Python Classes and Objects

TUYEN NGOC LE

Outline

- **Object Oriented Programming**
- **Python Classes**
- **Python Objects**

Object Oriented Programming

Python is a programming language that supports various programming styles, including object - oriented programming(OOP) through the use of objects and classes.
An object is simply a collection of data (variables) and methods (functions). For example, a parrot is an object.It has
Attributes (variables) - name, age, color, etc.
Behavior (functions) - dancing, singing, etc.
Similarly, a class is a blueprint for that object.

Python Classes

A class is considered as a blueprint of objects. We can think of the class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows, etc. Based on these descriptions we build the house. House is the object. Since many houses can be made from the same description, we can create many objects from a class.

Define Python Class

We use the class keyword to create a class in Python. For example:

```
# create a class
class Room:
    length = 0.0  # attributes
    breadth = 0.0 # attributes
```

```
# create a class
class student:
    name = ""         # attributes
    student_id = 0 # attributes
```

Python Objects

An object is called an instance of a class. Here's the syntax to create an object

```
objectName = ClassName()
```

```
# create a class
class student:
    name = ""         # attributes
    student_id = 0 # attributes
# create objects
student_1 = student()
student_2 = student()
```

Access Class Attributes Using Objects

We use the "." notation to access the attributes of a class. For example,

```
# create a class
class student:
    name = ""  # attributes
    student_id = 0 # attributes
# create objects
student_1 = student()
student_2 = student()
# access attributes using student_1
student_1.name = "Le Ngoc Tuyen"
student_1.student_id = 100
print("student_1' name: ", student_1.name)
print("student_1' id: ", student_1.student_id)
# access attributes using student_2
student_2.name = "Wang Mei Hua"
student_2.student_id = 101
print("student_2' name: ", student_2.name)
print("student_2' id: ", student_2.student_id)
```

Python Methods

We can also define a function inside a Python class. A function defined inside a class is called a method.

```
# create a class
class Room:
    length = 0.0    #Attributes
    breadth = 0.0  #Attributes

    # method to calculate area
    def calculate_area(self):  # Method
        print("Area of Room =", self.length * self.breadth)

# create object of Room class
study_room = Room()

# assign values to all the attributes
study_room.length = 42.5
study_room.breadth = 30.8

# access method inside class
study_room.calculate_area()
```

The __init__() Function

To understand the meaning of classes we have to understand the built-in __init__() function.
All classes have a function called __init__(), which is always executed when the class is being initiated.
Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created.
The __init__() function is called automatically every time the class is being used to create a new object.

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

p1 = Person("John", 36)

print(p1.name)
print(p1.age)
```

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.

It does not have to be named self , you can call it whatever you like, but it has to be the first parameter of any function in the class:

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

  def myfunc(abc):
    print("Hello! My name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

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

  def myfunc(abc):
    print("Hello! My name is " + abc.name)

p1 = Person("John", 36)
p1.myfunc()
```

Modify Object Properties

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myname(obj):
    print("Hello! My name is " + obj.name)
  def myage(obj):
    print("My age is " + str(obj.age))

p1 = Person("John", 36)
p1.myname()
p1.myage()
p1.age = 40
p1.myage()
```

Delete Object Properties

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myname(obj):
    print("Hello! My name is " + obj.name)
  def myage(obj):
    print("My age is " + str(obj.age))

p1 = Person("John", 36)
p1.myname()
p1.myage()
del p1.age
p2 = Person("Lee", 50)
p2.myname()
p2.myage()
```

```
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age
  def myname(obj):
    print("Hello! My name is " + obj.name)
  def myage(obj):
    print("My age is " + str(obj.age))

p1 = Person("John", 36)
p1.myname()
p1.myage()
del p1.age
p2 = Person("Lee", 50)
p2.myname()
p2.myage()
p1.myage()
```

→ `AttributeError: 'Person' object has no attribute 'age'`

The pass Statement

`class` definitions cannot be empty, but if you for some reason have a `class` definition with no content, put in the `pass` statement to avoid getting an error.

```
class Person:
```



```
class Person: ^ IndentationError: expected an indented block
```

```
class Person:
  pass
```

Python Inheritance

Inheritance is a way of creating a new class for using details of an existing class without modifying it.
The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

```
# base class
class Animal:
    def eat(self):
        print( "I can eat!")
    def sleep(self):
        print("I can sleep!")
# derived class
class Dog(Animal):
    def bark(self):
        print("I can bark! Woof woof!!")
# Create object of the Dog class
dog1 = Dog()
# Calling members of the base class
dog1.eat()
dog1.sleep()
# Calling member of the derived class
dog1.bark();
```

Python Encapsulation

Encapsulation is one of the key features of object-oriented programming. Encapsulation refers to the bundling of attributes and methods inside a single class.

It prevents outer classes from accessing and changing attributes and methods of a class. This also helps to achieve data hiding.

In Python, we denote private attributes using underscore as the prefix i.e single _ or double __

```
class Computer:
    def __init__(self):
        self.__maxprice = 900 # private variable
    def sell(self):
        print("Selling Price: {}".format(self.__maxprice))
    def setMaxPrice(self, price):
        self.__maxprice = price
c = Computer()
c.sell()
# change the price
c.__maxprice = 1000
# __maxprice is a private variable, this modification is not seen on the output
c.sell()
# using setter function to change the maxprice
c.setMaxPrice(1000)
c.sell()
```

Polymorphism

Polymorphism is another important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

In the above example, we have created a superclass: Polygon and two subclasses: Square and Circle. Notice the use of the render() method.
The main purpose of the render() method is to render the shape.
However, the process of rendering a square is different from the process of rendering a circle.
Hence, the render() method behaves differently in different classes. Or, we can say render() is **polymorphic**.

```
class Polygon:
    # method to render a shape
    def render(self):
        print("Rendering Polygon...")
class Square(Polygon):
    # renders Square
    def render(self):
        print("Rendering Square...")
class Circle(Polygon):
    # renders circle
    def render(self):
        print("Rendering Circle...")
# create an object of Square
s1 = Square()
s1.render()
# create an object of Circle
c1 = Circle()
c1.render()
```

Key Points to Remember

- Object-Oriented Programming makes the program easy to understand as well as efficient.
- Since the class is sharable, the code can be reused.
- Data is safe and secure with data abstraction.
- Polymorphism allows the same interface for different objects, so programmers can write efficient code.

Exercise

```
class TrafficLight:
    '''This is an updated traffic light class'''
    # Class variable
    traffic_light_address = 'Ming Chi'
    def __init__(self, color):
        # Instance variable assigned inside the class constructor
        self.color = color
    def action(self):
        if self.color=='red':
            # Instance variable assigned inside a class method
            self.next_color = 'yellow'
            print('Stop & wait')
        elif self.color=='yellow':
            self.next_color = 'green'
            print('Prepare to stop')
        elif self.color=='green':
            self.next_color = 'red'
            print('Go')
        else:
            self.next_color = 'Brandy'
            print('Stop drinking 😉')
```

```
# Creating class objects
for c in ['red', 'yellow', 'green', 'fuchsia']:
    c = TrafficLight(c)
    print(c.traffic_light_address)
    print(c.color)
    c.action()
    print(c.next_color)
    print('\n')
```

Exercise

1. Write a Python class Employee with attributes like emp_id, emp_name, emp_salary, and emp_department and methods like calculate_emp_salary, emp_assign_department, and print_employee_details.
Sample Employee Data:
"ADAMS", "E7876", 50000, "ACCOUNTING"
"JONES", "E7499", 45000, "RESEARCH"
"MARTIN", "E7900", 50000, "SALES"
"SMITH", "E7698", 55000, "OPERATIONS"

•Use 'assign_department' method to change the department of an employee.
•Use 'print_employee_details' method to print the details of an employee.
•Use 'calculate_emp_salary' method takes two arguments: salary and hours_worked, which is the number of hours worked by the employee. If the number of hours worked is more than 50, the method computes overtime and adds it to the salary. Overtime is calculated as following formula:
overtime = hours_worked - 50
Overtime amount = (overtime * (salary / 50))

https://www.w3resource.com/python-exercises/class-exercises/python-class-real-life-problem-1.php

Exercise

2. Write a Python class Restaurant with attributes like menu_items, book_table, and customer_orders, and methods like add_item_to_menu, book_tables, and customer_order.

Perform the following tasks now:

Now add items to the menu.

- ❑ Make table reservations.
- ❑ Take customer orders.
- ❑ Print the menu.
- ❑ Print table reservations.
- ❑ Print customer orders.

Note: Use dictionaries and lists to store the data

https://www.w3resource.com/python-exercises/class-exercises/python-class-real-life-problem-2.php

Exercise

3. Write a Python class BankAccount with attributes like account_number, balance, date_of_opening and customer_name, and methods like deposit, withdraw, and check_balance

https://www.w3resource.com/python-exercises/class-exercises/python-class-real-life-problem-3.php

Exercise

4. Write a Python class Inventory with attributes like item_id, item_name, stock_count, and price, and methods like add_item, update_item, and check_item_details.

Use a dictionary to store the item details, where the key is the item_id and the value is a dictionary containing the item_name, stock_count, and price.

https://www.w3resource.com/python-exercises/class-exercises/python-class-real-life-problem-4.php

References

- https://www.programiz.com/python-programming/class
- https://www.w3schools.com/python/python_classes.asp
- https://www.dataquest.io/blog/using-classes-in-python/
- https://www.w3resource.com/python-exercises/class-exercises/#application