Object Oriented Programming

Python Open Lab

Real-World Object

- We have many objects in the real world, like car, dog, people…

- So far, we try to put some real-world task into our program, like calculating the salary of workers.

- Can we put the real-world objects into our program?

Real World

- People, Dog, Car
- People pet dog; People drive car
- People have age, height, weight
- Car have price
- Dog have gender

Object in program

- We can not extract all information of read-world object into the program.

- We can simplify this question: Just pay attention to the attributes of a object and what the object can do, that's all we need!

- For people, attributes include age, weight, height, wealth, …, what people can do include walk(), sing(), drive(), run() …

Object Oriented Design

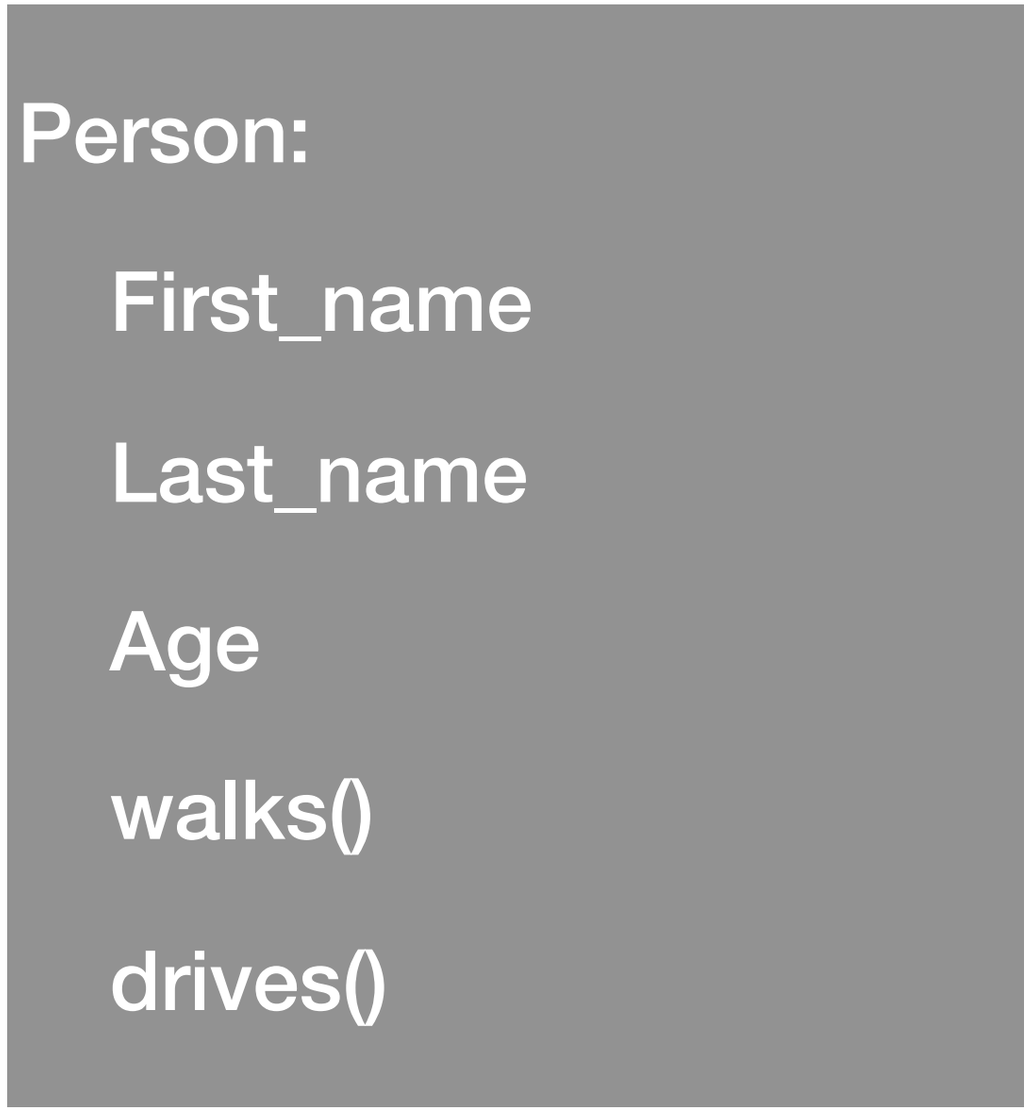
- A program is made of multiple objects.
- Objects interact with each other(in the program)
- We can see everything as an object.
- Our program contains dictionary, string, list, Integer, Person,…, other Objects. They are all objects! They interact with each other to get the result of output.

Terminology

- Class - description of object
    - Template of objects
- Instance - actual object that belongs to some class
    - They are concrete items, rather than abstract descriptions.
- Method - abilities of an object
    - Manipulate data it contains
    - Interact with other objects
- Attribute - individual characteristics/data with an object/class

Example

- An object: person
- Attributes:
  - First_name, Last_name, Age
  - Method: walks(), drives()



Class and instance

- We declare a class here
- Attribute: age
- Method: birthday
- Person

```
class Person:

    age = 20

    def birthday(self):

        self.age = self.age + 1

        print(self.age)
```

Instance

- So far, we have not met an instance.
- The idea of class and instance in object is similar to the idea in the functions.
- We define a class and if we do not create instance of this class, this class is never used. (very similar to function)

```
James = Person()
James.birthday()
```

Constructor

- A special method _ _init_ _() to specify the attribute values and actions at time of instance creation.

- This method is called only once at the time of object creation.

- Automatic call - no need to explicitly call(this means programmers do no need to call it, the program will call it automatically, the programmers just need to define it)

Constructor

```
class Person:
    age = 0 # we will change it
    def _ _ init_ _ (self, age):
        self.age = age
    def birthday(self):
        self.age = self.age + 1
        print(self.age)
```

```
James = Person(20)
James.birthday()
```

The constructor means we change the age of person when we build an instance of the person

Constructor

```
class Person:
    age = 0 # we will change it
    last_name = None
    first_name = None
    def _ _ init_ _ (self, age, last, first):
        self.age = age
        self.last_name = last
        self.first_name = first
    def birthday(self):
        self.age = self.age + 1
        print(self.age)
    def showName(self):
        name = last_name + “, “ + first_name
        print(name)
```

Constructor

```
class Person:
    age = 0 # we will change it
    last_name = None
    first_name = None
    def _ _ init_ _ (self, age, last, first):
        self.age = age
        self.last_name = last
        self.first_name = first
    def birthday(self):
        self.age = self.age + 1
        print(self.age)
    def showName(self):
        name = self.last_name + “, “ + self.first_name
        print(name)
```

Constructor

```
James = Person(20, “James”, “Michael”)

James.birthday()
James.showName()
```

Exercise

- Add ‘wealth’ to this person, and when we initiate this person, his/her wealth is 0.

- Add method increaseWealth(self, number), so when this method is called, the wealth of this person increases by the value of number.

- Add method decreaseWealth(self, number), so when this method is called, the wealth of this person decreases by the value of number.

Object in object

- Object can contain object

```
class ClassRoom:
    teacher = None
    classRoomName = ""
    def __init__(self, name):
        self.classRoomName = name #"DSSC207"
    def addTeacher(self, teacher):
        self.teacher = teacher
    def showTeacher(self):
        print(self.teacher.last_name)
    def showRoomName(self):
        print(self.classRoomName)
```

Object in object

- Exercise
  - Initialize the classroom name with “DSSC207”
  - Put a teacher(Person) whose age is 20, first name is “Michael”, last name is “Su” to the classroom.
  - Call the method of showTeacher() to see what you get.

Reference

- Slide of Kunal Baweja(Chapter 13, OOP)