

# Introduction to Object Oriented Programming in Python

Kevin Kam Fung Yuen, PhD,  
kevinkf.yuen@gmail.com  
16 Jan 2020

## Prerequisites

For this class, you should

- install *Python* and *Jupyter*;
- optionally install *Visual Studio Code* for the IDE;
- understand some basic syntax.

You may refer to my previous teaching materials to assist you to understand more.

- KKF Yuen, ANL251 Python Programming, <https://github.com/kkfyuen/ANL251Python> (<https://github.com/kkfyuen/ANL251Python>)

For more references, there are a lot of readings in the Python official websites.

- <https://www.python.org/doc/> (<https://www.python.org/doc/>)

## Create Your First OO Class

### Basic concepts of a class in OOP

A *class* in OO programming language is the code that defines a particular group of objects with the attributes and methods specified.

Attributes are variables which are used to store values of data.

Methods are functions to demonstrate the actions or behaviors in this class.



Fig 1: Class Diagram

### Understand the Code by Example

The python code of very basic level for a simple class for a cat may be defined as below.

```
In [ ]: class Cat:
        def whoamI(self):
            return {"name":self.name, "gender":self.gender, "age":self.age}
```

- *self* refers to the object itself created by this class. *self* is required in every method in the class. *self* in Python is similar to *this* in Java or C++.
- *def* defines a function called *whoamI*. In OOP, a function in a class is called a method or an operation.
- The cat class above only defines a method.

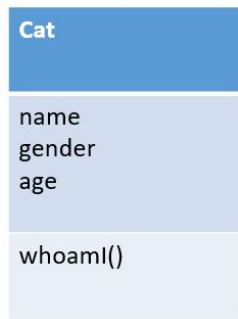


Fig 2: Cat class diagram

## Instantiate objects from the class

An *object* is an instance created from a *class*. This process is called *Instantiation*.

In other words, a *class* is the blueprint and an *object* is an implementation from the blueprint.

Many objects can be *instantiated* from one class with assigning their own values to attributes and/or methods to be unique individuals.

For example, two cat objects can be created from a cat class.

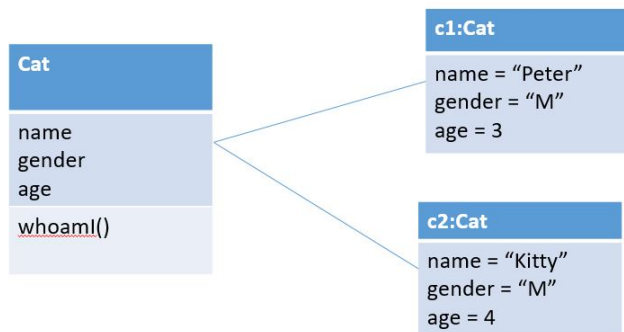


Fig 3: Various cat objects instantiated from the Cat class

### Cat object 1

```
In [ ]: c1 = Cat() # constructor with the same name as the class name.
c1.name = "Peter"
c1.gender = "M"
c1.age = 3
```

### Cat object 2

```
In [ ]: c2 = Cat()
c2.name = "Kitty"
c2.gender = "F"
c2.age = 4
```

## Access objects

Access objects by calling the method from an object.

```
In [ ]: print(c1.whoamI())
```

```
In [ ]: c2.whoamI()
```

## Question 1

A student created a code as below. Comment the result. Could you suggest any improvement for the code?

```
In [ ]: c3 = Cat()
        c3.Name = "Nic"
        c3.gender = "M"
        c3.age = 2
```

```
In [ ]: c3.whoamI()
```

## Initializer ( init )

For the basic case above, each time we define an object, we have to repeat the same code. During such process, some errors may be occurred due to careless typing. To address this issue, we introduce *initializer* for the constructor.

## Create a Cat class with initializer

The *initializer* method is automatically executed when an instance of the class is created. Normally it is put at the beginning of the method.

```
In [ ]: class Cat:

        # Initializer for attributes of an instance
        def __init__(self, name, gender,age):
            self.name = name
            self.gender = gender
            self.age = age

        def whoamI(self):
            return {"name":self.name,"gender":self.gender,"age":self.age}
```

## Instantiate cat objects by passing values to constructor

Pass the values to the constructor according to the specified format.

```
In [ ]: cat1 = Cat("Peter", "M",5)
        cat2 = Cat("Kitty", "F",4)
        cat3 = Cat("Nic", "M",2)
```

```
In [ ]: cat1
```

```
In [ ]: cat1.whoamI()
```

## Access the objects

Calculate the average age of cats

```
In [ ]: (cat1.whoamI()["age"] + cat2.whoamI()["age"] + cat3.whoamI()["age"]) /3
```

## Question 2

What is the expected result for the code below? Comment the result.

```
In [ ]: cata = Cat("Peter", "M",5)
        catb = Cat("Peter", "M",5)
        cata == catb
```

```
In [ ]: cata
```

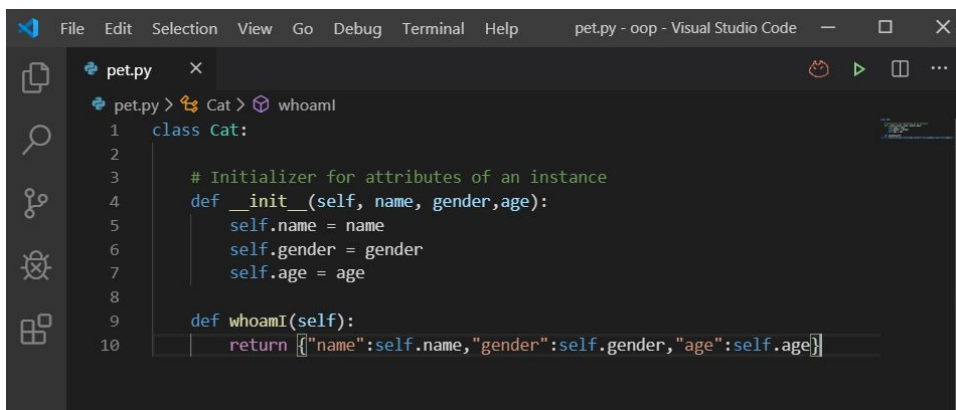
```
In [ ]: catb
```

## Storing class in module

Normally, we do not put the codes for class, main program or execution together. We need to separate them. The steps are demonstrated as below.

### Create a module

A module is the .py file. We create a file called *pet.py* in the same folder of your working file. Next we copy the code for the *Cat* class in the file. A demo by visual studio code is presented as below.

A screenshot of the Visual Studio Code editor window. The title bar says 'pet.py - oop - Visual Studio Code'. The editor shows a Python file named 'pet.py' with the following code:

```
1 class Cat:
2
3     # Initializer for attributes of an instance
4     def __init__(self, name, gender, age):
5         self.name = name
6         self.gender = gender
7         self.age = age
8
9     def whoamI(self):
10        return [{"name":self.name,"gender":self.gender,"age":self.age}]
```

Fig 4: code in *pet.py*

### Use class in the module

```
In [ ]: from pet0 import Cat    # from module import class
```

```
In [ ]: catPC = Cat("Peter", "M", 5)
catPC
```

```
In [ ]: isinstance(catPC, Cat)    #isinstance(object, class)
```

### Another way

```
In [ ]: import pet0 as pt
```

```
In [ ]: catm = pt.Cat("Peter", "M", 5)
catm
```

## Question 3

Show the steps.

- Define and add a Dog class in the pet.py file.
- Create several dog objects.
- Access the objects.

```
In [ ]: from pet1 import Dog
dog1 = Dog("snoopy", "M", 4.5)
dog1.whoamI()
```

## Inheritance

We found that dog and cat classes are shared with same attributes and methods. In other words, the codes are duplicated. **Inheritance** in OOP is the good approach to make the code simpler and more manageable.

**Inheritance** in OOP is used to create an “**is a**” relationship between classes. For example, a cat is a pet; a dog is a pet. The pet is a parent class. The cat and dog are the children class. Fig 5 shows the inheritance diagram.

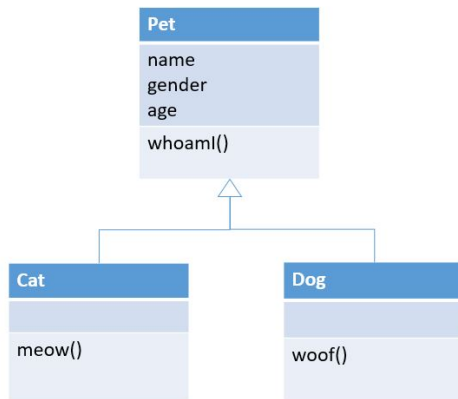


Fig 5: Inheritance diagram

## Code

For the syntax, we use the code for **inheritance**: *childClassName(parentClassName)*

```
class Pet:
    # Initializer for attributes of an instance
    def __init__(self, name, gender, age):
        self.name = name
        self.gender = gender
        self.age = age

    def whoamI(self):
        return {"name":self.name, "gender":self.gender, "age":self.age}

class Dog(Pet):
    def woof(self):
        return("woof")

class Cat(Pet):
    def meow(self):
        return("meow")
```

Fig 5: Inheritance code for pet class

## Test dog object

```
In [ ]: from pet2 import Dog
dog2 = Dog("snoopy", "M", 4.5)
dog2.whoamI()
```

```
In [ ]: dog2.woof()  # behaviour for dog
```

## Test cat object

```
In [ ]: from pet2 import Cat
cat2 = Cat("Kitty", "F", 3.5)
cat2.whoamI()
```

```
In [ ]: cat2.meow()  # behaviour for cat
```

## Question 4

Using the codes in the section above, execute and comment the codes below.

```
In [ ]: from pet2 import Pet
```

```
In [ ]: isinstance(cat2, Cat)
```

```
In [ ]: from pet2 import Pet
isinstance(cat2, Pet)
```

```
In [ ]: isinstance(cat2, Dog)
```

```
In [ ]: isinstance(dog2, Dog)
```