OOPs (Object-Oriented Programming System) kya hota hai?

Python mein OOPs (Object-Oriented Programming) ek tarika hai jisme code ko objects aur classes ke form mein likhte hain. Ye approach real-world entities ko represent karne ke liye use hoti hai. Simple words mein, OOPs ka use code ko zyada organized, reusable aur easy-to-understand banane ke liye hota hai.

Basic Concepts of OOPs in Python

- 1️⃣ Class: Ek blueprint hota hai jisme hum objects ka structure define karte hain.
- 2️⃣ Object: Class ka actual instance hota hai jo memory mein exist karta hai. (jaise name, age, salary etc)
- 3️⃣ Attributes (Properties): Object ke features hote hain (jaise ek car ka color, speed, etc.).
- 4️⃣ Methods (Behaviors): Object ke actions hote hain (jaise ek car ka chalna, rukna, etc.).

📌 Class Attribute Kya Hota Hai?

- Class attributes wo variables hote hain jo directly class ke andar define kiye jaate hain aur sabhi objects ke liye common hote hain.

```
class Student:
    school = "ABC School"  # Class Attribute (Sab students ke liye same)

# Objects create kiye
s1 = Student()
s2 = Student()

print(s1.school)  # ABC School
print(s2.school)  # ABC School

# Object se class attribute change (Sirf s1 ke liye change hoga)
s1.school = "XYZ School"
print(s1.school)  # XYZ School
print(s2.school)  # ABC School

# Class ke through class attribute change (Sab objects ke liye change hoga)
Student.school = "PQR School"
print(s1.school)  # XYZ School (kyunki yeh ab instance attribute ban gaya hai)
print(s2.school)  # PQR School
```

Instance Attribute Kya Hota Hai?

- Instance Attribute wo variable hota hai jo har object ke liye alag hota hai. Ye class ke bahar define kiya jata hai aur object ke through access kiya jata hai.

```
class Student:
    # Instance Attributes ko directly object ke through set karte hain
    pass
# Objects create kiye
s1 = Student()
s2 = Student()

# Instance Attributes set kiye object ke through
s1.name = "Amit"
s1.age = 20
s2.name = "Neha"
s2.age = 22

# Instance Attributes ko print karte hain
print(s1.name, s1.age)  # Amit 20
print(s2.name, s2.age)  # Neha 22
```

self Parameter Kya Hai?

- self ek special keyword hai jo instance method mein use hota hai. Jab hum class ke methods ko define karte hain, tab self ki madad se hum current object ke attributes aur methods ko access kar sakte hain.

- self ka matlab hai current instance ya current object. Ye automatically object ke reference ke roop mein pass hota hai jab hum method ko call karte hain.
-

Self ka kaam:

1. Instance attributes ko access karna: self se hum object ke attributes ko access kar sakte hain.
2. Instance methods ko call karna: self se hum object ke methods ko bhi call kar sakte hain.

```
class Person:
    def __init__(self, name, age):  # Constructor
        self.name = name  # Instance attribute
        self.age = age    # Instance attribute

    def greet(self):  # Method
        print(f"जय श्री राम, my name is {self.name} and I am {self.age} years
old.")

# Object create kiya
person1 = Person("loha", 99999)
```

```
# Method ko call kiya
person1.greet()  # जय श्री राम, my name is loha and I am 99999 years old.
```

Explanation:

- init method: Jab hum object person1 = Person("loha", 99999) banate hain, to ye method self ki madad se name aur age ko object ke attributes mein set karta hai.

- greet method: Jab person1.greet() call kiya jata hai, to self ki madad se object person1 ke attributes (name aur age) ko access karke print kiya jata hai.

Important Note:

- self ko manually pass nahi karte, Python automatically method call karte waqt object ko pass kar deta hai.
- self naam ko hum change kar sakte hain, lekin ye ek convention hai aur generally hum self hi use karte hain.
- self object ko reference karta hai.
- Ye method ke andar instance variables aur methods ko access karne ke liye zaroori hota hai.

Constructor Kya Hota Hai?

- Constructor ek special method hota hai jo jab bhi object create hota hai tab automatically call hota hai. Constructor ka main kaam hota hai object ke initial state ko set karna (jaise instance variables ko initialize karna).

- Python mein constructor ka naam hota hai init(). Ye method automatically call hota hai jab hum class ka object create karte hain.

Constructor ka Use:

- Object ke attributes ko initialize karna.
- Object create karte waqt initial values assign karna.

```
class Car:
    def __init__(self, brand, model, year):  # Constructor
        self.brand = brand  # Instance variable
        self.model = model  # Instance variable
        self.year = year    # Instance variable

    def display_info(self):
        print(f"Car Brand: {self.brand}, Model: {self.model}, Year:
{self.year}")

# Object create karte waqt constructor ko call kiya
car1 = Car("Toyota", "Corolla", 2020)
```

```
car2 = Car("Honda", "Civic", 2022)

# Method ko call kiya
car1.display_info()  # Car Brand: Toyota, Model: Corolla, Year: 2020
car2.display_info()  # Car Brand: Honda, Model: Civic, Year: 2022
```

Explanation:

- init(self, brand, model, year): Ye constructor method hai jo car1 aur car2 objects ke liye initial values assign karta hai.

- Jab car1 = Car("Toyota", "Corolla", 2020) run hota hai, to constructor automatically call hota hai aur brand, model, aur year ko car1 object ke attributes se assign karta hai.

Static Method Kya Hota Hai?

Static Method wo method hota hai jo class ke object ke bina call kiya ja sakta hai. Ye method class ke level pe hota hai, na ki object ke level pe. Static method ko class ke state (attributes) ki zarurat nahi hoti, aur ye independent hota hai.

Static Method ka Use:

- Agar aapko class ke state se koi interaction nahi karna hai, sirf kisi utility function ki tarah method ka use karna hai.

- Ye method object ke bina directly class se call kiya ja sakta hai.

- Static Method ka Syntax: Static method ko define karne ke liye hum **@staticmethod** decorator ka use karte hain.

```
class MathOperations:

    @staticmethod
    def add(a, b):  # Static Method
        return a + b

    @staticmethod
    def multiply(a, b):  # Static Method
        return a * b

# Static method ko class se call kiya jaa sakta hai, bina object banaye
result_add = MathOperations.add(5, 3)
result_multiply = MathOperations.multiply(5, 3)
```

```
print(f"Sum: {result_add}")          # Sum: 8
print(f"Product: {result_multiply}") # Product: 15
```

Explanation:

1. @staticmethod decorator se hum class ke methods ko static method banate hain.
2. add(a, b) aur multiply(a, b) methods ko hum object ke bina directly class se call kar rahe hain.

Static Method ka Key Point:

- Static methods ko class ke instance ke bina call kiya ja sakta hai.
- Ye object-specific data ko access nahi karte, na hi ye instance (self) ya class (cls) ko pass karte hain.

Example without Static Method (for comparison):

```
class MathOperations:
    def add(self, a, b):  # Instance Method
    return a + b

# Object create karte waqt
math = MathOperations()

# Instance method ko call karne ke liye object ki zarurat padti hai
result_add = math.add(5, 3)

print(f"Sum: {result_add}")  # Sum: 8
```

Aap dekh sakte ho ki instance method ko call karte waqt object ka reference (self) pass hota hai, jabki static method mein aisa nahi hota.

Inheritance Kya Hota Hai?

Inheritance ek concept hai jisme ek class (child class) apne parent class ke attributes aur methods ko inherit karti hai. Iska matlab hai ki child class ko parent class ke features milte hain, aur child class apne specific behavior ko add kar sakti hai.

Inheritance ka Use:

- Code reusability: Aapko bar-bar same code likhne ki zarurat nahi padti.
- Extend functionality: Parent class ki functionality ko child class mein extend kar sakte hain.

Types of Inheritance:

Single Inheritance: Ek child class ek parent class se inherit karti hai.

- Example:

```
class Animal:  # Parent Class
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound")

# Child class Animal se inherit kar rahi hai
class Dog(Animal):  # Child Class
    def __init__(self, name, breed):
        super().__init__(name)  # Parent class constructor ko call kiya
        self.breed = breed

    def bark(self):  # New method specific to Dog class
        print(f"{self.name} barks")

# Object of Dog class
dog1 = Dog("Tommy", "Labrador")

# Inherited method
dog1.speak()  # Tommy makes a sound

# Child class specific method
dog1.bark()  # Tommy barks
```

Explanation:

- Animal class ke paas speak() method hai.
- Dog class, Animal class se inherit kar rahi hai. Iska matlab Dog class ko Animal class ka speak() method milta hai.
- super() ka use karke child class ke constructor ko parent class ka constructor call karta hai.

Key Points:

- Child class apne parent class ki all methods aur attributes ko inherit karti hai.
- Child class apni new methods bhi define kar sakti hai.
- super() ka use parent class ke methods aur constructor ko access karne ke liye hota hai.

Multiple Inheritance: Ek child class multiple parent classes se inherit karti hai.

- Example:

```
class Animal:
    def __init__(self, name):
        self.name = name

    def speak(self):
        print(f"{self.name} makes a sound")

class Dog(Animal):  # Dog class inherits Animal
    def __init__(self, name, breed):
        super().__init__(name)
        self.breed = breed

    def bark(self):
        print(f"{self.name} barks")

class Puppy(Dog):  # Puppy class inherits Dog, which inherits Animal
    def __init__(self, name, breed, age):
        super().__init__(name, breed)
        self.age = age

    def play(self):
        print(f"{self.name} plays")

# Object of Puppy class
puppy1 = Puppy("Buddy", "Golden Retriever", 2)

puppy1.speak()  # Buddy makes a sound (inherited from Animal)
puppy1.bark()   # Buddy barks (inherited from Dog)
puppy1.play()   # Buddy plays (specific to Puppy)
```

Explanation:

- Puppy class, Dog class se inherit karti hai, jo ki Animal class se inherit karti hai.
- Puppy class ko speak(), bark(), aur play() methods milte hain.

Multilevel Inheritance: Ek child class doosri child class se inherit karti hai, jo parent class se inherit karti hai.

- Example:

```
class Animal:
    def speak(self):
        print("Animal makes a sound")
```

```python
class Dog:
    def bark(self):
        print("Dog barks")

class DogAnimal(Animal, Dog):  # Multiple inheritance
    def __init__(self, name):
        self.name = name

# Object of DogAnimal class
dog1 = DogAnimal()
dog1.speak()  # Inherited from Animal
dog1.bark()   # Inherited from Dog
```

Explanation:

- DogAnimal class dono Animal aur Dog class se inherit kar rahi hai.

Conclusion:

- Inheritance se aap apne class ko easily extend aur modify kar sakte ho, without rewriting the entire code.
- super() ka use parent class ko access karne ke liye hota hai.

🔥 OOPs ke Important Features

- Encapsulation – Data ko secure rakhna. (Private variables, getters, setters)
- Inheritance – Ek class doosri class ke features inherit kar sakti hai.
- Polymorphism – Same function ka multiple forms mein use.
- Abstraction – Sirf necessary details dikhana, baaki hide karna.

🎯 OOPs ka use karke hum code ko reusable, maintainable aur secure bana sakte hain! 😃