A Simple Guide to Object Oriented Programming for Data Scientist

How to read complex Python packages with ease.

Thu Vu Follow
Apr 28 · 4 min read ★

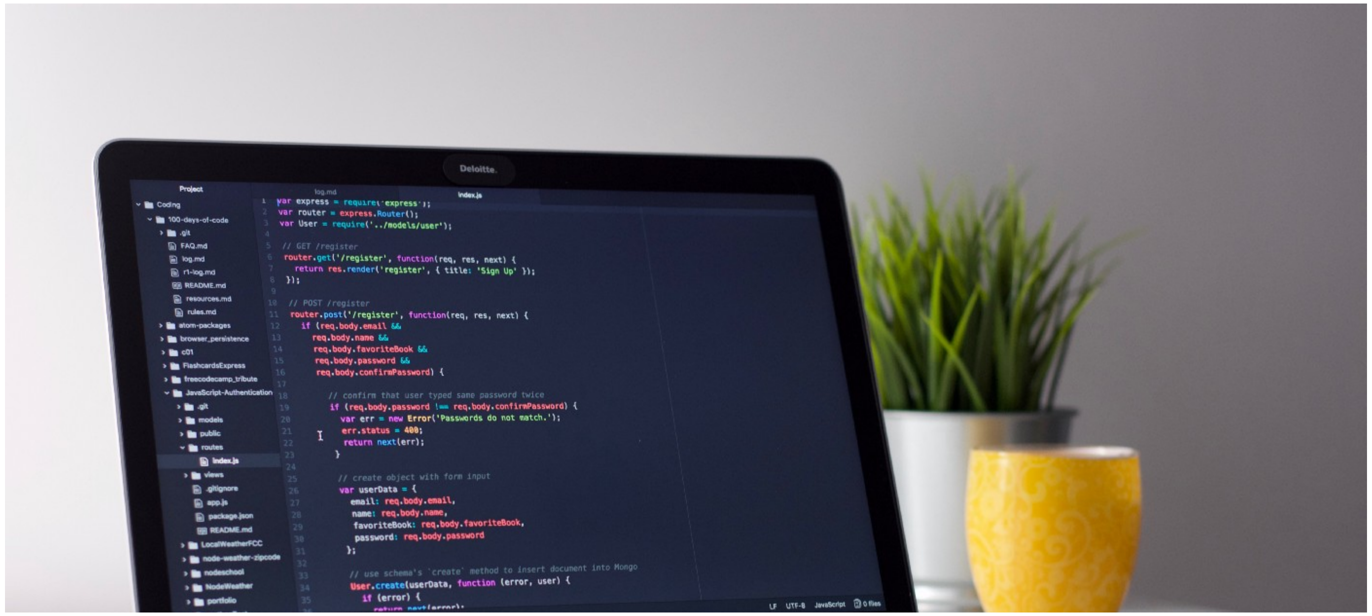


Photo by Clément H on Unsplash

Being a data scientist, you may not write Object Oriented (OO) code every day like a developer would do. You may never have to write OO code in your whole career! However, without know it, you are interacting daily with object oriented programming (OOP) through your use of packages and frameworks. Key data science libraries, such as *pandas, numpy, and scikit-learn* all heavily rely on OOP.

Take a look at these libraries' source code, they are full of classes, methods, attributes, etc. You might also have wondered why you need to declare a regression model (for example) as an instance of a class, and then run a fit method to train your machine learning model.

If you are a very curious individual, you may want to read and understand the source code of a cool machine learning package. Understanding OOP can help you do that with ease. If one day you want to write a Python package or framework, this knowledge will be extremely valuable.

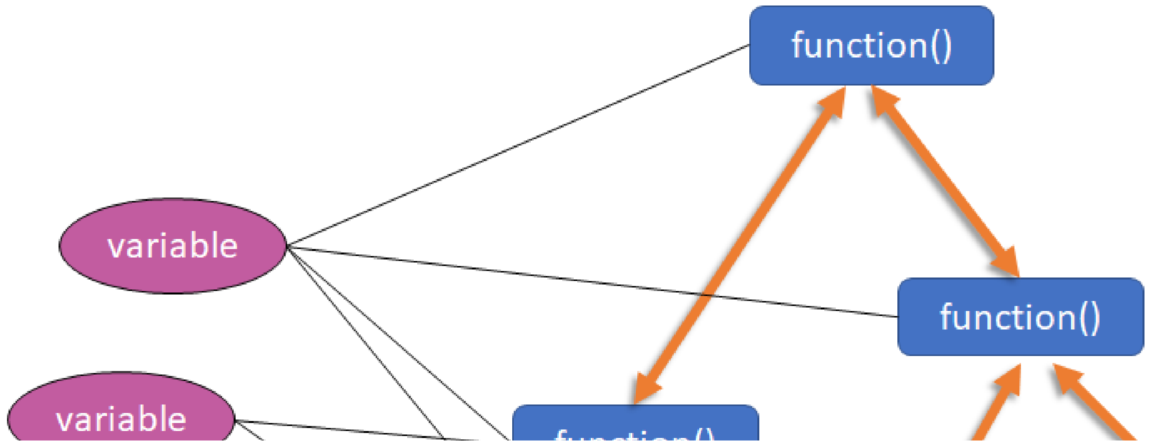
In this post, I will explain some main OOP principles to get you started.

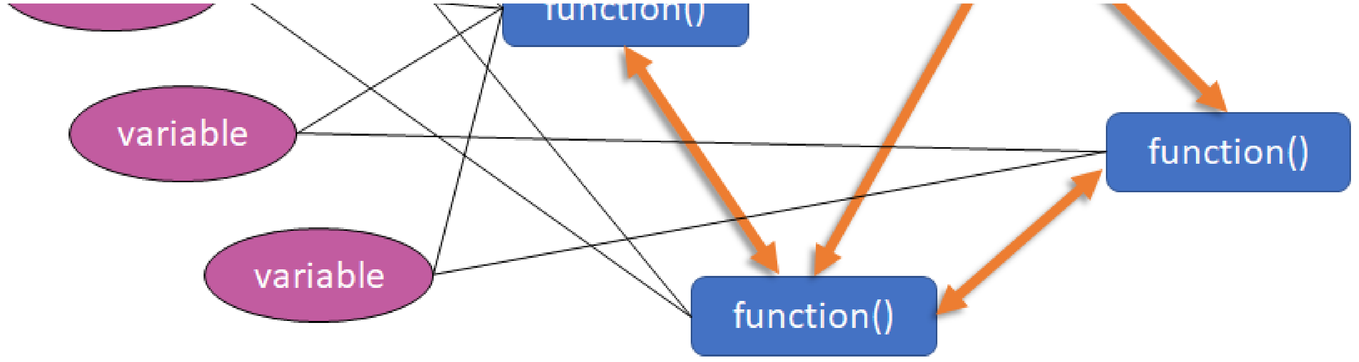
. . .

First of all, why we need OOP?

We cannot understand nor appreciate OOP if we don’t know what kind of problem it solves.

Most of us dread spaghetti code because it’s too confusing to read and maintain. But how is spaghetti code created?

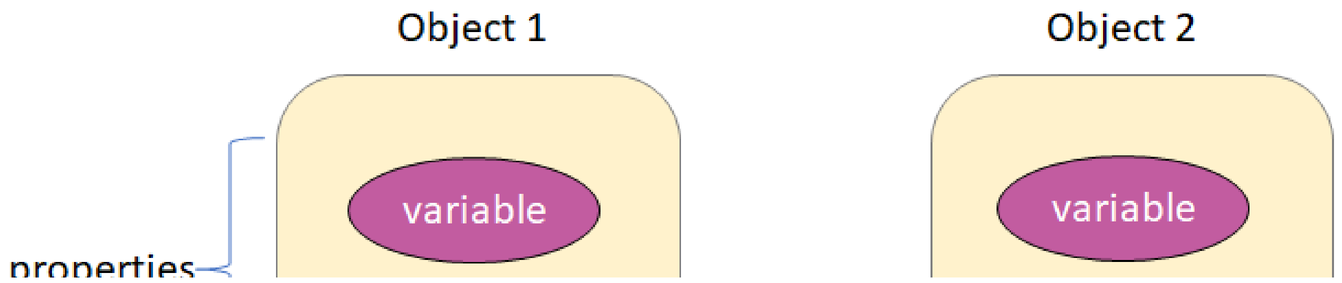


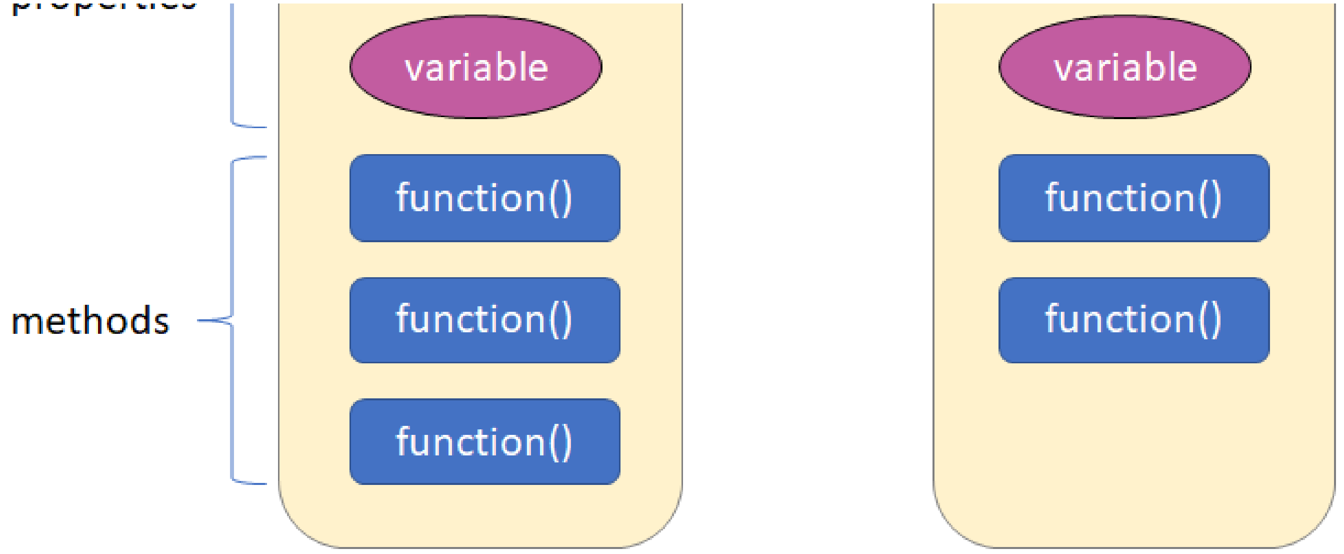


Spaghetti code. Source: Author.

Take a look at this code structure. I hope it looks familiar to you :). We can see 5 different functions calling each other, and a bunch of global variables that are accessed and used by one or multiple functions. Just imagine this structure expands a few times, things will start getting very messy and difficult to follow. OOP solves this problem through two principles: Encapsulation and Abstraction.

Encapsulation



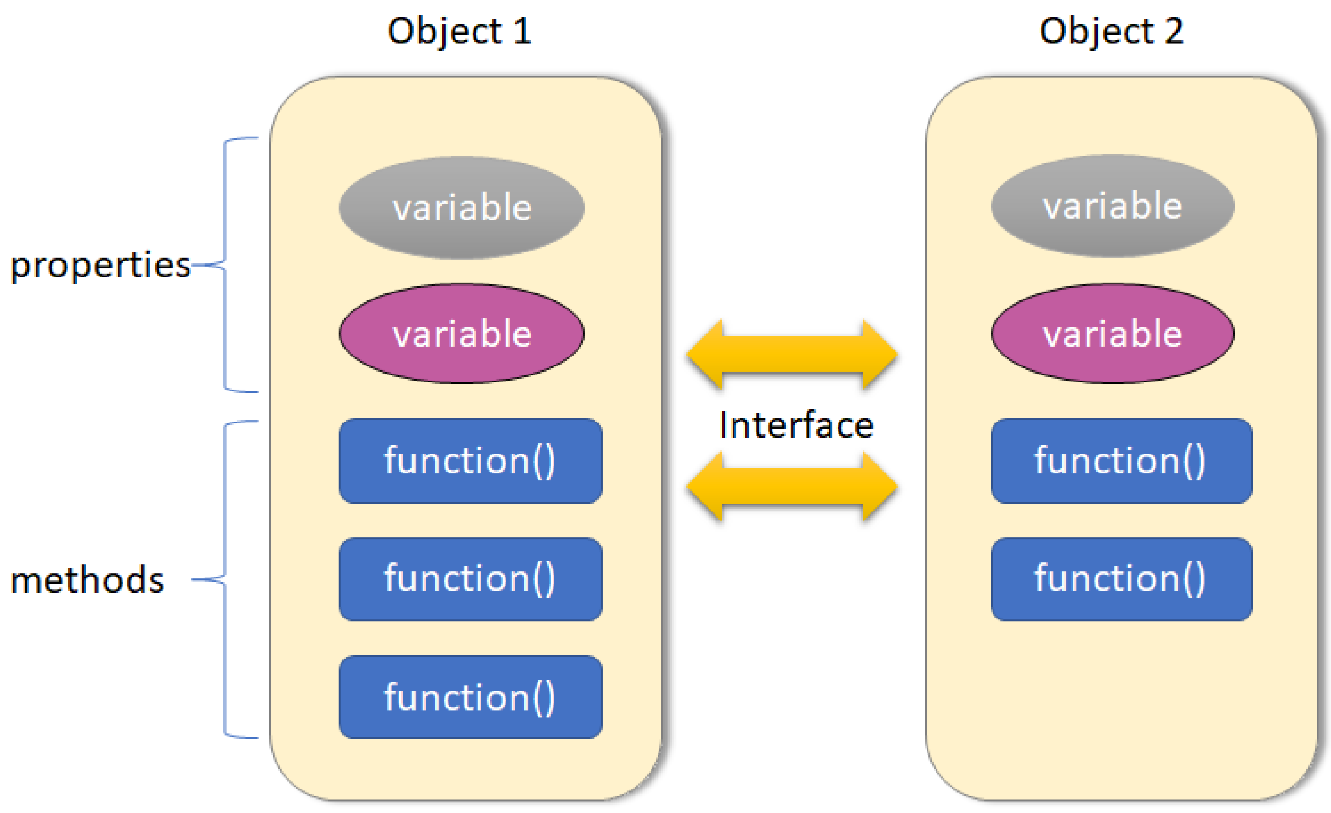


OOP groups variables and functions from our spaghetti structure above together into entities called “objects”. Variables inside objects are called **properties/attributes**, and the functions are called **methods**. Think of properties as characteristics of an object (such as a cat has blue eyes). On the other hand, methods are essentially the ability of an object to do things (such as a cat knows how to catch mice and say “meow”).

Objects interact with each other by making reference to each other's properties and by calling each other’s methods.

Encapsulation makes the code **easier to reproduce and maintain**. If we want to replicate an object to 10 objects, we simply replicate the whole object, instead of replicating each individual variable and function within the object.

Abstraction



When we don't want other objects to access and modify properties of an object, we **hide** these properties from other objects (i.e. the outside world).

Ideally, only the essential elements of an object are made available to other objects through the object's interface.

This is called "abstraction". Our mobile phones are examples of abstraction. Their interface offers us only the relevant handles to use them, but things like chips and memory cards are hidden from us.

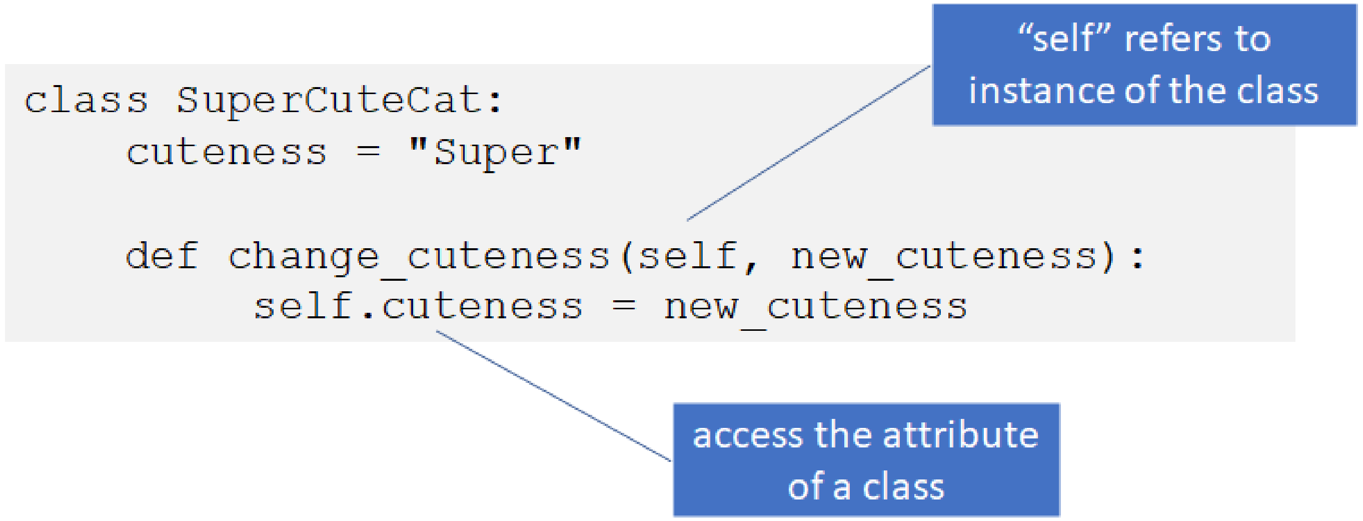
OOP in Python

In Python, **classes** serve as the code templates to create objects. This is similar to constructor functions in JavaScript.

An object is created using the constructor of the class. This object will then be called the `instance` of the class. In Python we create instances in the following manner:

```
Instance = class(arguments)
```

Let's look at an example of a class below:



```
class SuperCuteCat:
    cuteness = "Super"

    def change_cuteness(self, new_cuteness):
        self.cuteness = new_cuteness
```

"self" refers to instance of the class
access the attribute of a class

Example of class. Source: Author

Functions within a class cannot access directly attributes of the class, just like a normal "global variable" as how we may think about it. Instead, we need to use the `self` keyword to access the attributes of the class. This is also similar to JavaScript. In Python, the `self` keyword is always put in the first argument.

To instantiate the class, we simply call:

```
>>> MySuperCuteCat = SuperCuteCat()
```

You may also often see `__init__` method in Python classes. This method is simply used to initialize several attributes of a class.

. . .

I hope this post gives you a bit clearer idea of what OOP is, and helps you understand Python packages and frameworks better as a data scientist. If you are interested in learning more about OOP, this page might be fun to read.

Thank you for reading! Enjoy learning.

Programming Data Science Coding Python Object Oriented