00 Introduction to OOP Part Zero

November 8, 2020

Coursework delivered by: Alison Mukoma

Copyright: Evelyn Hone College cc DevsBranch.

1 Object Oriented Programming

Python programes are multi-paradigm; you can write programs or libraries that are largely procedural (what we have seen so far), object-oriented, or functional. Examples of OOP languages are Java and C#, C++ etc.

1.1 History

1966 - **Alan Kay** pioneers the OOP works. He was largely inspired by Skecthpad (MIT project created by Ivan Sutherland in 1960–61)
1960s - Simula language
1970s - Smalltalk language developed at Xerox PARC by Alan Kay, Dan Ingalls and Adele Goldberg
1970s - Influenced Lisp
1980s - Objective-C, C++
1990s and later - Python (1991), Java (1991), C# (1999), VB.Net (2002)

What's the original idea of OOP? 1. Message Passing - A method call is message passing. It is conceptualized as a message (the name of the method and its input parameters) being passed to the object for dispatch. 2. Encapsulation - binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse

Or in **Alan Kay's** own words
> "… how to organize things so that interesting worthwhile models are both easy to build, and help make complex things more understandable. Simple things should be simple, complex things should be possible."

1.2 Nerds Section

History and Future of Computing, Robert Martin
Future of Programming, Bret Victor <- Highly recommended

1.3 Python's flavour of OOP - In a Nutshell

1.3.1 Class vs Instance

Strictly speaking, everything in Python is an **object**!

1.3.2 Method vs Function

A `class` is simply a logical grouping of data and functions (the latter of which are frequently referred to as **methods** when defined within a class).

What do we mean by "logical grouping"? Well, a class can contain any data we'd like it to, and can have any functions (methods) attached to it that we please. Rather than just throwing random things together under the name "class", we try to create classes where there is a logical connection between things.

1.3.3 Class vs Instance variables/methods

Classes can be thought of as blueprints for creating objects. When I define a Customer class using the class keyword, I haven't actually created a customer. Instead, what I've created is a sort of instruction manual for constructing "customer" objects.

Sample Code ...

```
class Customer(object):
    """A customer of FNB Bank with a checking account. Customers have the
    following properties:

    Attributes:
        name: A string representing the customer's name.
        balance: A float tracking the current balance of the customer's account.
    """

    def __init__(self, name, balance=0.0):
        """Return a Customer object whose name is *name* and starting
        balance is *balance*."""
        self.name = name
        self.balance = balance

    def withdraw(self, amount):
        """Return the balance remaining after withdrawing *amount*
        dollars."""
        if amount > self.balance:
            raise RuntimeError('Amount greater than available balance.')
        self.balance -= amount
        return self.balance

    def deposit(self, amount):
        """Return the balance remaining after depositing *amount*
        dollars."""
        self.balance += amount
```

```
        return self.balance

patricia = Customer('Patricia Pythonista', 1000.0)
esther = Customer('Esther Pythonista', 2000.0)

patricia.withdraw(100.0)
esther.deposit(200.0)
```

Trivia! 1. Which is class object? 2. Which is instance? 3. How many Customer class is there? How many customer instance? 4. Take a guess, what is self in all the methods in Customer class? 5. Which one is instance method? 6. Take a guess, what is **init** method? 7. Which is the instance variable? 8. How do we access an instance variable? 9. Can we have self.balance (another variable) outside of **init** method?

Note: If we have time, we'll go through @property!
Ref: https://jeffknupp.com/blog/2014/06/18/improve-your-python-python-classes-and-object-oriented-programming/

1.3.4 Class Method and Variable

Where is the class variable and class method? When should we use class variable / method (*hint*: datetime)?

```
class Student(object):
    ID = 1
    def __init__(self):
        self.id = Student.ID
        Student.ID += 1

    @classmethod
    def get_student_count(cls):
        #return Student.ID
        return cls.ID - 1

s1 = Student()
Student.get_student_count()
```

Warning!

```
class Student(object):
    DEFAULT_STUDENT_CLASS = 'Business Block F1'

s1 = Student()
s1.DEFAULT_STUDENT_CLASS = 'Business Block F6'

print(Student.DEFAULT_STUDENT_CLASS)  # 'Business Block F1'
print(S1.DEFAULT_STUDENT_CLASS)      # 'Business Block F6'
```

1.3.5 Static Methods

If Class method is method at the Class level, instance method is method at the instance level, then what is static method? Why do you need static method?

Note: It's not used very often, use it wisely.

```
class Car(object):
    ...
    @staticmethod
    def make_car_sound():
        print 'VRoooooommmm!'

Car.make_car_sound()
```

1.3.6 Summary

1.4 Nerds Section

Python Methods vs Function under the hood, Thomas Ballinger
Python's Instance, Class, and Static Methods Demystified
Python's Class Development Toolkit, Raymond Hettinger <- Recommended!

1.5 How is class related to dictionary?

Remember dictionary stores key/value pairs? How would class looks like in dictionary?

Note: This does not mean class is stored in Dictionary form. We'll need to dig deeper into Python memory to understand better.