Object-Oriented Programming - Introduction

 (https://github.com/learn-co-curriculum/dsc-oop-intro-v2-4)  (https://github.com/learn-co-curriculum/dsc-oop-intro-v2-4/issues/new/choose)

Introduction

In this section, you'll be introduced to the concept of object-oriented programming (OOP) in Python. OOP has become a foundational practice in much of software development and programming, allowing developers to build upon each other's code in a fluent manner.

Programming Paradigms

Programming paradigms are formal approaches for structuring code to achieve the desired results.

Why Do We Need Them?

For very simple programming tasks, there is essentially only one "correct" way to structure the code. For example, if you needed to print the string "Hello, world!", this is how you would do it:

```
print("Hello, world!")
```

```
Hello, world!
```

But once your code starts to get more complex, the structure gets less intuitive and obvious. For example, if you needed to reshape some data then display a bar graph, or fit a model then use it to make predictions, how would you design that?

Deciding on a paradigm and sticking to it helps to guide your code design choices, and helps others to understand what your code is doing.

Procedural Programming

The oldest (and probably most intuitive) modern programming paradigm is procedural programming. This involves writing a series of sequential steps to be executed, possibly with the use of techniques for ***control flow*** (e.g. `if` statements) and ***modular procedures*** (e.g. functions).

Data science code written in a **notebook** is almost always following a procedural programming paradigm. It is useful for telling a story with a single thread, but less useful for building libraries or

software that runs without human intervention. Once code starts to get more complicated, we start incorporating more-complex paradigms such as functional programming or OOP.

Functional Programming

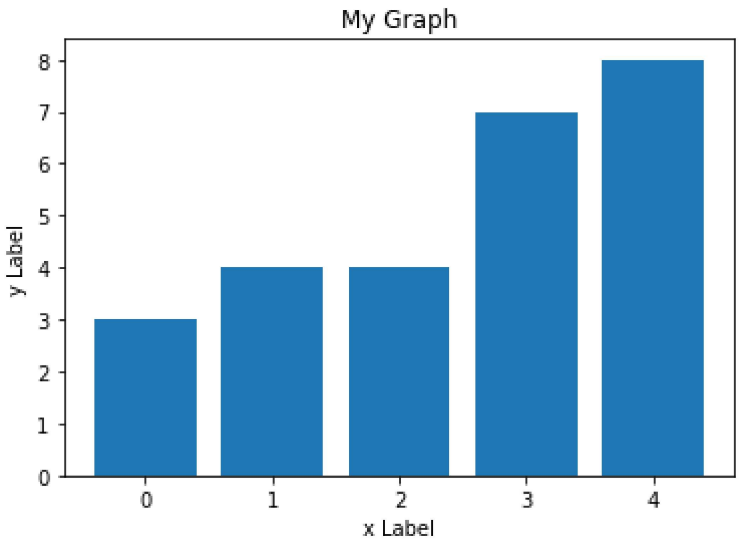
"Purely functional" programming, using a language like Haskell or Clojure, means that procedural programming is abandoned entirely -- rather than a series of steps, the program consists only of functions, which in turn can be composed of functions or apply functions.

In the development of data science libraries, they tended not to use purely functional programming, but nevertheless incorporated some functional principles.

For example, here is the functional interface to Matplotlib:

```
import matplotlib.pyplot as plt

plt.figure()
plt.bar(range(5), [3, 4, 4, 7, 8])
plt.title("My Graph")
plt.xlabel("x Label")
plt.ylabel("y Label");
```

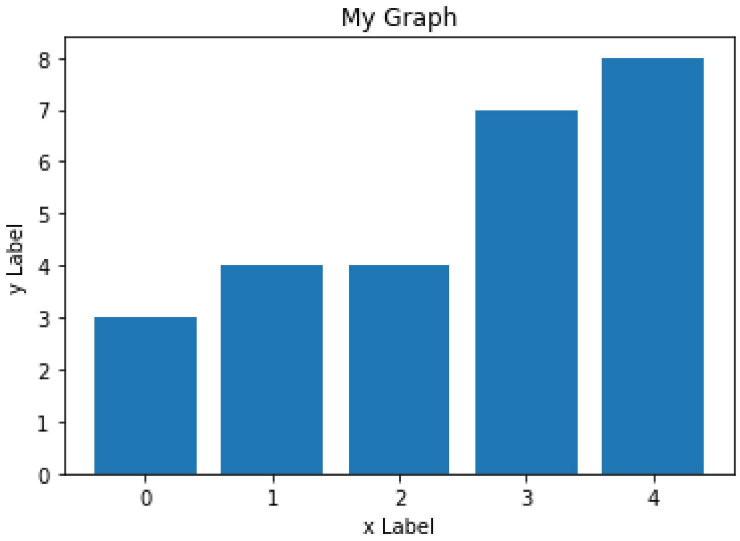


Note that we created this graph without instantiating any variables. We just imported the library, then called a series of functions to create the desired graph. We could rewrite that code snippet like this, to make that aspect even clearer:

```
from matplotlib.pyplot import figure, bar, title, xlabel, ylabel

figure()
bar(range(5), [3, 4, 4, 7, 8])
title("My Graph")
```

```
xlabel("x Label")
ylabel("y Label");
```



This approach is still preferred by some "old school" data science practitioners, but it has some issues.

It uses ***global variables***, which can get messy as code gets more complex. When the `title()` function is called in the above snippet, for example, the internal logic first has to find the current global axes object, then apply the label to that object. For a programmer to understand what axes object that is, they would need to closely follow the steps of the code, since there is no unique variable assigned to it. With no variable assigned, that also means that the code is less flexible and steps must be performed ***one at a time***.

Object-Oriented Programming (OOP)

Object-oriented programming takes these global variables and functions and makes them into "member variables" (AKA ***attributes***) and "member functions" (AKA ***methods***). This allows code to be more organized and clear.

For example, in the previous functional Matplotlib example, you might ask *What is `title()` being called on? Is it the figure or the axes?*

To answer this, we could look at the **Matplotlib source code (https://github.com/matplotlib/matplotlib/blob/v3.5.1/lib/matplotlib/pyplot.py#L3024-L3027)**, which shows this:

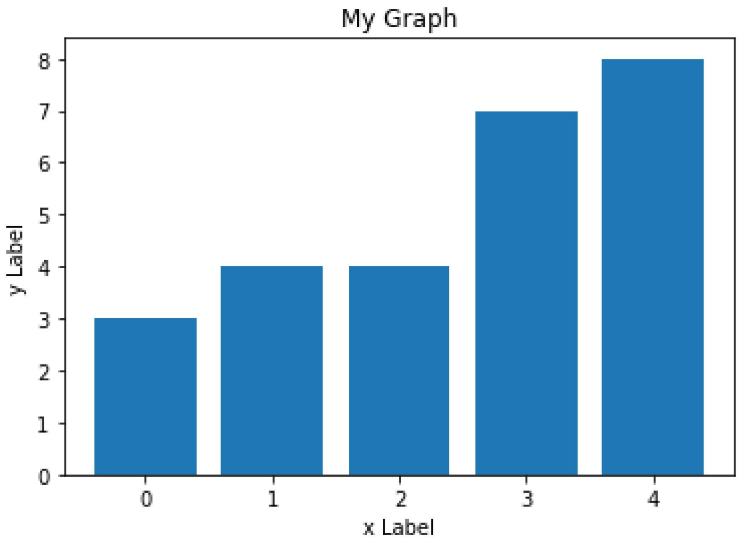
```
def title(label, fontdict=None, loc=None, pad=None, *, y=None, **kwargs):
    return gca().set_title(
        label, fontdict=fontdict, loc=loc, pad=pad, y=y, **kwargs)
```

`gca()` means "get current axes", so we can tell that this is being applied to the axes.

Or if we use the object-oriented Matplotlib interface instead, the answer becomes much clearer, just by looking at our code:

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots()
ax.bar(range(5), [3, 4, 4, 7, 8])
ax.set_title("My Graph")
ax.set_xlabel("x Label")
ax.set_ylabel("y Label");
```



As you can see, the title is being applied to the axes, not the figure. We can tell because the method call is structured like `ax.<method name>()` and `ax` is our axes variable.

A key takeaway here is that ***you can often do the exact same thing using different paradigms***. They are just different approaches to structuring code, and different people might prefer different approaches.

OOP Topics

In this section, we will cover:

Classes and Instances

A Python class can be thought of as the blueprint for creating a code object. These objects are known as an instance objects or instances. We'll go over how to create classes as well as instances.

Methods and Attributes

Next, we'll dive deeper into how to specify and invoke the functions and variables that are "bound" to instance objects. This includes the ***encapsulation*** and ***abstraction*** principles of OOP.

Inheritance

Inheritance means that classes can be defined that take on the traits of other classes. This is especially useful when interacting with complex code libraries.

OOP and Scikit-Learn

Scikit-learn is the most popular machine learning library in use today, and its organization relies heavily on object-oriented programming. We'll go over the types of classes used and some of the most common methods and attributes you should know about.

Summary

Object-oriented programming (OOP) is a way of organizing your code that can make many types of applications easier to write by combining related variables/properties and functions/methods into objects containing both behavior and state.

How do you feel about this lesson?

Have specific feedback?

Tell us here! (https://github.com/learn-co-curriculum/dsc-oop-intro-v2-4/issues/new/choose)