Calendar program - CA341

# Imperative implementation

The files: calendar.py(The user should run calendar.py to use the imperative calendar program)

* For my imperative implementation of the calendar program I began by creating a dictionary (Line 2). After some research into which data type should hold the information of the week days and their related appointments I reached the conclusion that a dictionary is the optimum choice for this task and because I was very experienced with using dictionaries/hashmaps. The keys in the dictionary will be the weekdays Monday to Sunday as strings and the initial values will be a list I chose to make the value a list because I believe it’s the most convenient for this program and the speed of searching for an appointment within the list will be predictable.
* The first function in the program is ‘is\_valid\_format(n)’(line 12) which when supplied with n a time will verify the time passed to it is in HH:MM format and return a Boolean value accordingly it does this by using the regular expression module in python.
* The ‘dict\_str()’ (line 16) method when called prints the days of the week from the ‘week’ dictionary and the corresponding values. Using a for loop I extract the key and value pair using the ‘.join()’ method add them to a string and print them to the terminal.
* The ‘add\_app()’(Line 22) method adds an appointment to the dictionary. The user is asked to input the day the wish to add the appointment for the program verifies that this day exists as a key in the dictionary. It then asks the user to input the start and end time of their appointment after verifying with ‘is\_valid\_format(n)’ adds it to relevant key[value] in the dictionary as a string. Otherwise returns error message telling user either the time or day is incorrect
* The ‘rem()’(line 40) method asks the user to input a day it verifies if this day has any appointments then asks the user to enter the appointment they would like to remove. Throwing error message in relation to any invalid inputs.
* When the user runs the program, they are prompted with a message to choose what they would like to do add, remove, print, help, day, end. When the user inputs one of these words into the terminal they can then execute the related task which leads them to the above-mentioned methods. If the user chooses ‘day’ the program asks for a day and prints out the appointments for that day only.

# Object-oriented implementation

The files: week.py, day.py, appointment.py, main.py (The user should run main.py to use the OO calendar program)

* The week class was the first to be created. It contains a dictionary data structure which contains the days as keys and appointments as values. Different classes will interact with the week class to access its private variable week{} and add or remove appointment as the user wishes. The week class also contains a method that can be called to print the week key and value pairs.
* Next I created the day class this inherits from the week class and has day for as a private variable. This day variable is passed as a string and corresponds to the week days in the week dictionary. The day class also has a valid\_day(day) method which requires it to be passed a day and it will then return a Boolean value based on whether the day the user inputted corresponds to a key in the week dictionary.
* Next I created the appointment class. This class import the day and week objects and the regular expression module in python. The private variables are start\_time and end\_time which refer to the start and end time of the appointment. The first method is\_valid\_format(n) which gets passed a start and end time and validates they follow the following format HH:MM. The add\_app(start\_time, end\_time, d) creates the appointment when passed the start time, end time and day. It first validates that the day is correct by calling the above mentioned method from the day class and the times are correct by the is\_valid\_format(). If these all return true the method appends the provided times into the day corresponding the appropriate key in the dictionary in week. However, if the information is incorrect it returns the appropriate message to the user telling them whatever is incorrect. The last method in the class is the remove method rem(start\_time\_end\_time, d) which when supplied with the correct appointment to remove access the week dictionary and removes that value from the corresponding key.
* All of these classes are supplied with user input and utilised in the start.py file which is the main() fucntions it asks the user for inputs and calls the various objects and classes to use their methods and perfom the calendar functions. If the user types day also the program will ask for a specific day and print he appointments for that day

# Analytical comparison

The learning goals of this assignment for me was to better understand the two programming paradigms imperative and object-oriented. For the task of creating the calendar program I chose to implement both paradigms in python as I felt this will help me understand the similarities and differences much better. For the imperative approach it was simply a dictionary that contained the week days and values which would hold appointments. Then statements followed which at a very basic level through direct assignments would append and remove values from the dictionary. This approach is essentially what imperative programming is all about. The benefit of this is that it adopts a very linear approach which doesn’t require a lot of foresight and allows the programmer to basically ‘sit down’ and finish the program with very little planning. This does make the code transparent as we are dealing with a single state and for this assignment its easily readable however for large projects imperative programming beings to show its flaws. If this project was any longer it would simply mean that the block of code would start to reach enormous sizes and become difficult to debug and understand as there is no way to distinguish one part of the code from another this is where Object oriented programming excels at.

The object-oriented implementation of the program shows similarities to the imperative implementation e.g. direct assignment. However, it guards the week data structure through encapsulation, this is because the week data structure is in the week class which defines the traits of the week it can be interacted with however the structure cannot be changed and this is a benefit of object oriented programming. The day class inherits from the week class and goes further by outlining how a day behaves, for this program is simply is if that day is in the week. The appointment class is where we see the similarities between the imperative approach and the object-oriented approach. The add and remove function interact with the week class through direct assignment the difference however is that the appointment class can be instantiated and create separate instances of appointment this gives the program a level of robustness that the imperative approach doesn’t provide.

In conclusion the imperative programming paradigm is attractive due to its straightforward ward and less taught out approach which can create large powerful programmes but not very accessible or easy to debug. However due to the separate files and classes of the object oriented-approach it offers many advantages. Bugs are much easier to find as the programmer isn’t lost in blocks of code and greater security is achieved as the program only requires the start.py file to run and the rest of the files can be hidden to avoid the code being tampered with or stolen etc.