**Analytical Comparison of the Object-Oriented and Imperative Implementations**

Both implementations were created using Python 3. My reasoning for choosing python is because it is a multi-paradigm programming language, meaning that it can support such paradigms like Object-Oriented and Imperative..

Object-Oriented Implementation

With the Object-Oriented style of programming, it is based upon the concept of objects. This allows the objects to encapsulated their data and their various methods which act on that data, which can all only be accessed once an object is firstly initialized. The way in which the code is structured places an emphasis on the data and it’s related actions that can be executed, rather than logic. Essentially, a class is a blueprint for each instance of it, it groups the code based on it’s specific purpose, in contrast to Imperative. With that, the various classes within my program represent the parts of the blueprint for the week calendar, such as the multiple appointments, the different days and the overall calendar itself for a specified week. With this style, It helped me to visualize the code and understand it to a deeper extent as I could envision it like I would with any other real life object.Throughout the development process of my program, I was able to think less about writing the code and focus more on the data modelling.

When I first began tackling the two implementations, I chose to begin with the Object-Oriented style as it controls the complexity of the code more sufficiently through using objects. Also, upon planning what my program will do and how it will do it, I was able to draw up class diagrams to model my different classes and this helped with figuring out the flow of my program and how exactly each class will interact with the other. Not just that but overall it lets the program to be organised within a far more maintainable way. Through completing the Object-Oriented version first, I was more aware in how I would go about completing the imperative version, more so than if it was the other way around.

For this version, I chose to split up each individual class along with the main function in comparison to my Imperative program. As although this isn’t a requirement of the python language, by doing this it meant that each class could be focused on individually, through providing more structure and organisation and allowing me to achieve a clearer understanding of what each class’s part is within the entire program.

Each of my classes rely on one another in order for my calendar program to work correctly, but no inheritance exists between them. Within the appointment class, an object is first initialized upon a user requesting to add to the calendar, the data it requires is the title, starting and finishing time, all as strings, and it can check for if times are valid and in the correct format, it also can convert a time of type string to minutes which can then be used in checking if any times are the same or overlap before adding a new requested appointment. Within the day class, each day object comprises a list of appointment objects. It has the ability to store a new appointment for that specific day and to remove also. Through calling on each of the appointment objects, it applies it minutes method to organise them within the list, so that upon showing the appointments for that day they can be ordered by the appointment that comes first based on their start time. Here in this class, I also used a class attribute to count the number of appointments added for that day, which differs to all the other types of attributes used, as this is an attribute for that day class, rather than an attribute for an instance of day. In my Imperative version, this was not so easily done and I have to simply just return the length of the appointment list. Within the calendar class, I deciding upon using this class as a way to structure my previous two, upon initizaling I chose for it to take in no parameters, but rather to create a new dictionary with a key for each day of the week and their corresponding value initializing the day class for each one, which creates a list of appointment objects that are appended to it.

Lastly, I added a separate main function and imported each class from each separate file. This way I could retrieve the user input by asking which which function they wanted to do and process this by firstly initializing a new calendar object and calling all necessary methods according to the user’s request and then return the results to the terminal. Through creating this function separately to the classes files, I used it to conduct sufficient testing of my classes throughout the development process of each. Also, it made the locationing of bugs within my code far easier.

Imperative Implementation

With the imperative programming style, it is far more straightforward in its approach. The idea behind this paradigm, is to provide an exact set of instructions on what needs to be achieved. Within my imperative program, I tried to achieve a purely imperative style through designing the program using a multitude of functions to manipulate the inputted data. With the structure of the Imperative program, I also had to ensure that the order in which I placed all my functions was correct and that no one was called before the it was defined as the imperative style is sequential. It’s similar to following instruction in real life, like following a recipe for example, however in the case of programming a calendar, I believe the object-oriented style is more suitable. The imperative style requires more detail for each step and this can lead to redundant code. It also allows for the data within the program to be updated or modified by anything as none of the data is refined to a particular class like my other implementation. Not just that but it requires the parameterization of each function too.

In brief, the Object-Oriented style follows the same characteristics as the Imperative does, yet it has far more features included within. However, after having programmed imperatively, I believe that this style would be far more suitable for smaller programs and those which don’t require as much complexity. That being because as my programs got larger and more complex functions were added, I found that it became more difficult to manage in contrast to the Object-oriented version. I believe the Imperative be easier to grasp for beginners, as in order to understand Object-Oriented you’d first need to know how to write code imperatively. However, in turn, having all the code together in the one place representing different functions can help with examining the entire program as a whole and all data is easily accessible can be beneficial in some cases.

Within my imperative program, there isn’t as much structure as seen within my other implementation. However, similarity the overall calendar for the week is also stored through using a dictionary, with the value for each of days being a list of lists that represent the appointments. Each method that is seen within each of the classes in the Object-Oriented implementation is shown as a function, translated to the imperative style. In comparison to the OO version, when a user of the program wants to add a new appointment, the data is split up into a list and then added to the list for the specified day rather than creating an object and so, each individual piece of information of the appointment can only be accessed through indices and not simply through calling instance variables, which can lead to confusion. The inputted data is also checked to ensure the correct format is given, like in OO and if any errors do occur it is printed to the terminal. When a user requests to perform a particular action on the calendar, this version directly calls the corresponding function using the inputted data as parameters.

In conclusion, both of the two versions of the Calendar program retrieve the data and output the results in the same way, along with performing essentially the same functionality. It is simply just the manner in which they conduct and organise their various functionality and how the data is model in each that differs. I believe that for this certain program however, that the Object-Oriented style is the more appropriate option for to use.