When to use OOP

**What exactly is an object?**

Objects must have both data and behaviour. If it is only data, then better off stored as a list, set or dictionary or some other data structure. If it is only behaviour, no stored data, better to just use a function

**OOP vs. Procedural**

OOP is meant to be more self-documenting even though the amount of code written can be a lot longer than procedural. Moreover, OOP is more extensible cause easier to just add functions to objects.

Whether to use OOP or simply let data persists in variables and have functions to manipulate them also depends on the complexity of the problem that is being worked on. If you know you will have multiple instances of these data, then objects would be better. Else procedural would be faster cause the mental model to understand them is less complex.

The interaction between objects is a big advantage for OOP. I.E. procedural has no such thing as inheritance.

**How to add behaviour to class data with properties**

Other languages like Java insists that attributes can never be accessed directly. Instead, they have methods like getters and setters to read and write attributes n objects. The rationale is that 1 day we can add extra code when a value is set or retrieved hence making it for extensible. If in Java, we used direct attribute access and 1 day later we want to add extra code when getting and setting attributes, we would have to trawl the code base and change all instances where we directly access the attribute

Moreover, this better aligns with the principles of encapsulation because public members are not made private. However, in Python we don’t really have the concept of private members.

In Python, we can use the property keyword to make methods look like attributes. Property keyword serves the use case where we can write our code to use direct member access and if we need to alter the implantation of getting and setting, we can do so without changing the interface.

**Manager Objects**

Manager objects are objects that manager other objects. Management objects don’t represent concrete ideas but they communicate between different objects.

Example, program that find and replace action for text files stored in compressed ZIP file. Objects required are:

1. ZIP file
2. Text File
3. Manager Object that does the following
   1. Unzip compressed file
   2. Perform the find-and-replace
   3. Zip up the new file

The Manager Object would look like this

Manger object would also have an overall manager method for each of the 3 steps. The manager method delegates responsibility to other objects.

While we can do all the 3 steps in 1 method without creating an object, the advantages to separating the 3 steps in 1 manager method on a manger object are

* Increased Readability -> Code for each step is self-contained
* Extensibility -> If we want to use compressed TAR files instead of ZIP file, we can override the zip and unzip methods without duplicating find\_and\_replace method
* Partitioning -> External class could create an instance of this class and call the find\_replace method directly on some folder without having the zip the content

In our example, the Zip and Text files already are objects native to Python.