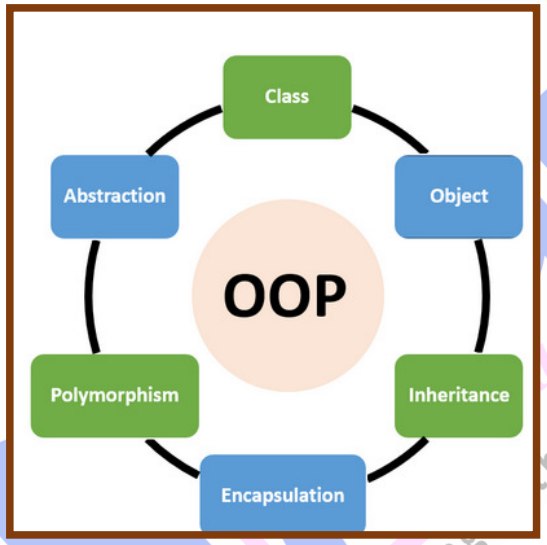
Overview of OOP (Object-Oriented Programming) terminology in Python

What is OOP (Object-Oriented Programming)?

OOP stands for Object-Oriented Programming, a programming approach where we design programs using objects that represent real-world entities. Python supports OOP principles such as classes, objects, inheritance, encapsulation, and polymorphism.

This approach makes your code:
- Easier to manage
- More reusable
- Easier to understand and extend



Why Use OOP in Python?

- Modularity: Code is organized into classes and objects, making it easier to manage.
- Reusability: Inheritance allows for code reuse, reducing redundancy.
- Maintainability: Encapsulation and abstraction make the code easier to maintain and modify.
- Flexibility: Polymorphism allows for flexible and dynamic method implementations.

Overview of Object-Oriented Programming Terminology in Python

Key OOP Terminologies in Python
Let's understand each one clearly:

1. Class
A **class** is like a **blueprint or template** for creating objects.
It defines **attributes (data)** and **methods (functions)** that describe how the object behaves.

Example:
```
class Car:
    # attributes
    brand = "Tata"
    model = "Nexon"

    # method
    def show_details(self):
        print("Brand:", self.brand)
        print("Model:", self.model)
```
Here,
- Car is a **class**
- brand and model are **attributes**
- show_details() is a **method**

2. Object
An **object** is an **instance of a class**.
Once the class is defined, you can create as many objects as you want.

Example:
```
# Creating an object of the class
my_car = Car()

# Accessing method
my_car.show_details()
```

Output:
Brand: Tata
Model: Nexon
Each object will have its own copy of the class data and can access its methods.

3. Method
A **method** is a function that belongs to a class.
It operates on the **data (attributes)** that belong to the object.
In Python, every method inside a class has at least one parameter — self, which refers to the **current object**.

Example:

```
class Student:
    def greet(self):
        print("Hello, student!")

s1 = Student()
s1.greet()
```

Output:
Hello, student!

4. Encapsulation (Hiding the Data)
Encapsulation means **keeping the data (variables) and the methods (functions) that work on that data together in one place — inside a class.**
It also means **restricting direct access** to some details of an object to protect the data.

Simple meaning:
It's like keeping your personal things in a locker — only you (or people with the key) can open it. Others can use the locker through you, but not open it directly.

In Python:
We use encapsulation to protect important information, such as passwords or account balances.
We usually make data private by adding **double underscores (__)**, so it can't be changed directly from outside the class.

Why it's useful:
- Keeps your data safe from unwanted modification.
- Makes your program more secure and controlled.
- Helps organize your code better.

5. Inheritance (Reusing Code)
Inheritance means **a class can borrow properties and behaviors (methods) from another class.**
The class that gives its features is called the **Parent (or Base) class**, and the one that receives them is the **Child (or Derived) class.**

Simple meaning:
It's like a child inheriting qualities from parents — such as habits, looks, or talents — but the child can also have their own unique qualities.

In Python:
It allows us to **reuse existing code** instead of writing it again.

For example, if a Vehicle class has features like speed and color, a Car class can inherit them and add its own features like air conditioning or music system.

Why it's useful:
- Reduces code duplication.
- Makes maintenance easier.
- Builds logical relationships between classes.

6. Polymorphism (Many Forms of the Same Thing)
Polymorphism means **the same function or method name can work in different ways depending on the object that uses it.**

Simple meaning:
It's like the word *"run"* — a human runs on legs, a computer runs a program, and a car runs on petrol.
Same word, different meanings depending on the context.

In Python:
Different classes can have a method with the same name (like speak()), but each class defines it differently.
When you call that method, Python automatically uses the version that matches the object.

Why it's useful:
- Increases flexibility.
- Makes code cleaner and easier to understand.
- Allows one function to handle different types of objects.

7. Abstraction (Showing Only What's Needed)
Abstraction means **hiding complex details and showing only the essential features to the user.**

Simple meaning:
Think of a **TV remote** — you press a button to change the channel, but you don't know or need to know how the signals are processed inside.
You just use the simple interface provided.

In Python:
We use abstraction to make programs easier to use and understand.
The user only interacts with necessary parts — the rest is hidden inside the class.

Why it's useful:
- Reduces complexity.
- Makes the program easier to use and maintain.
- Helps focus on *what* an object does, not *how* it does it.