Advanced OOP Concepts in Python

1. Class & Static Variables

1. Class Variables

Class variables wo variables hotay hain jo **class ke andar** define kiye jaate hain lekin **kisi function ke andar nahi hote**. Ye **saare objects ke darmiyan shared** hote hain.

- Ye variable **class level par hota hai**.
- Saare objects (instances) isay **share** karte hain.
- Isay aap **class name** ya **object** dono se access kar saktay ho.
-
- Agar aap isay **class se change karo**, to har object mein update ho jata hai.

```
class Student:

    # Class variable

    school_name = "Allied School"


    def __init__(self, name):

        self.name = name  # Instance variable


# Objects banayein

s1 = Student("Ali")

s2 = Student("Sara")


print(s1.school_name)  # Allied School

print(s2.school_name)  # Allied School


# Class level par variable change karo

Student.school_name = "City School"
```

print(s1.school_name) # City School

print(s2.school_name) # City School

`school_name` ek **class variable** hai jo **dono students ke liye same hai**.

Jab `Student.school_name` ko update kiya, to **dono objects mein naya naam dikh raha hai**.

`self.name` har object ka apna hota hai — ye **instance variable** kehlata hai.

2. Static Variables

Python mein "static variable" ka concept **class variables ke jesa hi hota hai**. Yahan **`static` keyword nahi hota** jaise Java mein hota hai.

✅ Baatein Jo Yaad Rakho:

- Static variable aur class variable Python mein **ek hi cheez** hain.
- Ye class ke andar define kiye jaate hain, aur **har object isay share karta hai**.
- Static variables ko change karne ka best tareeqa hota hai: **class name se change karo**.

2. Composition vs Aggregation

Composition: Strong relation - inner object delete ho jata hai.

Aggregation: Weak relation - inner object alag exist karta hai.

Example (Composition):

class Engine: pass

class Car:

 def __init__(self):

 self.engine = Engine()

Example (Aggregation):
engine = Engine()

class Car:

 def __init__(self, engine):

 self.engine = engine

Roman Urdu: Composition mein dono juday hotay hain, Aggregation mein loose connection.

3. Method Resolution Order (MRO)

Definition: Decide karta hai method pehle kis class se call hogi.

Example:

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

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

class C(B, A): pass

obj = C()

obj.show() # B

Roman Urdu: Pehle B check hoti hai, jahan milta hai wahi se method chalti hai.

4. Decorators in Class

Decorators asal mein ek function hota hai jo doosre function ya method ko modify karta hai bina uske code ko directly badle. Yeh ek tarah ka wrapper hota hai jo original function ke behavior ko change kar sakta hai.

Class ke andar Decorators

Class ke andar decorators ko aksar **methods** ke upar lagaya jata hai, jese:

- `@staticmethod`
- `@classmethod`
- `@property`

Yeh built-in decorators hain jo method ki functionality ko modify karte hain.

Common class decorators:

1. **@staticmethod**
   Is decorator se hum class ke andar koi method define karte hain jo **instance variable** ya **self** ko access nahi karta. Is method ko hum class ke object ke bina bhi call kar sakte hain.

2. **@classmethod**
   Is decorator se method ko modify karte hain jisse woh class khud ko (class ko) first argument ke taur pe leta hai, usually `cls` naam se. Is method ko bhi hum bina object banaye call kar sakte hain.

3. **@property**
   Yeh decorator method ko aise behave karwata hai jaise woh ek attribute (property) ho, bina brackets ke call kar sakte hain.

```
class Car:
    wheels = 4  # Class variable

    def __init__(self, model):
        self.model = model  # Instance variable

    @staticmethod
    def honk():
        print("Beep! Beep!")

    @classmethod
    def number_of_wheels(cls):
        print(f"Har car mein {cls.wheels} wheels hote hain.")

    @property
    def car_model(self):
        return self.model

# Object banta hai
my_car = Car("Toyota")

# Static method bina object ke call kar sakte hain
Car.honk()        # Output: Beep! Beep!

# Class method bhi bina object ke call kar sakte hain
```

```
Car.number_of_wheels()  # Output: Har car mein 4 wheels hote hain.

# Property method ko bina brackets ke call karte hain
print(my_car.car_model)  # Output: Toyota
```

@staticmethod method ko instance ya class variable ki zarurat nahi hoti, direct class se call kar sakte hain.

@classmethod method ko class ka reference chahiye hota hai, jise hum `cls` kehte hain.

@property method ko aise use karte hain jaise woh attribute ho, bracket ke bina access karte hain.

Custom Decorator kya hota hai?

Custom decorator ek aisa function hota hai jo doosre function/method ko wrap karta hai aur uske behavior ko modify karta hai.

```
def mera_decorator(func):
    def wrapper(*args, **kwargs):
        print("Decorator: Method call hone se pehle")
        result = func(*args, **kwargs)  # Original method ko call karte hain
        print("Decorator: Method call hone ke baad")
        return result
    return wrapper

class Gari:
    def __init__(self, model):
        self.model = model

    @mera_decorator
    def start(self):
        print(f"{self.model} start ho rahi hai.")

# Object banta hai
meri_gari = Gari("Honda")

# Decorated method call karte hain
meri_gari.start()
```

5. Callable Objects

Callable objects woh objects hotay hain jinhein aap function ki tarah **parentheses** `()` laga ke call kar sakte hain. Matlab, agar kisi object ke baad `()` lagayen aur woh successfully execute ho jaye, to woh object callable hai.

Function khud bhi callable hota hai. Magar Python mein functions ke ilawa bhi dusray types ke objects callable ho sakte hain, jaise ke:

- Functions

- Methods
- Classes (jab aap class ko call karte hain to uska constructor chalta hai)
- Objects jin mein `__call__` method define ho

Callable object woh hota hai jisko `()` lagake call kar saken.

Functions aur classes Python mein naturally callable hain.

Aap apni class mein `__call__` method define kar ke uske objects ko bhi callable bana sakte hain.

Callable hone ka matlab yeh hota hai ke aap us object ko function ki tarah use kar sakte hain.

6. Module vs Package

Module ek single `.py` file hoti hai jisme Python ka code likha hota hai — functions, classes, variables waghera.

Use: Aap module ko import karke uske functions ya classes ko kisi aur file mein use kar sakte hain.

Package kya hota hai?

➤ Package ek folder hota hai jo multiple modules ko organize karta hai. Har package ke andar ek `__init__.py` file hoti hai (empty bhi ho sakti hai), jo Python ko batata hai ke yeh folder ek package hai.

Use: Packages large projects ke liye useful hote hain — jahan multiple related modules hotay hain.

Feature	Module	Package

Kya hota hai?	Ek single Python `.py` file	Ek folder jisme multiple modules hote hain
File Type	`.py` file	Folder with `__init__.py` file
Example	`math.py`, `utils.py`	`numpy`, `pandas` (yeh packages hain)
Import kaise?	`import module_name`	`from package import module`

7. Error Handling

Python mein jab koi **error ya exception** hota hai, to program crash kar jata hai agar aap us error ko handle nahi karte.

OOP (Object-Oriented Programming) mein hum errors ko **classes** aur **objects** ke zariye bhi **handle** kar sakte hain.

OOP ke andar Error Handling kaise karte hain?

```
class Calculator:

    def divide(self, a, b):

        try:

            result = a / b

            return result

        except ZeroDivisionError:

            return "Error: Division by zero is not allowed."
```

```
calc = Calculator()

print(calc.divide(10, 2))  # Output: 5.0

print(calc.divide(10, 0))  # Output: Error: Division by zero is not allowed.
```

Aap apni error class bhi bana sakte hain jo `Exception` class se inherit kare:

```
class NegativeNumberError(Exception):
    pass

class MathOperations:
    def square_root(self, number):
        if number < 0:
            raise NegativeNumberError("Negative number ka square root nahi nikal sakte.")
        return number ** 0.5

math = MathOperations()

try:
    print(math.square_root(9))     # Output: 3.0
    print(math.square_root(-4))    # Raises custom error
except NegativeNumberError as e:
    print("Custom Error:", e)
```

Concept	**Explanation in Roman Urdu**
`try-except`	Error ko pakarne ke liye
`except SomeError:`	Specific error handle karne ke liye
`raise`	Apni error throw karne ke liye
`Exception`	Sab errors ka base class
Custom Error Class	Apni error logic define karne ke liye class banani

8. Advanced OOP Concepts

Includes: Inheritance, Polymorphism, Encapsulation,

Abstraction Example (Polymorphism):

```
class Dog: def speak(self): print("Bark")

class Cat: def speak(self): print("Meow")

def animal_sound(animal): animal.speak()

animal_sound(Dog())

animal_sound(Cat())
```

Roman Urdu: Same method, alag object pe alag kaam.

9. Testing OOP Code

Example:

```
import unittest

class TestMath(unittest.TestCase):

 def test_add(self):

 self.assertEqual(2 + 2, 4)

unittest.main()
```

Roman Urdu: Har part ka test likhna zaroori hai.

10. SOLID Principles

🔟 S.O.L.I.D ka matlab kya hai?

Letter	**Principle Name**	**Roman Urdu Meaning (Asaan)**
S	**Single Responsibility Principle**	Ek class sirf ek kaam kare
O	**Open/Closed Principle**	Code open ho extension ke liye, closed ho change ke liye

L	**Liskov Substitution Principle**	Subclass parent ki jagah use ho sakti ho
I	**Interface Segregation Principle**	Bade interface ke bajaye chhote chhote interfaces banao
D	**Dependency Inversion Principle**	High-level classes low-level pe depend na karein directly

11. Iterable Protocol & Iterators

Agar koi object `for loop` mein use ho sakta hai to wo **Iterable** hai.

✅ Examples of Iterables:

- `list`
- `tuple`
- `string`
- `dict`
- `set`

Iterator woh object hota hai jo `next()` function se agla item deta hai, jab tak items khatam na ho jaayein.

Agar aap kisi **iterable** (jaise list) pe `iter()` lagao, to aapko **iterator** milta hai.

`__iter__()` class ko iterable banata hai.

`__next__()` se har baar naya number milta hai.

Jab limit poori ho jaati hai to `StopIteration` raise hota hai.

12. Object-Based vs Object-Oriented

Object-based: Objects hain, inheritance nahi.

Object-oriented: Full features like inheritance,

encapsulation. Roman Urdu: OOP fully supported ho to

object-oriented.

13. Python Unified Type System

Definition: Sab kuch object hai.

Example:

```
print(type(5)) # <class 'int'>

print(type("hello")) # <class 'str'>
```

Roman Urdu: Har cheez Python mein object hai.