ADVANCED PYTHON PROGRAMMING

Introduction to Object-Oriented Programing

Peter Lo

Object Oriented Programming

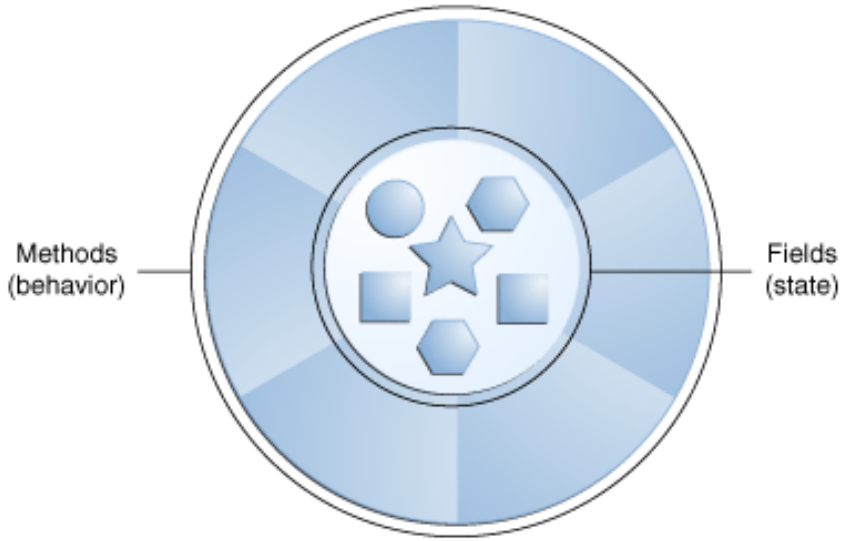
Object, Class and Instance

Object

- Real-world objects share two characteristics: They all have state and behavior.
  - E.g. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail).
- Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Software Object

- Software objects are conceptually similar to real-world objects, they also consist of state and related behavior.
- An object stores its state in fields (variables in some programming languages) and exposes its behavior through methods (functions in some programming languages).



Object-Oriented Programming

- Object-oriented Programming (OOP) is a programming paradigm which provides a means of structuring programs so that properties and behaviors are bundled into individual objects.
- Object-oriented programming models real-world entities as software objects, which have some data associated with them and can perform certain functions.
- The key takeaway is that objects are at the center of the object-oriented programming paradigm, not only representing the data, as in procedural programming, but in the overall structure of the program as well.

Class

- A class is a blueprint or template or set of instructions to build a specific type of object.
- Basically, a class will consists of field, static field, method, static method and constructor.
  - Field is used to hold the state of the class (eg: name of Student object).
  - Method is used to represent the behavior of the class (eg: how a Student object going to stand-up).
  - Constructor is used to create a new Instance of the Class.

Instance

- An instance is a specific object built from a specific class.
- It is assigned to a reference variable that is used to access all of the instance's properties and methods.
  - For instance, it is a unique copy of a Class that representing an Object.
- When a new instance of a class is created, a room of memory is allocated for that class instance.

Objects and Classes

- Languages that support object-oriented programming typically use inheritance for code reuse and extensibility in the form of either classes or prototypes.
- Those that use classes support two main concepts:
  - Classes – the definitions for the data format and available procedures for a given type or class of object
  - Objects – instances of classes

Defining a Class

- To define a class, in typical simple Python fashion, we use the word **class**, followed by the name of your new class.
- We use a colon after the name, and then anything contained within the class definition is indented.
- There is no parentheses with a class

```
class pet:
```

Define a Property for Class (Indentation)

- In order to give the class a couple of properties, simply define some variables inside the class.
- A value also need to defined for each variable.

```
class pet:
    Number_of_Leg = 0
```

it doesn't matter in this case since the number of legs will be specific to each instance of the class - a fish doesn't have the same amount of legs as a dog or a duck, etc. The value need to change for each object anyway

Create an Instance (Object) for Class

- A class on its own isn't something you can directly manipulate; first, we have to create an instance of the class to play with. We can store that instance in a variable.
- Outside of the class (without any indentation), let's make an instance of the class and store it in the variable.
- To make a new instance of a class, you simply type the name of the class, and then a pair of parentheses.
- The parentheses is a way of passing in a variable for use by the class when you first create the instance.

```
class pet:
    Number_of_Leg = 0

Dog = pet()
print(type(Dog))
```

A class on its own isn't something that you can directly manipulate.

```
<class '__main__.pet'>
```

Access to Property of Object

- To reference a property of an object, first we have to tell Python which object (or which instance of a class) we're talking about. Then we're going to write a period to indicate that we're referencing something that's contained within our instance. After the period, we add the name of our variable.

```
class pet:
    Number_of_Leg = 0

Dog = pet()
Dog.Number_of_Leg = 4
print (Dog.Number_of_Leg)
```

```
4
```

Create Method

- Methods are functions contained within a class.
- You define one in exactly the same way as you would a function, but the difference is that you put it inside a class, and it belongs to that class.
- If you ever want to call that method, you have to reference an object of that class first, just like the variables.

```
class pet:
    Number_of_Leg = 0

    def Sleep(self):
        print ("zzz")

Dog = pet()
Dog.Sleep()
```

Create Method with Data

- When we were accessing variables from outside the class, we have to reference it by first specifying the instance containing that variable.
- 'self' is just a reference to the object that is currently being manipulated. So to access a variable in the current class, you simply need to preface it with 'self' and then a period

```
class pet:
    Number_of_Leg = 0

    def Sleep(self):
        print ("zzz")

    def Count_Legs(self):
        print ("Number of legs =", self.Number_of_Leg)

Dog = pet()
Dog.Number_of_Leg = 4
Dog.Count_Legs()
```

```
Number of legs = 4
```

Instance Attributes

- All classes create objects, and all objects contain characteristics called attributes (referred to as properties in the opening paragraph).
- Use the **__init__()** method to initialize (e.g., specify) an object's initial attributes by giving them their default value (or state).
- This method must have at least one argument as well as the self variable, which refers to the object itself.

```
class Dog:
    # Initializer / Instance Attributes
    def __init__(self, Name, Age):
        self.Name = Name
        self.Age = Age
```

Class Attributes

- While instance attributes are specific to each object, class attributes are the same for all instances.

```
class Dog:
    # Class Attribute
    Species = 'Mammal'

    # Initializer / Instance Attributes
    def __init__(self, Name, Age):
        self.Name = Name
        self.Age = Age
```

Instantiating Objects

- Instantiating is a fancy term for creating a new, unique instance of a class.

```
class Dog:
    # Class Attribute
    Species = 'Mammal'

    # Initializer / Instance Attributes
    def __init__(self, Name, Age):
        self.Name = Name
        self.Age = Age

Chihuahua = Dog("Lucky", 5)
Pug = Dog("Lucky", 5)

Chihuahua == Pug
```

```
False
```

Instance Methods

- Instance methods are defined inside a class and are used to get the contents of an instance.
- They can also be used to perform operations with the attributes of our objects. Like the **__init__** method, the first argument is always self:

```
class Dog:
    # Class Attribute
    Species = 'Mammal'

    # Initializer / Instance Attributes
    def __init__(self, Name, Age):
        self.Name = Name
        self.Age = Age

    # instance method
    def Description(self):
        return "{} is {} years old".format(self.Name, self.Age)

    # instance method
    def Speak(self, Sound):
        return "{} says {}".format(self.Name, Sound)


# Instantiate the Dog object
Chihuahua = Dog("Lucky", 5)

# call our instance methods
print(Chihuahua.Description())
print(Chihuahua.Speak("Woo Woo"))
```

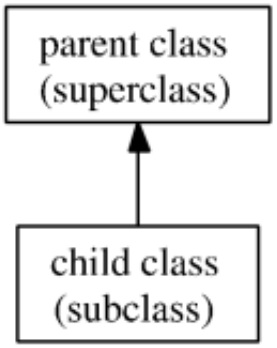
```
Lucky is 5 years old
Lucky says Woo Woo
```

Destructor

- Destructors are called when an object gets destroyed. In Python, destructors are not needed as much needed in C++ because Python has a garbage collector that handles memory management automatically.
- The ***__del__()*** method is a known as a destructor method in Python. It is called when all references to the object have been deleted i.e when an object is garbage collected.

Child Class

- Inheritance is the process by which one class takes on the attributes and methods of another. Newly formed classes are called Child Classes, and the classes that child classes are derived from are called Parent Classes.
- Child classes inherit all of the parent's attributes and behaviors but can also specify different behavior to follow.
- The most basic type of class is an object, which generally all other classes inherit as their parent



Example

```
# Parent class
class Dog:
    # Class Attribute
    Species = 'Mammal'

    # Initializer / Instance Attributes
    def __init__(self, Name, Age):
        self.Name = Name
        self.Age = Age

    # instance method
    def Description(self):
        return "{} is {} years old".format(self.Name, self.Age)

    # instance method
    def Speak(self, Sound):
        return "{} says {}".format(self.Name, Sound)

# Child class (inherits from Dog() class)
class Chihuahua(Dog):
    def Run(self, Speed):
        return "{} runs {}".format(self.Name, Speed)

# Child classes inherit attributes and
# behaviors from the parent class
Lucky = Chihuahua("Lucky", 5)
print(Lucky.Description())
print(Lucky.Run("Faster"))
```

```
Lucky is 5 years old
Lucky runs Faster
```