📘Python OOP – Classes and Objects

🕰️1. Introduction to OOP

- **OOP (Object-Oriented Programming)** allows modeling real-world entities as **objects** in Python.
- Main concepts: **Classes, Objects, Attributes, Methods, Inheritance, Polymorphism, Encapsulation**.
- Focus here: **Classes, Objects, self, init, Methods, Variables**.

🕰️2. Classes and Objects

- **Class:** Blueprint for objects. Defines attributes (variables) and methods (functions).
- **Object:** Instance of a class.

Syntax:

```
class ClassName:
    # attributes
    # methods

# Creating object
obj = ClassName()
```

Example 1: Simple Class and Object

```
class Person:
    pass

p1 = Person()
print(type(p1))  # <class '__main__.Person'>
```

🕰️3. The `self` Parameter

- `self` refers to the instance of the class.
- Used to access instance attributes and methods.
- Must be the **first parameter in all instance methods**.

Example:

```python
class Person:
    def greet(self, name):
        print(f"Hello, {name}!")

p1 = Person()
p1.greet("Alice")
```

Output:

```
Hello, Alice!
```

🕰️4. The `__init__` Constructor

- `__init__` is called automatically when an object is created.
- Used to initialize object attributes.

Syntax:

```python
class ClassName:
    def __init__(self, param1, param2):
        self.attr1 = param1
        self.attr2 = param2
```

Example:

```python
class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

p1 = Person("Alice", 25)
print(p1.name, p1.age)
```

Output:

```
Alice 25
```

🕰️5. Methods in Classes

- **Instance Methods:** Access instance attributes via `self`.
- **Class Methods:** Access class attributes via `@classmethod`.
- **Static Methods:** Do not access instance or class attributes. Use `@staticmethod`.

Example: Instance Method

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

    def greet(self):
        print(f"Hello, {self.name}!")

p1 = Person("Alice")
p1.greet()
```

Output:

```
Hello, Alice!
```

Example: Class Method

```python
class Person:
    species = "Human"

    @classmethod
    def show_species(cls):
        print(cls.species)

Person.show_species()  # Human
```

Example: Static Method

```python
class MathOps:
    @staticmethod
    def add(a, b):
        return a + b

print(MathOps.add(5, 7))  # 12
```

🕰️6. Instance vs Class Variables

Instance Variables: - Unique to each object. - Defined inside `__init__` using `self`.

Class Variables: - Shared among all objects. - Defined directly inside class.

Example:

```
class Person:
    species = "Human"  # class variable

    def __init__(self, name, age):
        self.name = name  # instance variable
        self.age = age

p1 = Person("Alice", 25)
p2 = Person("Bob", 30)

print(p1.name, p1.age, p1.species)
print(p2.name, p2.age, p2.species)
```

Output:

```
Alice 25 Human
Bob 30 Human
```

Modifying Class Variable:

```
Person.species = "Homo Sapiens"
print(p1.species)  # Homo Sapiens
print(p2.species)  # Homo Sapiens
```

🕰️7. Local Variables in Methods

- Defined inside methods and exist only during method execution.
- Cannot be accessed outside the method.

Example:

```
class Calculator:
    def multiply(self, a, b):
        result = a * b  # local variable
        return result

calc = Calculator()
print(calc.multiply(5, 6))  # 30
```

- `result` is local to `multiply` method.

🕰️8. Real-Time Examples

1️⃣Employee Management System

```
class Employee:
    company = "XYZ Corp"  # class variable

    def __init__(self, name, salary):
        self.name = name  # instance variable
        self.salary = salary

    def display(self):
        print(f"Name: {self.name}, Salary: {self.salary}, Company:
{Employee.company}")

emp1 = Employee("Alice", 50000)
emp2 = Employee("Bob", 60000)

emp1.display()
emp2.display()
```

2️⃣Bank Account System

```
class BankAccount:
    bank_name = "ABC Bank"

    def __init__(self, owner, balance=0):
        self.owner = owner
        self.balance = balance

    def deposit(self, amount):
        self.balance += amount
        print(f"Deposited {amount}, New Balance: {self.balance}")

    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            print(f"Withdrawn {amount}, Remaining Balance: {self.balance}")
        else:
            print("Insufficient Balance")

acc1 = BankAccount("Alice", 1000)
acc1.deposit(500)
acc1.withdraw(2000)
```

🕰9. Summary

- **Class:** Blueprint for objects.
- **Object:** Instance of class.
- **self:** Refers to instance.
- `__init__`: Constructor for initialization.
- **Instance Variables:** Unique per object.
- **Class Variables:** Shared across objects.
- **Local Variables:** Exist only inside methods.
- **Methods:** Instance, Class, Static.

📥Download Options

- Download as DOCX
- Download as TXT
- Download as PDF