

## Day 6: Study Notes & Recap

### 1. Aptitude: Profit & Loss

**Goal:** Understand business calculations involving Cost Price (CP), Selling Price (SP), and margins.

#### Key Concepts & Formulas

- **Cost Price (CP):** The amount paid to purchase an article.
- **Selling Price (SP):** The price at which an article is sold.
- **Profit (Gain):** When  $SP > CP$ .

$$\bullet \quad \text{Profit} = SP - CP$$

- **Loss:** When  $CP > SP$ .

$$\bullet \quad \text{Loss} = CP - SP$$

#### Percentage Calculations

**Important Rule:** Profit% and Loss% are always calculated on **Cost Price** (unless specified otherwise).

- **Profit %:**

$$\frac{\text{Profit}}{CP} \times 100$$

- **Loss %:**

$$\frac{\text{Loss}}{CP} \times 100$$

#### Quick Formulas for SP and CP

- **Finding SP when Profit% is given:**

$$SP = \left( \frac{100 + \text{Profit}\%}{100} \right) \times CP$$

- **Finding CP when Profit% is given:**

$$CP = \left( \frac{100}{100 + \text{Profit}\%} \right) \times SP$$

## 2. Programming: Character Frequency in a String

**Problem Statement:** Count how many times each character appears in a string. **Logic:** Traverse the string and maintain a count for each unique character using a Hash Map (Dictionary) or an Frequency Array (for ASCII).

### Approach 1: Python (Using Dictionary)

Python dictionaries are perfect for this as they map keys (characters) to values (counts).

```
def count_char_frequency(input_str):
    frequency = {}

    for char in input_str:
        if char in frequency:
            frequency[char] += 1
        else:
            frequency[char] = 1

    return frequency

text = "programming"
print(count_char_frequency(text))
# Output: {'p': 1, 'r': 2, 'o': 1, 'g': 2, 'a': 1, 'm': 2, 'i': 1, 'n': 1}
```

### Approach 2: C/C++ (Using Frequency Array)

Since `char` types are essentially integers (ASCII values), we can use an array of size 256.

```
#include <iostream>
#include <string.h>
using namespace std;

void countFreq(char str[]) {
    int freq[256] = {0}; // Initialize all to 0

    for(int i = 0; str[i] != '\0'; i++) {
        freq[str[i]]++; // Use ASCII value as index
    }

    // Print results
    for(int i = 0; i < 256; i++) {
        if(freq[i] != 0) {
            cout << (char)i << ": " << freq[i] << endl;
        }
    }
}
```

## 3. Python Concept: Strings & Built-in Methods

**Core Concept:** Strings in Python are **immutable**. You cannot change a character in place (e.g., `s[0] = 'a'` will throw an error).

## Common Built-in Methods

Method	Description	Example
<code>.lower()</code> / <code>.upper()</code>	Converts case.	<code>"Hi".lower()</code> → <code>"hi"</code>
<code>.strip()</code>	Removes leading/trailing whitespace.	<code>" hey ".strip()</code> → <code>"hey"</code>
<code>.split(delim)</code>	Splits string into a list based on delimiter.	<code>"a,b,c".split(",")</code> → <code>['a','b','c']</code>
<code>.join(iterable)</code>	Joins elements of a list into a string.	<code>"-".join(['a','b'])</code> → <code>"a-b"</code>
<code>.replace(old, new)</code>	Replaces substrings.	<code>"hello".replace("l", "p")</code> → <code>"heppo"</code>
<code>.find(sub)</code>	Returns index of first occurrence (or -1).	<code>"hello".find("e")</code> → <code>1</code>

## Slicing Syntax

- `string[start:stop:step]`
- Example: `text = "Python"`
  - `text[0:2]` → `"Py"`
  - `text[::-1]` → `"nohtyP"` (Reverse string)

## 4. C/C++ Concept: String Handling

**Core Concept:** In C, a string is a 1D array of characters terminated by a **Null Character** (`'\0'`).

### Memory Representation

```
char str[] = "Hello"; | H | e | l | l | o | \0 | |---|---|---|---|---| | 0 | 1 | 2 | 3 | 4 | 5 |
```

- The size of the array is usually Length + 1 (for the null terminator).
- If you forget the `\0`, the program may read garbage memory until it crashes.

### Important Functions ( `<string.h>` / `<cstring>` )

- `strlen(s)` : Returns length of string (excluding `\0`).
- `strcpy(dest, src)` : Copies content of `src` to `dest`.
- `strcat(dest, src)` : Concatenates (joins) `src` to the end of `dest`.
- `strcmp(s1, s2)` : Compares two strings.

- Returns 0 if equal.
- Returns <0 if s1 is ASCII-smaller than s2 .
- Returns >0 if s1 is ASCII-larger than s2 .

## 5. SQL: The WHERE Clause

**Purpose:** Used to filter records. It extracts only those records that fulfill a specified condition.

**Syntax:**

```
SELECT column1, column2
FROM table_name
WHERE condition;
```

### Operators used with WHERE

#### 1. Comparison Operators:

- = (Equal)
- <> or != (Not equal)
- > (Greater than), < (Less than)
- >= , <=

#### 2. Logical Operators:

- AND : Both conditions must be true.
- OR : At least one condition must be true.
- NOT : Negates a condition.

#### 3. Special Operators:

- **BETWEEN:** Selects values within a range.

```
SELECT * FROM Products WHERE Price BETWEEN 10 AND 20;
```

- **LIKE:** Search for a pattern (using % for wildcards).

```
SELECT * FROM Students WHERE Name LIKE 'A%'; -- Names starting with A
```

- **IN:** To specify multiple possible values for a column.

```
SELECT * FROM Employees WHERE City IN ('Paris', 'London', 'Berlin');
```

