Learning Object-oriented Programming with Python

B.C.D.

Learning Object-oriented Programming with Python

B.C.D.

Tweet This Book!

Please help B.C.D. by spreading the word about this book on Twitter!

The suggested hashtag for this book is #OOPwithPython.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

https://twitter.com/search/#OOPwithPython

Contents

Chapter One: Introduction

Object-oriented Programming(OOP) and Python are very close to each other. The purpose of this book is to explain the principles of OOP with examples from the Python language. At the end of this book, you'll have a much clear idea of OOP and how Python uses it to help you solve many problems. You'll also have some fundamental idea of how to write programs using Python.

Going on ahead, I'm going to assume few things -

1. You already have some idea about programming and want to learn more about it
2. You have idea about data types, variables, functions and basic data structures like array etc
3. You have a computer in front of you so that you can try the examples out
4. You have Python installed in your computer
5. You have a Python interpreter(e.g. IDLE, bpython, ipython etc) installed in your computer

Number 1 is compulsory. If it's not true, you might as well close this book. This book is solely for people who want to learn more. If 1 is true, then 2 should be true. The rest are upto you but I recommend that you try things hands on.

My Setup

- I will be using Python 2.7.4 throughout the book for the examples.
- I'll be using `bpython` to interpret the examples.

What is Object-oriented Programming(OOP)?

Here is the definition of OOP as noted in Wikipedia -

> Object-oriented programming (OOP) is a programming paradigm that represents concepts as "objects" that have data fields (attributes that describe the object) and associated procedures known as methods. Objects, which are usually instances of classes, are used to interact with one another to design applications and computer programs.

What it essentially means is that, OOP is a fundamental style of programming. There are three other styles: imperative, functional, and logic programming. They all have their merits and demerits. But that's a whole different topic. In this book, we'll concentrate totally on the object-oriented paradigm of programming.

OOP depends on two basic concepts: Class and Object. As mentioned in the definition, objects are just instances of one or more class and they have their own features. When you're programming following these two concepts, it's called Object-oriented Programming. We'll revisit these concepts in next chapters.

What is Python exactly?

As you might know Python is a programming language. More precisely -

> It is an interactive, object-oriented, extensible programming language. It provides an extraordinary combination of clarity and versatility, and is free, open-source, and comprehensively ported.

I know that brings more question in mind. To explain it clearly, Python is a high-level general-purpose programming language. It's called `general-purpose` because Python can be used to build various kinds of softwares, to solve (mostly) any kind of problem one might have. It's also really easy to learn. Printing hello world is as easy as -

```
>>> print "Hello World"
Hello World
```

Reading a file is as easy as -

```
>>> file = open("test.txt", "r")
>>> file.readlines()
['Hello world\n', 'How are you?\n']
```

Python follows object-oriented programming paradigm, along with imperative and functional. They are really interesting. You can look them up.

Python has a dynamic type system. In the easiest example, that means you do not have to define the data type of a variable before using it. You can just pick a name and assign a value, and you're done.

```
>>> number = 10
>>> print number
10
>>> print type(number)
<type 'int'>
>>> string = "Hello world"
>>> print type(string)
<type 'str'>
```

Easy. Right? Now let's dig in to our main topics.

Chapter Two: Class and Objects

Like I mentioned before, Class and Object is the two basic concepts on which OOP is done. It's easier to understand if you stop thinking this from a programmers point of view. Living life and programming are more similar to each other than you can imagine. Everything you see around you are objects. All of them belongs to some class. You utilize the objects to perform some action. That is life. And so is object-oriented programming.

What is a Class?

A class is what you may call in your daily life *a type of things*. Vehicle, Car, Train, Animal, Bird, Human, Furniture, Table, Chair, any type of thing you can imagine, they are all classes.

When I say *a Bird*, I don't mean the bird that's sitting on the tree beside your house. A Bird is a type of living being. So is a Human, or a Dog. Classes can be recognized by the virtue that they'll have some properties that will be common to all the things of that type. E.g. That Spitz at your neighbour's home is a kind of Dog. And a Dog barks. That's a property of a Dog. Hence all living beings of the type Dog, will bark. Maybe that Spitz barks a bit too much. But it does bark and that's the point. Similarly, a human would have two legs and would walk or talk or sing. A bird would have wings, would chirp and lay eggs.

A class can have subclasses. Subclasses are also classes but they inherit some properties from the superclass, the class they were *extended from*. Sounds hard? Let me make is clear.

Look up and read the examples of class I've written. I've put some hierarchy there. From each 3 of those examples, the first one is a superclass, and next two are subclasses. First three are Vehicle, Car and Train. You can define them like this -

```
class Vehicle():
    has_wheel = True
    runs_on_fuel = True

    def honk(self):
        print "Honk! Honk!"

class Car(Vehicle):
    number_of_wheels = 4

class Train(Vehicle):
    number_of_wheels = 108
```

Each class has some properties. It can have some unique properties, or it can inherit some properties from a superclass. In the above example, Vehicle is the superclass, because Car and Train are *extending* it, and for the same reason Car and Train are subclasses as they are the ones that are *extending* Vehicle.

A Vehicle class has two properties `has_wheel` and `runs_on_fuel`. Any *thing* of the type of Vehicle would have these two properties. Then we're *extending* the class Vehicle and creating the classes Car and Train. The syntax `class Car(Vehicle)` tells python that Car is a class and it will inherit *all* the features of a Vehicle. Hence, the classes Car and Train by default have those two properties, `has_wheel` and `runs_on_fuel`, in them. And they both also define a property of their own. A Car has 4 wheels, while a Train has 108 wheels. (The numbers are approximate)

A Class can also have methods. Methods are also known as subroutines or procedures or functions. Methods are functions that belongs to a class. These methods define actions that a thing of a certain class can perform. E.g. a Vehicle can honk it's horn. It's an action the Vehicle can perform. And as all the subclasses of Vehicle inherits all the properties and methods, a Car and a Train can also honk their horns.

So we found that a Class can, in it's most basic form, hold some properties, these are features of that class, and some methods, these are some actions the things of that class can perform. These *things* of the type of certain class, is called an *Object*.

Class names start with capital letter

That's a convention we follow while writing python codes. Class names should start with capital letters. That helps us distinguish them from object names or other types of variable names.

What is an Object?

An object can be considered as an example or instance of a class. Remember that bird on the tree, it's an example of the class Bird. Or you can call it an instance of the class Bird. Similarly, that Honda Civic you saw on the road other day is an instance of the class Car.

Now, what's so special about objects? An object or instance of a class contains *all* the properties and methods of it's class. So any instance of the class Vehicle would have `has_wheel` set to True, and any instance of the class Car would have `4` wheels.

OK. Now where do we find objects? Well, everything is an object. Every single thing you see around yourself is an object. That book, that pen, this keyboard, that cellphone, that tree, that moon, that sun, everything is an object. The sun you see everyday is an instance of the class Star. A Star radiates energy. Hence sun radiates energy in the form of light. Our beloved earth is an instance of the class Planet. Planets are round in shape. hence earth is round.

Now let's take that Honda Civic and prove our theories from previous section. Civic is an instance of the class Car. Hence it has `4` wheels. Now, the class Car is a subclass of Vehicle. Hence any instance of the class Car would also have all the properties of the superclass Vehicle. Hence Civic `has_wheel` and `runs_on_fuel` set to True. Seems legit? Of course it does. Want some examples?

Let's first define our classes -

```
>>> class Vehicle():
...     has_wheel = True
...     runs_on_fuel = True
...     def honk(self):
...         print "Honk! Honk!"
...
>>> class Car(Vehicle):
...     number_of_wheels = 4
...
>>> class Train(Vehicle):
...     number_of_wheels = 108
...
```

Now let's create some objects -

```
>>> vehicle = Vehicle()
>>> print vehicle.has_wheel
True
>>> vehicle.honk()
Honk! Honk!

>>> car = Car()
>>> print car.runs_on_fuel
True
>>> print car.number_of_wheels
4
>>> car.honk()
Honk! Honk!

>>> train = Train()
>>> train.number_of_wheels
108
>>> train.honk()
Honk! Honk!
```

I think the code pretty much talks for itself. I'll still explain it. This is how we create an object from a class -

```
>>> vehicle = Vehicle()
```

Now any property or method of an object can be accessed by a simple `.` (dot). Syntaxes are -

```
>>> object_name.property_name
>>> object_name.method_name()
```

Hence you see, when I print `vehicle.has_wheel`, it prints `True`. Because the Vehicle class has the property `has_wheel` set to `True` and like we learned, any object of a class by default gets all the properties and methods of that class.