📅 Day 17: Learning Summary

Aptitude: Calendars

The core of calendar aptitude is understanding how "extra" days shift the day of the week over time.

1. The Concept of Odd Days

Odd days are the remainder days left after dividing total days by 7.

- **Months:** * 31 days: 3 odd days ($31 \pmod 7$)
  - 30 days: 2 odd days
  - 28 days: 0 odd days
  - 29 days: 1 odd day
- **Years:**
  - Ordinary Year (365 days): 1 odd day.
  - Leap Year (366 days): 2 odd days.
- **Centuries (Cumulative):**
  - 100 years = 5 odd days
  - 200 years = 3 odd days
  - 300 years = 1 odd day
  - 400 years = 0 odd days (Cycles restart here)

2. Leap Year Rules

- **Rule 1:** If a year is not a century year, it must be divisible by 4.
- **Rule 2:** If it is a century year (ends in 00), it **must** be divisible by 400 to be a leap year (e.g., 2000 is a leap year, but 1900 is not).

💻 Programming: Find First Non-Repeating Character

This problem is best solved using a **Two-Pass Hash Map** approach to maintain linear time complexity.

The Algorithm

1. **Step 1:** Initialize a dictionary/frequency map. Iterate through the string and increment the count for each character.
2. **Step 2:** Iterate through the string a second time in its original order.
3. **Step 3:** Check the frequency map for each character. The first one with a count of `1` is the result.

Example Input: `"swiss"`

- Map: `{'s': 3, 'w': 1, 'i': 1}`
- Second Pass:
  - 's' -> count is 3 (skip)
  - 'w' -> count is 1 (**Found!**)

Concept: Python Polymorphism

Polymorphism allows for a unified interface for different underlying forms (data types).

Types in Python

- **Duck Typing:** "If it looks like a duck and quacks like a duck, it is a duck." Python doesn't check the type of an object, only that the required methods are present.
- **Method Overriding:** When a subclass provides a specific implementation for a method already defined in the parent class.

```
class Animal:
    def speak(self): pass

class Dog(Animal):
    def speak(self): return "Woof!"

class Cat(Animal):
    def speak(self): return "Meow!"
```

- **Operator Overloading:** Using the same operator for different purposes (e.g., `+` adds numbers but concatenates strings).

C++ Concept: Function Overloading

Function overloading is a feature where multiple functions have the same name but different parameters. It is resolved at **compile-time**.

Requirements for Overloading:

Functions must differ in:

1. **Number of parameters.**
2. **Data types of parameters.**
3. **Sequence of parameters.**

Important: The **return type** is not considered.

```
void print(int i) { cout << "Integer: " << i << endl; }
void print(double f) { cout << "Float: " << f << endl; }
```

```
void print(string s) { cout << "String: " << s << endl; }
```

SQL: GROUP BY Clause

The `GROUP BY` clause is used to arrange identical data into groups, typically to perform mathematical summaries.

Deep Dive on Syntax

- **Placement:** Comes after the `WHERE` clause but before `ORDER BY`.
- **The HAVING Clause:** Since `WHERE` cannot be used with aggregate functions, `HAVING` is used to filter groups *after* they are formed.

```
SELECT Department, AVG(Salary)
FROM Employees
GROUP BY Department
HAVING AVG(Salary) > 50000;
```

- **Aggregate Functions:**
  - `COUNT()` : Number of rows.
  - `SUM()` : Total value.
  - `AVG()` : Average value.
  - `MAX()` / `MIN()` : Extreme values