Lesson 6: Intro to Object-Oriented Programming

Motivation

Question: If we have some rectangle, and we need to keep track of where it is (x and y coordinates) in our code, how many variables do we need to do that?

> **We need 2 variables:** one for x and one for y

Question: If we actually had 5 rectangles, now how many variables do we need to keep track of?

> **10 variables -** 5 x and 5 y. We could put those into lists (i.e. x_coordinates = [] and
>
> y_coordinates = []) which kind of helps.

Question: Now what if we also needed to keep track of how big those rectangles are.. how many variables do we need now?

> **We need 4 variables per rectangle:** x, y, length, and width

Question: If we had some function called describe() that prints out some information about the rectangles (i.e. “Rect at (x, y) of size (len, width).”) how many parameters would that function take?

> **It needs 4 variables…**

Okay, what's the point?

Right now we don’t really have a nice way of grouping related pieces of data and functionality together. We could use a dictionary, but that

a) doesn’t really help us right functions for these rectangles, it just means we only have to pass 1 argument per function
b) still means when we create each rectangle, we need to remember to initialize all the keys and values that other pieces of code expect

Classes and Objects

In Python and in other object-oriented languages we have a way of defining our own types through **classes.** We can create a class for a type we want, and then **instantiate** (create instances of) that class to create **objects.**

Each class defines **what data each object has** and **what the behaviour of each object is** (i.e. functions of the class). Sometimes we call these functions **methods** (functions that belong to a class).

Here's a snippet of code showing classes in Python:

```
# Defining a class for rectangles
class Rectangle:
    def __init__(self, x, y, length, width):
        self.x = x
        self.y = y
        self.length = length
        self.width = width

    def describe(self):
        print(f"I am a Rectangle at ({self.x}, {self.y}) of length {self.length} and width
{self.width}")

# Creating rectangle objects
my_first_rectange = Rectangle(100, 100, 50, 80)
other_rectangle = Rectangle(10, 20, 30, 40)

# Calling methods
my_first_rectange.describe()
other_rectangle.describe()

# Getting out data
print(my_first_rectange.x)
```

Whats __init__?

__init__ is a **special function that gets called everytime you create an object.** In this __init__ we are saying **when we create a Rectangle we will provide an x, y, length, and width** parameter. Inside of the __init__ we save that data inside of the object.

Note: In general we shouldn't put anything special in __init__. We should only set some attributes

Dot Notation

We use dot notation to **access some data in an object or call a method on that object.** In the line `my_first_rectange.describe()` we are calling the describe() method for the my_first_rectangle object.

Private and Public Members

Sometimes when we talk about the data in a class we call them **member variables** or **attributes.** In the example above, **x is a member of each Rectangle.**

As a **convention in Python,** when the name of a member variable (or method!) starts with two underscores, (i.e. self.__color) **we call those "private" members.** In other languages, you have to care a lot about that, but in Python, just know that **we use private members for data that other code shouldn't touch** (i.e. a function we only use inside the class). Python makes it hard for other code to use it… although it's not impossible.

When we name something with only one underscore (i.e. self._one_underscore) **technically other code can use that** but they shouldn't.

Exercises

1. Write a class for a triangle. Up to you to decide what data the class holds, but it should provide an area() and perimeter() method.

2. Use your Triangle class to create a list of triangles. Then add a describe() method in the Triangle class, and use it to describe all the rectangles.

Things that are kind of nice

- If we used classes and we need to change something in our code, most of the time it's not a big deal
    - i.e. we usually edit some existing method, and the change will impact all objects of that class automatically

- When we want to do cool things, a lot of the time, we use a library and the library makes us use their objects. So we need to know what objects are and how to create/use them.

- If we want to have objects that have a lot in common (i.e. a Ford F-150 and a Honda Civic) we can create a "base class" and then create "subclasses" that inherit some parent functionality… but we won't really learn or use that for now.

- We can create classes that have other objects inside of them (i.e. a Car class might have an Engine object inside of it). This means as our code gets more and more complicated, we can use these class abstractions to ***simplify our life.***

More Exercises

1. What data should a Car hold? What are the types? Write a class for a Car.
    - Write a method price() that returns some price (i.e. random price).
    - Write another method called describe() that prints out information about the car (i.e. "I am a Ford F-150 that costs $10,000".

2. Write a class that describes one bouncing ball (i.e. what parameters are there?) and gives it functionality (i.e. draw on the screen, move the ball).

3. Use the BouncingBall class to create a pygame app that has 3 bouncing balls instead of 1.
    a. **Fun fact:** In pygame and in other graphics animation programming, we actually just **draw one frame, clear the screen, then quickly draw the next frame.** That's why in pygame we see the following code which **sets the entire screen to white** (i.e. clear the screen) then **updates what's on the screen (on top of the white)**.

       This all happens so fast that our eyes can't actually tell….

```
    screen.fill((255, 255, 255))

    …. update the screen (i.e. draw a circle) …

    pygame.display.flip()
```

Homework:

1. Can you create a class Animation that lets you add shapes and then runs the animation through pygame?... Basically I want a class that represents 1 animation window in pygame. I want to be able to add shapes to it (i.e. through a method add_shape()). I then want to call a method run() (i.e. my_animation.run()) and have the window pop up and have everything run.
    a. **Challenge:** If you can get this working for Circles, can you make your Animation class also accept other shapes?