Day 26 Learning Report

1. Aptitude: Statement & Assumptions

In logical reasoning, an **assumption** is something "taken for granted." It is the hidden link between the premise and the conclusion.

Key Rules for Validity:

- **Implicit vs. Explicit:** The assumption must be internal to the statement. If it requires external information not implied by the text, it is invalid.
- **Logical Connection:** The conclusion must fall apart if the assumption is proven false.
- **Avoid Generalizations:** Assumptions containing words like "all," "each," "every," or "only" are often (but not always) invalid unless the statement is equally absolute.
- **Positive Intent:** Usually, if a notice or advertisement is given, the assumption is that people will read and respond to it.

2. Programming: Find All Subarrays with Sum K

Finding subarrays with a target sum is a classic interview problem. While a nested loop (O(n^2)) works, the **Prefix Sum + Hash Map** approach is the optimal O(n) solution.

Logic (Prefix Sum Technique):

1. Maintain a `current_sum` as you iterate through the array.
2. Store the frequency of each `current_sum` in a hash map.
3. For every element, check if `current_sum - K` exists in the map.
4. If it exists, it means the subarray between the previous occurrence and the current index sums up to K.

Example (Python):

```
def find_subarrays(arr, k):
    count = 0
    curr_sum = 0
    prefix_sums = {0: 1} # Base case: sum of 0 seen once

    for x in arr:
        curr_sum += x
        if (curr_sum - k) in prefix_sums:
            count += prefix_sums[curr_sum - k]
        prefix_sums[curr_sum] = prefix_sums.get(curr_sum, 0) + 1
    return count
```

3. Python Virtual Environments (`venv`)

A virtual environment is a self-contained directory tree that contains a Python installation for a particular version of Python, plus a number of additional packages.

Why use them?

- **Dependency Isolation:** Project A might need Django 3.0, while Project B needs Django 4.0. Virtual environments prevent version conflicts.
- **Clean Global Space:** Keeps your system Python clean from hundreds of small packages.
- **Reproducibility:** You can generate a `requirements.txt` file specifically for that project.

Core Commands:

- Create: `python -m venv myenv`
- Activate (Windows): `myenv\Scripts\activate`
- Activate (Mac/Linux): `source myenv/bin/activate`

4. C/C++ Concept: The Build Process

The transition from source code to an executable file involves four distinct stages:

1. **Preprocessing:** Handles directives starting with `#`. It expands macros, includes header files (`#include`), and strips comments.
2. **Compilation:** The preprocessed code is translated into **Assembly code** specific to the target processor architecture.
3. **Assembly:** The assembler converts assembly code into **Object code** (machine code in `.obj` or `.o` files).
4. **Linking:** The linker combines multiple object files and library files into a single **Executable**. It resolves function calls and global variables across files.

5. SQL: INDEX

An index is a data structure (usually a B-Tree) that improves the speed of data retrieval operations on a database table.

Key Concepts:

- **Trade-off:** While `SELECT` queries become faster, `INSERT`, `UPDATE`, and `DELETE` become slower because the index must also be updated.
- **Clustered Index:** Determines the physical order of data in a table (usually the Primary Key). Only one per table.
- **Non-Clustered Index:** A separate structure from the data rows. It contains pointers to the data locations.

Syntax:

```
-- Creating an index on the 'email' column
CREATE INDEX idx_user_email ON Users(email);
```

When to use: Use indexes on columns frequently used in `WHERE` clauses, `JOIN` conditions, or `ORDER BY` statements.