Comparing Imperative programming and Object-Oriented Programming.

# My object-oriented implementation

My program includes the use of three different classes; day, week and appointment.

The Appointment class creates an appointment object with the attributes details of the appointment, appointment start time and end time. The class includes a string method override which formats the object to be human readable when printed.

The Day class initialises a day with a name and an empty list for appointments. The appointment list will be made up of instances of the Appointment class. The Day class implements a show appointments method which is used to access the string method of each appointment instance in the appointment list for when it is printed. This class also has its own string method override, used when printing out the day and it’s appointments.

The Week class inherits the day of the week and the list of appointments for a given day from the Day class, the Week class then initialises a list of the days of the week, each being an instance of Day. This Class contains the core functionality of the program; showing the appointments for one day, for the entire week, adding appointments and removing appointments. The function for showing the appointments for the week involves iterating the list of days and printing the name attribute from the day instance then calling the show appointment method from the Day class, allowing access to the string method within in the appointment class. Printing the appointments for an single day is involves the same implementation as printing the week, the only difference is that the function matches the day the user inputs and only prints the name attribute and the appointments for that day. It uses the day input to select the desired day and the other parameters are used to make an instance of appointment which is then added to the appointment list of the desired day. The method for Removing takes in two parameters, the desired day and the appointment that you want to remove. The function iterates through the list of days, matches it with the desired day and then accesses the list of appointments for that day. Finally it the iterates through the list of appointments and tries to match the detail attribute of an instance of appointment within the list with the user input, if there is a match it the pops the index at which it is at in the list.

There is a run method which allows the user to interact with the program.

# My imperative implementation

My imperative program involves an infinite loop that relies on user input to access and change a data structure. The data structure is a dictionary which maps a string, day if the week, to a list, which is a list of appointments for each day. The while loop checks the users input looking to match it to one of the commands.

To print out the whole week we iterate through the items in the dictionary and print out the keys and values in a chosen format.

Printing out all the appointments for one day involves iterating through the dictionary and matching a key to a day that the user inputs, once matched it then iterates through the appointment list and prints out each one.

Adding an appointment involves taking in from the user a day, the details of the appointment, a start time and an end time. Then the program iterates through the keys in the dictionary and matches it to the day and then appends a string, which is comprised of the details, start and end time , to the list of appointments.

Removing an appointment involves taking in two inputs from the users, a day and an appointment. Again, the program iterates through the dictionary in search of the desired day. Once found, it will then iterate through the appointment list, splitting each appointment string and comparing the details, which is in the first position of the split list, to the user input. If there is a match the program then pops the index of the matched appointment.

In my this program I had to add each function in-line instead of using functions, this lead to an excessive amount of iteration through the dictionary.

# Comparison of the two implementations

The two implementations involve two different approaches. Imperative programming involves following the statements laid out in the program which change the state and the data within the program. The sequence in which these statements are presented is important because if the statements are implemented in a different order the outcome would differ. Imperative programming is native to computers as the instructions in machine code is written in imperative style. Object oriented programming differs to imperative programming, it involves the use of objects which have attributes and different methods. These attributes and methods can only be access when an instance of this object has been made, they are private variables and methods in the class. Object attributes can only be modified by the methods defined within the object class, meaning the program has to use these methods in order to update the attributes

As described above, my implementation of the imperative program involved setting out different steps and implementing the functionality without the use of methods or classes. The main data structure used in my program is a dictionary. The use of a dictionary allowed me to store and access the necessary data used to implement the calendar and its functions. Each part of the functionality had to be achieve by using the in-built functions in the dictionary library. In the my object oriented program I had three data types / objects; day, week, appointment, each having their own methods used to access and update their defined attributes. In my implementation the week class contains the main functionality of the program. It is able to have all this functionality as it inherits from the Day class and also uses the Appointment class within it defined functions. In this implementation the calendar functionality is accessed by creating an instance of week and then calling one of its in-built methods to carry out this function.

Getting all the appointments for a week in both implementations was quite similar but in the object oriented program this function could only be accessed once an instance of week was made, also there is more options for when printing due to the different string methods in the three classes.

The only difference in the print appointments for a day function is that the program iterates through a list of instances of day, allowing for easier access to its attributes and string method.

The add function in the object oriented program incorporates appending instances of the appointment class into the appointment list with the desired instance of day within the week.

For the remove method, the implementation of this was much easier than the imperative counterpart as I was able to compare a string to an attribute of an instance of appointment, instead of a string to a list.

In my opinion, for a calendar program like this, object oriented programming gives the implementation more structure and the code is presented in a clear and concise manner, making it easier to understand from a user’s point of view. The use of the classes allows for more options when creating these objects. Defining the different attributes of the objects allows for easier access and modification to data. Implementing a the calendar in an imperative manner was a much simpler and straightforward process, you had one data structure which you iterated through and modified as you wished. This provided a less structed approach than the object oriented one. In conclusion, I believe that the object oriented approach is better suited to this problem, due to the versatility the different objects provide.