Object-Oriented Programming in Python

If you find yourself doing the same thing in multiple places, you should use a function. If you need to do the same operation on a group of items, you should use a loop. The use case for classes is a bit more complex. In general, a class is useful when you want to group together variables and functions because they represent a single thing that has a set of properties.

A class defines a type of object. All objects are instances of classes, and therefore share properties with objects of the same class.

I've provided a stripped-down example of a class I recently wrote as an example. I need to process ~800 spectra, and for each I only have a 2-d image that is 4096x4096 pixels, so the images take up a significant amount of memory. I created a class so that I can have each `TouImage` instance keep track of the attributes related to the data.

Class names and attributes

The class name starts with a capital letter, and instead of using `snake_case` as is standard in Python, class names use `CapWords`. Then I define a few class attributes. These are variables I can access from anywhere the class is defined. I could have them be attributes of instances of the class, but since they're the same for every member of the class, I set them as attributes of the class itself.

Class methods

Class definitions also include function definitions. The first of these is usually `__init__`. Python objects often have methods or attributes that start and end with two underscores, sometimes called dunder methods or dunder attributes. `__init__` is called after a new instance of the class is created and is passed the same arguments as the line that creates the class. `__init__` is not a constructor, it is an initalizer. There is a constructor in Python, it's called `__new__`, but it's not commonly used. Typically `__init__` is used to assign instance variables from the arguments. In my example, I create a new `TouImage` object and assign its `.path` and `.time` based on the arguments passed in, then initialize the other attributes it will have to `None`. This isn't necessary but it is good practice to define every attribute in `__init__`. Just as in normal Python with variables, if you want an instance attribute (`self.`) you can just start using it.

I also define another dunder method, `__lt__`, which defines the behavior of the `<` operator. This is sufficient to enable sorting, as I do later on with a list of `TouImage`s. Every functionality you see in Python you can get out of your own classes by defining dunder methods. You can enable iterating over a class, adding one class to another with `+`, or calling an instance like a function.

Then I define class methods, which I later use to manipulate the instances of the class, in this case to process the images. In this case, they're structured around using a particular function `remove_cosmic_rays` which needs a list of `TouImage`s sorted by exposure time, and will remove cosmic rays from them.

The Rabbit Hole

Object-Oriented programming is a powerful tool, but like all other tools can be used incorrectly and just make your code harder to use. A general rule of thumb is that if you ever define a class that has `__init__` and one other method, you actually wanted a function. Python's objects are easy to get into, but the rabbit hole is deep indeed with all sorts of crazy things like metaclasses.