Object-Oriented Programming: Reflection

The idea of object-oriented programming (OOP) is to create and work with data structures that the developer defines itself. Such a self-built structure is called a “class”. A class is in this sense comparable to tuples which also allow as entries different data types (for example to combine strings with integers or float entries within a tuple), but in a class it is called an “instance” that would then represent exactly one tuple and the class itself is the structure that defines what one tuple is (= its “attributes”) and what you can do with it (= its “methods”). Reading a book like (Phillips, 2018) which relates OOP to Python, is for a newcomer not that easy and it is probably for a beginner a good idea to have some sample code in his/her repository (perhaps from previous courses) that already defined some classes. Furthermore, it is also useful to bear in mind that OOP doesn’t always improve the code in terms of efficiency but rather in terms of readability, reusability and error tracking (upGrad, 2021).

In general, if you never applied OOP before, it is eventually a good idea to read aside from (Philipps, 2018) other books such as (Connolly & Begg, 2015) or (Ambler, 2003) which provides you with the principles of UML (Unified Modelling Language) and object-oriented design. Once you see the relation between OOP and system modelling and you also have some example codes (in Python for example) in your background, you can use the book from Phillips not only to improve details into your own classes (for example using assert statements inside your “__init__” method to restrict values for attributes) but also to connect the whole system. Nevertheless, the latter point is much more difficult to achieve without using further sources such as (freeCodeCamp, 2021) which helps you to understand how attributes can be collected into a list or a dictionary within a class. As a personal note, I think a

beginner in OOP works more like a bricklayer. A bricklayer also uses many bricks as main components but he also needs other sources such as cement, a water bubble, a mortar for combining the bricks. In this sense, the book of Phillips represents a bit the bricks as main source that always helps you to come back to but it is also important to use other resources which give you much more in-depth knowledge when you need it (for example chapter 10 in (Bruegge, 2014) helps you to connect an entire system although this chapter uses Java as programming language).

Having just said this, I think this is one of my personal critics of OOP. I read a lot of books and online-resources before I started to code my first project in using OOP because I felt unsure for a long time. This was related to two main reasons. Firstly, I didn't find one core book that understands the main questions that a beginner of OOP has in mind. Secondly, because of reason 1, I was struggling for a long-time to see advantages in using OOP. For example, when someone already coded before in Python and one already saw a lot of useful data structures and libraries, it is difficult to understand why should you use OOP and also when you use it, how to use it in the most nicely and efficient way? I directly got the idea that OOP improves readability among developers, especially when several developers work on the same project. But understanding main concepts such as encapsulation specifically related to OOP took me some time. To be honest, only after some application of decorators such as "getter" and "setter" within a class broke the ice to get the idea of encapsulation and it increased my persuasion using OOP in the future. Taking the last example again, restricting for example the changeability of a variable within a class in using a decorator can be much more secure than the way where one always has to think about which data structure should be used that allows to change one value but not the other ones (immutable tuple vs. list for example). Furthermore, after

some time I also learned about useful constructors such as “__repr__” which essentially allows you to give an object a customized representation (similar to “__str__”). Why this could be of interest? I think here specifically about output that will be shown to non-Python user. Before, when I wanted to present several data instances to someone, I often used a simple for-loop over a list or I sometimes worked with a pandas data frame in using the pandas library. However, the list approach is not reader-friendly and the pandas data frame often needs some data preparation before. The “__repr__” seems to be a new visualization alternative in this respect.

To conclude, some months ago I started as a beginner in OOP but I had professional experience as a Data Analyst. Although, it took some time that I could see and apply the advantages of OOP, it certainly will improve my professional skills. As a recommendation for every starter, I can only say that reading and listening to different sources is essential in understanding why OOP can be useful (Sometimes it also helps to read simple sample codes). Sticking to one core book (such as (Phillips, 2018)) seems not to be the best approach but a source like Phillips should rather be a backyard where you can always come back to. As I already highlighted, there are other advantages aside from readability of your code from which you can profit as programmer (“getter”, “__repr__” but also the “assert” statement within “__init__”). In this respect, going deeper into OOP in the future is specifically important when you want to improve as a Functional Analyst. However, my personal view is, professions such as a Data Analysts often work with specified tools and libraries (such as pandas in Python) which already provide an analyst with a whole range of functions and classes which will repeatedly be used so that OOP is probably less recommended in this field (often there is no need to define own classes).

Reference List:

Ambler, S. (2003) *Elements of UML Style.* Cambridge: Cambridge University Press

Bruegge, B. (2014) *Object-oriented software engineering : using UML, patterns, and Java.* Harlow: Pearson

Connolly, T. & Begg, C. (2015) *Database Systems: A Practical Approach to Design, Implementation, and Management.* Global Edition. Edinburgh: Pearso.

freeCodeCamp (2021) Object-Oriented Programming with Python - Full Course for Beginners. Available from: https://www.youtube.com/watch?v=Ej_02ICOIgs&t=5100s [Accessed 10 April 2022]

Philips, D. (2018) *Python 3 Object-Oriented programming*. 3rd ed. Packt Publishing

upGrad (2021) What are the Advantages of Object-Oriented Programming? Available from: https://www.upgrad.com/blog/what-are-the-advantages-of-object-oriented-programming/ [Accessed: 09 May 2022]