TechSaksham Training - Day 5 Notes

🔹 Object-Oriented Programming (OOPs) in Python

Why Use OOPs?

- Promotes **code reusability** and **modular programming**.
- Makes complex code more **structured and manageable**.

6 Pillars of OOPs:

1. **Class** - Blueprint for creating objects.
2. **Object** - An instance of a class.
3. **Inheritance** - Enables a class to acquire attributes & methods of another class.
4. **Abstraction** - Hides implementation details, exposing only the essentials.
5. **Polymorphism** - Allows different classes to share the same method names but behave differently.
6. **Encapsulation** - Restricts direct access to some of an object's components.

Creating a Class & Object

- In OOPs, functions are called **methods**, and parameters are called **attributes**.
- **Calling a class:**

```
class Sample:
    def greet(self):
        print("Hello, TechSaksham!")

obj = Sample()  # Creating an object
obj.greet()
```

🔹 Types of Methods in Python OOPs

1️⃣ Instance Method

- Can **only be accessed via an object**.

2️⃣ Class Method

- Can be accessed **without creating an object**.
- Uses **@classmethod decorator**.
- Uses `cls` instead of `self`.

```
class Example:
    class_variable = "Hello"

    @classmethod
    def show(cls):
        print(cls.class_variable)

Example.show()
```

3️⃣ Static Method

- Scope is limited **only to that function**.
- Uses **@staticmethod decorator**.
- Cannot access class or instance variables.

```
class Demo:
    @staticmethod
    def display():
        print("This is a static method.")

Demo.display()
```

🔹 Banking Application (Problem Statement)

Functionalities:

✅ Withdraw Money
✅ Deposit Money
✅ Check Balance
✅ Exit the Program

Approach:

📌 **User Authentication:** Validate PIN before allowing transactions.
📌 **Encapsulation:** Use a **Bank class** to handle transactions.
📌 **Validation:** Ensure amounts are **non-negative** and prevent overdraft.
📌 **Interactive Menu:** Loop through options until the user exits.

🔹 Inheritance in Python

Types of Inheritance:

1️⃣ **Single Inheritance** – One parent, one child.
2️⃣ **Multilevel Inheritance** – Parent → Child → Grandchild.
3️⃣ **Multiple Inheritance** – One child, multiple parents.
4️⃣ **Hierarchical Inheritance** – One parent, multiple children.
5️⃣ **Hybrid Inheritance** – Combination of the above types.

Diamond Problem in Python (OOPs)

📌 Occurs in **multiple inheritance** when a class inherits from two classes with a common ancestor.
📌 **Method Resolution Order (MRO)** helps determine which method to call.

```
class A:
    def show(self):
        print("Class A")

class B(A):
    def show(self):
        print("Class B")

class C(A):
    def show(self):
        print("Class C")

class D(B, C):  # Multiple Inheritance (Diamond Structure)
    pass

obj = D()
obj.show()  # Which show() method will be called?
```

🔹 Constructor & Destructor in Python

Constructor (`__init__`)

- Automatically **called when an object is created**.

Destructor (`__del__`)

- Called when an object is **deleted or goes out of scope**.

🔹 Classroom Activity: Problem-Solving & Code Explanation

📌 Each student explained **one problem, its approach, and implementation**.

Key Concepts Discussed:

✅ **Map & Filter Functions** – Used for efficient data handling.
✅ **Sorting a List** – Optimized sorting techniques.
✅ **Finding GCD of Two Numbers**.
✅ **Optimized Prime Number Detection**.
✅ **Neon Numbers, Armstrong Numbers, and Magic Numbers**.
✅ **Linear Search & Finding the Second Smallest Number**.
✅ **Pattern Printing (Butterfly & Binary Patterns)**.
✅ **Swapping Two Numbers Using Two Different Methods**.
✅ **Solving Quadratic Equations** in Python.

💡 Summary:

Day 5 focused on **Python OOPs, inheritance, constructors, and method resolution order (MRO)**. The interactive classroom activity helped solidify problem-solving skills. Looking forward to diving into **Data Structures & Algorithms next!** 🚀