CS4700_Final

This is a repo containing the work done for my CS4700 Final Report, where I have decided to analyze the two programming paradigms of Object Oriented Programming and Functional Programming. I have written up two code examples, each written with both paradigms.

Simple RPG

The first code example showcases the strengths of Object Oriented Programming and compares it against Functional Programming. It is a simple RPG-style game where there can be heroes, monsters, and boss monsters. The hero can attack and heal, the monster can attack, and the monster can block a hero from healing. This plays into OOP's strengths because it contains many elements with varying capabilities and requirements, which can easily be done with classes and inheritance in this simple scenario. This is difficult to implement properly when using the Functional Programming, since class structures aren't necessarily a part of it.

In the `ex1_RPG_FP.py` file, we can see that the class structure is imitated in functional programming with a dictionary. It would also be possible to include necessary functions in objects by adding functions to the dictionary and calling them later. I decided to forgo this to better showcase how functional programming works by using a minimal amount of variables and avoiding a class-like structure where possible.

The strengths of OOP are greatly showcased in the `ex1_RPG_OOP.py` file, which has the OOP version of this program. The classes are very straightforward, and would be even more clearly defined if Python had an explicitly declared type system. Each of the three elements in the game inherit from a common `Creature` class, gaining attributes from it. This shows inheritance as one of the big points of OOP. This allows a class to inherit attributes and methods from another class.

Merge Sort

The other example I have written up is a simple Merge Sort algorithm. This is directly translated from the Lisp Merge Sort algorithm that we developed for Assignment 2, so it was not developed from nothing. This was helpful, but it did lead to many complications when developing the Object-Oriented variation of the program. This merge sort algorithm showcases the strengths of Functional Programming because it is algorithmic in nature, and is of a step-by-step nature, which is where Functional Programming thrives. It's not so good for OOP because it does not require any customly designed data types or anything like that, and does not require any abstraction whatsoever.

We can easily see these weaknesses in the `ex2_MergeSort_OOP.py` file. One of the biggest weaknesses of OOP showcased here is that Object-Oriented programs can easily grow out of hand on the side of the developer. It is far easier for an Object-Oriented program to become spaghetti code. We can also see that the two classes defined here (`MergeSort` and `Node`) only exist to add complication to the algorithm, rather than make it easier to develop. It was also a lot more difficult to keep track of all the objects that were created and what needed to be passed into different constructers as the merge sort went on.

The functional programming version of this program is found in `ex2_MergeSort_FP.py`, and the strengths of Functional Programming are greatly showcased here. The biggest strength seen here is how simple a functional program can be. A lot of the fat can be cut off of an algorithm when a purely functional paradigm is applied to it. There was also a somewhat significant increase in both time and memory performance when using the functional program over the object oriented one.

Conclusion

In conclusion, I would still say that neither paradigm is strictly better than the other. However, in specific use cases, one method can certainly be better than the other. For algorithmic programs, I would say that functional programming is far better. For programs that require data abstraction and methods to be performed on that data, object-oriented programming would be a lot better to use than functional programming. So in a lot of cases in between those two extremes, I would say it's all a matter of personal preference.