oop

November 16, 2025

1 Object Oriented Programming (OOP)

- Method of programming that focus on using object and class to organize and structure code.

1.0.1 Object

- Instance of class

1.0.2 Class

- Blueprint for creating object

1.0.3 Object contains

1. Data (Attributes)
2. Methods (Functions)

1.1 Fundamental principle of OOP

1. Encapsulation

- Concept of bundling the attribute and methods that work on the data/attribute into a single unit

2. Abstraction

- Hiding the complex implementation details and showing only the essential features of the object

3. Inheritence

- Allowing a new class to inherit attributes and methods from an existing class

4. Polymorphism

- Allowing objects of different class to be treated as a object of common superclass (overriding or overloading of methods)

1.2 Why OOP?

- Modularity
- Code reuse
- Real world modeling

- Maintainability

1.2.1 magic methods

[3]:

```python
# Empty class
class Person:
    pass
```

[6]:

```python
# Class with method

class Person:
    def hello(self):
        print("Hello, world!")

# Object creation
p = Person()
p.hello()
```

```
Hello, world!
```

[7]:

```python
# Class with attribute and method

class Person:
    name = "Hari"

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


p = Person()
p.hello()
```

```
Hello, Hari
```

[11]:

```python
class Person:
    name = "Hari"

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


p = Person()
p.hello()
print(p)
print(p.__doc__)
print(p.__class__)
```

```
Hello, Hari
<__main__.Person object at 0x748872c9a120>
```

```
None
<class '__main__.Person'>
```

[16]:

```python
class Person:
    """
    This is class Person, It has a method hello
    """
    name = "Hari"

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

    def __str__(self):
        return f"Object of class Person"


p = Person()
p.hello()
print(p)
print(p.__doc__)
print(p.__class__)
```

```
Hello, Hari
Object of class Person

This is class Person, It has a method hello

<class '__main__.Person'>
```

[23]:

```python
# Constructor and Destructor

class Person:
    """
    This is class Person, It has a method hello
    """
    name = "Hari"

    def __init__(self):
        print("This is constructor")

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

    def __str__(self):
        return f"Object of class Person"

    def __del__(self):
        print("This is destructor")
```

```
p = Person()
p.hello()
print(p)
print(p.__doc__)
print(p.__class__)
```

```
This is constructor
This is destructor
Hello, Hari
Object of class Person

This is class Person, It has a method hello

<class '__main__.Person'>
```

[31]:

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

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

p = Person("Gopal")
print(p.name)
p.greet()
```

```
Gopal
Hello, Gopal
```

[40]:

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

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

p = Person("Gopal")
# print(p.__name)
p.greet()
```

```
Hello, Gopal
```

[41]:

```
persons = [Person("Hari"), Person("Sita"), Person("Ram"), Person("Shyam")]

for person in persons:
    person.greet()
```

```
Hello, Hari
Hello, Sita
Hello, Ram
Hello, Shyam
```

[42]:

```python
# Q. Write a class User,
#     that have data name and age (store through constructor and make private)
#     Make a info method that prints, Hello, Ram. You are 20 years old.
#     finally make a object and class function.
```

[43]:

```python
class User:
    def __init__(self, name, age):
        self.__name = name
        self.__age = age

    def info(self):
        print(f"Hello, {self.__name}. You are {self.__age} years old.")

p = User("Gopal", 22)
# print(p.__name)
p.info()
```

```
Hello, Gopal. You are 22 years old.
```

2 Inheritence

1. Parent class -> base class, from which other class can inherit
2. Child class -> derived class, that inherits from the parent class

2.0.1 Types of inheritence

1. Single inheritence

- one child class is derived from one parent class

2. Multiple

- One child class is derived from more then one parent class

3. Multilevel

- One child class is derived from one parent class and another grandchild class is derived from child class

4. Hierarichal

- More then one child class are derived from one parent class

5. Hybrid

- Any two or more combinations

[52]:

```
# Single Inheritence

class Person:
    __name = "Ram"

    def display_name(self):
        print(f"Name: {self.__name}")

class Employee(Person):
    company = "STN"

    def info(self):
        self.display_name()
        # print(f"Name: {self.__name}")
        print(f"Company: {self.company}")
```

[53]:

```
employee = Employee()
employee.info()
```

```
Name: Ram
Company: STN
```

[56]:

```
# Using constructor

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

    def display_name(self):
        print(f"Name: {self.__name}")

class Employee(Person):
    def __init__(self, name, company):
        self.company = company
        super().__init__(name)

    def info(self):
        self.display_name()
        # print(f"Name: {self.__name}")
        print(f"Company: {self.company}")
```

[57]:

```
employee = Employee("Ram", "STN")
employee.info()
```

```
Name: Ram
Company: STN
```

[2]:

```python
# Polymorphism

class Bird:
    def fly(self):
        return "Bird: Flying in the sky"


class Airplane:
    def fly(self):
        return "Airplane: Flying using fuel"
```

[ ]:

```python
b = Bird()
p = Airplane()

# Normal way
print(b.fly())
print(p.fly())
```

```
Bird: Flying in the sky
Airplane: Flying using fuel
```

[5]:

```python
def flying(obj):
    print(obj.fly())

flying(p)
flying(b)
```

```
Airplane: Flying using fuel
Bird: Flying in the sky
```

[7]:

```python
# Abstraction

# 1. Abstract class -> can not create object
# 2. Abstract method -> must override

from abc import abstractmethod, ABC
import math

class Shape(ABC):
    @abstractmethod
    def area(self):
        pass
```

[8]:

```python
class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius

    def area(self):
        return math.pi * self.radius * self.radius
```

[9]:
```
c = Circle(5)
c.area()
```

[9]: 78.53981633974483

[11]:
```
class Rectangle(Shape):
    def __init__(self, l, b):
        self.l = l
        self.b = b

    def area(self):
        return self.l * self.b
```

[12]:
```
r = Rectangle(3, 4)
r.area()
```

[12]: 12

[14]:
```
# s = Shape()
```

[20]:
```
# Encapsulation

class Math:
    @staticmethod
    def add(a, b):
        return a + b

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

    @staticmethod
    def divide(a, b):
        return a / b
```

[21]:
```
m = Math()
m.add(3, 4)
```

[21]: 7

[22]:
```
m.multiply(4, 5)
```

[22]: 20

[23]:
```
m.divide(4, 2)
```

```
[23]: 2.0
```

```
[ ]:
```