:* `#define SQUARE(x) ((x) * (x))`

SQL: LEFT & RIGHT JOIN

Unlike `INNER JOIN` , Outer Joins (Left and Right) preserve records even when there is no match in the other table.

1. LEFT (OUTER) JOIN

Returns **all** records from the left table and the matched records from the right table.

- If there is no match, the result is `NULL` from the right side.
- *Use Case:* Finding all customers and seeing what orders they placed (including those who haven't ordered yet).

2. RIGHT (OUTER) JOIN

Returns **all** records from the right table and the matched records from the left table.

- If there is no match, the result is `NULL` from the left side.
- *Note:* `A RIGHT JOIN B` is functionally identical to `B LEFT JOIN A` .

Visual Comparison

- **INNER JOIN:** Only the middle (Intersection).
- **LEFT JOIN:** The whole Left circle + intersection.
- **RIGHT JOIN:** The whole Right circle + intersection.