14. OOP - Part 1

October 20, 2022

1 Introduction

- Python is a multi-paradigm programming language.
- One of the popular approaches to solve a programming problem is by creating objects. This is known as Object-Oriented Programming (OOP).
- An object has two characteristics:
  - attributes
  - behavior
- An object is an entity that has a state and a defined set of operations which operate on that state. The state is represented as a set of object attributes.

2 Class

- A class is a blueprint for creating objects (a particular data structure), providing initial values for state (member variables or attributes), and implementations of behavior (member functions or methods).
- In Python, objects are created using the `class` keyword.

[3]:
```
class Employee:
    '''Class of employees'''
    emp_id = 457839
```

[4]:
```
emp1 = Employee()
```

[5]:
```
emp1.emp_id
```

[5]:
```
457839
```

2.1 Constructor

- A constructor is a special method that is used to initialize objects.
- In python we use `__init__()` function to define constructor.
- The `__init__()` function is called automatically every time the class is being used to create a new object.

[6]:
```
class Employee:
    '''Employee class'''
    def __init__(self):
        self.emp_name = "Rohit"
        self.emp_id = 445851
        self.age = 30
```

2.2 The self parameter

- The `self` parameter is a reference to the current instance of the class, and is used to access variables that belongs to the class.

[7]:
```
e1 = Employee()
```

[8]:
```
e1.age
```

[8]:
```
30
```

[9]:
```
e1.emp_name
```

[9]:
```
'Rohit'
```

[10]:
```
# Parameterized constructor

class Employee:
    '''Employee class'''
    def __init__(self, emp_name, emp_id, age):
        self.emp_name = emp_name
        self.emp_id = emp_id
        self.age = age
```

[11]:
```
e2 = Employee("Akash", 786974, 32)
```

[12]:
```
e2.emp_name
```

[12]:
```
'Akash'
```

[13]:
```
e2.age
```

[13]:
```
32
```

[14]:
```
e1.age
```

[14]:
```
30
```

[ ]:
```
e1.emp_id
```

[19]:
```python
# Parameterized constructor

class Employee:
    '''Employee class'''
    def __init__(self, age=0, emp_name = "No Name", emp_id = 10000 ):
        self.emp_name = emp_name
        self.emp_id = emp_id
        self.age = age
```

[20]:
```python
e3 = Employee()
```

[21]:
```python
e3.emp_name
```

[21]:
```
'No Name'
```

[22]:
```python
e4 = Employee(28, "Raj", 4865798)
```

[23]:
```python
e4.emp_name
```

[23]:
```
'Raj'
```

[25]:
```python
emp4 =Employee(age=90, emp_id=456, emp_name="fff")
```

2.3 Modify Object Properties

[26]:
```python
e4.emp_id
```

[26]:
```
4865798
```

[27]:
```python
e4.emp_id = 786985
```

[28]:
```python
e4.emp_id
```

[28]:
```
786985
```

2.4 Object Methods

Methods in objects are functions that belong to the object.

[35]:
```python
class Employee:
    '''Class of employees'''
    def __init__(self, name = "Akash", emp_id = 487587, age = 30):
        self.name = name
        self.emp_id = emp_id
        self.age = age

    def get_details(self):
        '''Display employee details.'''
        print("Emp Name:",self.name)
        print("Emp Id:", self.emp_id)
        print("Age:", self.age)
```

[33]:
```
e2 = Employee("Pratap", 4484754, 45 )
```

[34]:
```
e2.get_details()
```

```
Emp Name: Pratap
Emp Id: 4484754
Age: 45
```

[36]:
```
e3 = Employee()
```

[37]:
```
e3.get_details()
```

```
Emp Name: Akash
Emp Id: 487587
Age: 30
```

- **Always add docstring to your classes and methods**

3 Inheritance

- Inheritance allows us to define a class that inherits all the methods and properties from another class.
- Parent class is the class being inherited from, also called base class.
- Child class is the class that inherits from another class, also called derived class.

3.1 Create a Parent Class

- Any class can be a parent class, so the syntax is the same as creating any other class.

[43]:
```
class Person:
    def __init__(self, fname, lname):
        self.firstname = fname
        self.lastname = lname

    def printname(self):
        print(self.firstname, self.lastname)
```

[39]:
```
p1 = Person("Amitabh","Bachchan")
```

[40]:
```
p1.printname()
```

```
Amitabh Bachchan
```

3.2 Create a Child Class

- To create a class that inherits the functionality from another class, send the parent class as a parameter when creating the child class.

[41]:
```
class Student(Person):
    pass
```

[42]:
```
s1 = Student("Ritik","Rawat")
```

[44]:
```
s1.printname()
```

```
Ritik Rawat
```

3.3 Add __init__() to child class

[45]:
```
class Student(Person):

    def __init__(self, fname, lname, school_name = "ACTS"):
        Person.__init__(self, fname, lname)
        self.school_name = school_name # also new properties can be added

    def school_info(self):
        print("School Name:", self.school_name)
```

[46]:
```
s2 = Student("Ritik","Rawat")
```

[47]:
```
s2.printname()
```

```
Ritik Rawat
```

[48]:
```
s2.school_info()
```

```
School Name: ACTS
```

3.4 the super() Function

- Makes the child class inherit all the methods and properties from its parent

[49]:
```
class Student(Person):
    '''Class to define student and its methods'''
    def __init__(self, fname, lname, school_name = "ACTS"):
        super().__init__(fname, lname)
        self.school_name = school_name # also new properties can be added

    def school_info(self,city):
        '''Function to get info about school.'''
        print("School Name:", self.school_name, "in",city)
```

[50]:
```python
s5 = Student("Ritik","Rawat")
```

[51]:
```python
s5.printname()
```

```
Ritik Rawat
```

[52]:
```python
s5.school_info("Bangalore")
```

```
School Name: ACTS in Bangalore
```

3.5 Multiple Inheritance

- A class can be derived from more than one base class in Python, similar to C++. This is called multiple inheritance.
- In multiple inheritance, the features of all the base classes are inherited into the derived class.

[53]:
```python
# Base class 1
class Mother:
    def __init__(self, mothername):
        self.mothername = mothername

    def mother(self):
        print(self.mothername)

# Base class 2
class Father:
    def __init__(self, fathername):
        self.fathername = fathername

    def father(self):
        print(self.fathername)
```

[54]:
```python
class Son(Mother, Father):
    '''Class to define son'''
    def __init__(self,mothername, fathername):
        Mother.__init__(self, mothername)
        Father.__init__(self, fathername)

    def parents(self):
        print("Father :", self.fathername)
        print("Mother :", self.mothername)
```

[55]:
```python
child = Son("mom","dad")
```

[56]:
```python
child.parents()
```

```
Father : dad
Mother : mom
```

[57]:
```
child.mother()
```

```
mom
```

[58]:
```
child.father()
```

```
dad
```

3.6 Multilevel Inheritance

- We can also inherit from a derived class. This is called multilevel inheritance. It can be of any depth in Python.
- In multilevel inheritance, features of the base class and the derived class are inherited into the new derived class.

[59]:
```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

4 Polymorphism

- The word polymorphism means having many forms.
- In programming, polymorphism means same function name being uses for different types.

[60]:
```
# Example of inbuilt polymorphic function
len("HI")
```

[60]:
```
2
```

4.1 Polymorphism with class methods

- We can use the concept of polymorphism while creating class methods as Python allows different classes to have methods with the same name.
- We can then later generalize calling these methods by disregarding the object we are working with.

[61]:
```
class Cat:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a cat. My name is {self.name}. I am {self.age} years old.")
    def make_sound(self):
        print("Meow")
```

[62]:
```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age

    def info(self):
        print(f"I am a dog. My name is {self.name}. I am {self.age} years old.")

    def make_sound(self):
        print("Bark")
```

[63]:
```
cat_1 = Cat("Kitty", 2.5)
dog_1 = Dog("Fluffy", 4)
```

[64]:
```
for animal in (cat_1, dog_1):
    animal.make_sound()
    animal.info()
    print()
```

```
Meow
I am a cat. My name is Kitty. I am 2.5 years old.

Bark
I am a dog. My name is Fluffy. I am 4 years old.
```

4.2 Polymorphism with Inheritance

- Polymorphism allows us to access these overridden methods and attributes that have the same name as the parent class.

[65]:
```
class Shape:
    def __init__(self, name):
        self.name = name

    def area(self):
        pass

    def fact(self):
        return "I am a two-dimensional shape."
```

[66]:
```
class Square(Shape):
    def __init__(self, length):
        super().__init__("Square")
        self.length = length

    def area(self):
        return self.length**2

    def fact(self):
        return "Squares have each angle equal to 90 degrees."
```

[67]:
```
class Circle(Shape):
    def __init__(self, radius):
        super().__init__("Circle")
        self.radius = radius

    def area(self):
        return 3.14*self.radius**2
```

[68]:
```
a = Square(4)
b = Circle(7)
```

[69]:
```
b.fact()
```

[69]:
```
'I am a two-dimensional shape.'
```

[70]:
```
a.fact()
```

[70]:
```
'Squares have each angle equal to 90 degrees.'
```

[71]:
```
b.area()
```

[71]:
```
153.86
```

[72]:
```
a.area()
```

[72]:
```
16
```