10. Object Oriented Programming

《Python programming》 / Lecturer : Zhiyi Luo (罗志一)

School of Computer Science and Technology
计算机科学与技术学院

浙江理工大学
ZHEJIANG SCI-TECH UNIVERSITY

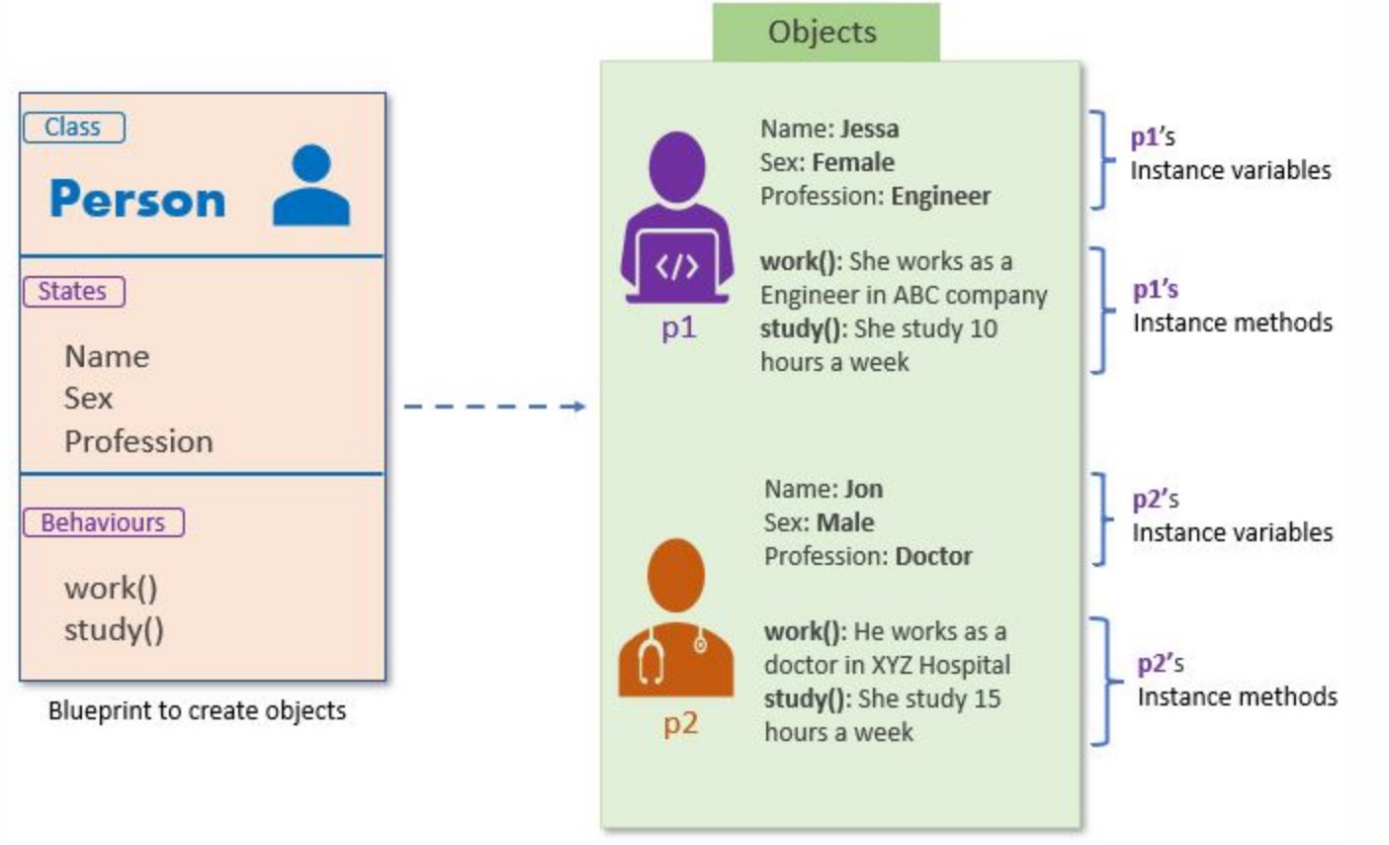
Object-oriented Programming

- **Objected-oriented programming (OOP**) is a programming paradigm based on the concept of **“objects”.** The object contains both data and code; Data in the form of properties (often known as attributes), and code, in the form of methods (actions object can perform).
- An object-oriented paradigm is to design the program using classes and objects. An object has the following two characteristics:
  - Attribute
  - Behavior
- One important aspect of OOP in Python is to create reusable code using the concept of inheritance. This concept is also known as DRY (Don’t Repeat Yourself).

Class and Objects

- Class: The class is a user-defined data structure that binds the data members and methods into a single unit. Class is a blueprint or code template for object creation. Using a class, you can create as many objects as you want.
- Object: An object is an instance of a class. It is a collection of attributes (variable) and methods. We use the object of a class to perform actions.
- Objects have two characteristics: They have states and behaviors (object has attributes and methods attached to it) Attributes represent its state, and methods represent its behavior. Using its methods, we can modify its state.

Class and Objects



Create a Class

- In Python, class is defined by using the class keyword. The syntax to create a class is given below.

```
class class_name:
    <statement 1>
    <statement 2>
    .
    .
    <statement N>
```

class_name: It is the name of the class
statements: Attributes and methods

Define a class

```
class Person:
    def __init__(self, name, sex, profession):
        # data members (instance variables)
        self.name = name
        self.sex = sex
        self.profession = profession

    # Behavior (instance methods)
    def show(self):
        print('Name:', self.name, 'Sex:', self.sex, 'Profession:', self.profession)

    # Behavior (instance methods)
    def work(self):
        print(self.name, 'working as a', self.profession)
```

Class Attributes

- In Class, attributes can be defined into two parts
  - Instance variables: The instance variables are attributes attached to an instance of a class. We define instance variables in the constructor (the __init__() method of a class).
  - Class variables: A class variable is a variable that is declared inside of class, but outside of any instance method or __init__() method.

Accessing properties and assigning values

- An instance attribute can be accessed or modified by using the dot notation: instance_name.attribute_name
- A class variable is accessed or modified using the class name

Create a object

- We can create any number of objects of a class. Use the following syntax to create an object of a class.
  - reference_variable = classname()

```
class Employee:
    # class variables
    company_name = 'ABC Company'
    # constructor to initialize the object
    def __init__(self, name, salary):
        # instance variables
        self.name = name
        self.salary = salary
    # instance method
    def show(self):
        print('Employee:', self.name, self.salary, self.company_name)

# create first object
emp1 = Employee("Harry", 12000)
emp1.show()

# create second object
emp2 = Employee("Emma", 10000)
emp2.show()
```

Example Explanation

- In the above example, we create a Class with the name Employee.
- Next, we defined two attributes name and salary.
- Next, in the __init__() method, we initialized the value of attributes. This method is called as soon as the object is created. The init method initializes the object.
- Finally, from the Employee class, we created two objects, Emma and Harry.
- Using the object, we can access and modify its attributes.

```
class Employee:
    # class variables
    company_name = 'ABC Company'
    # constructor to initialize the object
    def __init__(self, name, salary):
        # instance variables
        self.name = name
        self.salary = salary
    # instance method
    def show(self):
        print('Employee:', self.name, self.salary, self.company_name)

# create first object
emp1 = Employee("Harry", 12000)
emp1.show()

# create second object
emp2 = Employee("Emma", 10000)
emp2.show()
```

constructor

Parameters to constructor

Instance variable

Instance method

Object of class

Constructors in Python

- In Python, a constructor is a special type of method used to initialize the object of a Class. The constructor will be executed automatically when the object is created. If we create three objects, the constructor is called three times and initialize each object.
- The main purpose of the constructor is to declare and initialize instance variables. It can take at least one argument that is self. The __init__() method is called the constructor in Python.
- A constructor is optional, and if we do not provide any constructor, then Python provides the default constructor. Every class in Python has a constructor, but it’s not required to define it.

Inheritance in Python

- In an Object-oriented programming language, inheritance is an important aspect. In Python, inheritance is the process of inheriting the properties of the parent class into a child class.
- The primary purpose of inheritance is the reusability of code. Using inheritance, we can use the existing class to create a new class instead of recreating it from scratch.

```
class BaseClass:
  Body of base class
class DerivedClass(BaseClass):
  Body of derived class
```

Use of Inheritance

- In the below example, from a vehicle class, we are creating a Car class. We don’t need to define common attributes and methods again in Car class. We only need to add those attributes and methods which are specific to the Car.
- In heritance, the child class acquires all the data members, properties, and functions of the parent class. Also, a child class can customize any of the parent class methods.

Use of Inheritance

```
# Base class
class Vehicle:
    def __init__(self, name, color, price):
        self.name = name
        self.color = color
        self.price = price
    def info(self):
        print(self.name, self.color, self.price)

# Child class
class Car(Vehicle):
    def change_gear(self, no):
        print(self.name, 'change gear to number', no)

# Create object of Car
car = Car('BMW X1', 'Black', 35000)
car.info()
car.change_gear(5)
```