Day 22: Logical Reasoning, Stacks, & Optimization

1. Logical Reasoning: Assumptions & Conclusions

Focusing on structured thinking helps in identifying the underlying premise of an argument.

- **Statement & Assumption:** An assumption is something taken for granted or "hidden" within the statement.
  - *Check:* If the assumption is false, does the statement still make sense? If not, the assumption is implicit.
- **Statement & Conclusion:** A conclusion is a logical derivation that must be 100% true based solely on the given statement.
  - *Check:* Avoid using external information; only use what is explicitly stated.

2. Programming: Validate Parentheses (Stack)

The **Last-In, First-Out (LIFO)** property of a stack is perfect for this problem because the last opening bracket must be the first one to be closed.

```
def is_valid(s: str) -> bool:
    stack = []
    # Mapping for easy lookup
    mapping = {")": "(", "}": "{", "]": "["}

    for char in s:
        if char in mapping:
            # Pop the top element if stack isn't empty, else assign a dummy value
            top_element = stack.pop() if stack else '#'

            # Check if the opening bracket matches the expected mapping
            if mapping[char] != top_element:
                return False
        else:
            # It is an opening bracket, push to stack
            stack.append(char)

    # If the stack is empty, all brackets were matched correctly
    return not stack

# Example usage:
# print(is_valid("()[]{}")) -> True
# print(is_valid("([)]"))   -> False
```

3. Python Concept: List Comprehensions

List comprehensions provide a concise way to create lists. They are generally faster than manual `for` loops because they are optimized at the C level in CPython.

Syntax: `new_list = [expression for item in iterable if condition == True]`

Examples:

- **Basic:** `squares = [x**2 for x in range(10)]`
- **With Condition:** `evens = [x for x in range(20) if x % 2 == 0]`
- **Nested (Flattening):** `flat = [num for sublist in matrix for num in sublist]`

4. C/C++ Concept: Loop Optimization

Optimization aims to reduce the "cost" of the loop in terms of CPU cycles.

- **Loop Unrolling:** Reducing the number of iterations by doing more work inside each iteration to minimize jump instructions.
- **Loop Invariant Code Motion (Hoisting):** Moving calculations that don't change inside the loop to the outside.
  - *Inefficient:* `for(int i=0; i < strlen(str); i++)` (calculates length every time).
  - *Optimized:* `int n = strlen(str); for(int i=0; i < n; i++)` .
- **Strength Reduction:** Replacing expensive operations (like `*` or `/`) with cheaper ones (like `+` or bit-shifting).

5. SQL: Subqueries

Subqueries (inner queries) allow you to perform complex filtering based on the results of another query.

- **Scalar Subquery:** Returns a single value.

```
SELECT name, salary
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

- **IN / NOT IN Subquery:** Checks against a list of values.

```
SELECT name
FROM students
WHERE department_id IN (SELECT id FROM departments WHERE location = 'New York
```

- **Correlated Subquery:** The inner query depends on the outer query for its values.

```
SELECT e1.name, e1.salary
FROM employees e1
WHERE e1.salary > (SELECT AVG(e2.salary)
                   FROM employees e2
                   WHERE e2.department = e1.department);
```