Comparing Imperative and Object-Oriented Programming

15321711 – Alexander Cahill

Both implementations of the program were written in Python, the imperative being written in python 2.7 and the Object Oriented being written in python 3.7.

# Imperative Implementation of Calender Program

In the imperative implementation the program’s state is changed by functions defined in the program. Each of these functions are called in the main by commands entered by the user. Each of the commands: ‘a’ to and an appointment, ‘r’ to remove appointments and ‘s’ to show appointments call the functions ‘add\_appt()’, ‘remove\_appt()’ and ‘show()’.

The program uses an array called ‘days’, containing seven arrays each named after a day of the week, to store the appointments. This array of arrays is a global variable and each day array is used to store a tuple containing the start time and end time for each appointment. Each day array is not sorted as new appointments are simply appended as per the user.

When adding an appointments the ‘add\_appt()’ function is called and the user is prompted to enter a day, the appointment start time and end time. In this function there are checkers to make sure the day and times entered are valid. The program then checks if the day entered has any appointments booked already, if not it simple appends the appointment to that day’s array.

If there is already an appointment on the desired day it checks if the appointment overlaps with any others, if so the user is notified, the function finishes running and the appointment is not added, if not then the appointment is simply appended to the desired day’s array.

When removing appointments the ‘remove\_appt()’ function is called and the user is prompted to enter a day. There is also a checker in this function to ensure a valid day is entered. The program checks this days array for any appointments, if there are none the user is notified and the function finishes running. If appointments are found on the day entered they are listed and the user is asked which one they would like to remove. The selected appointment is then removed from the array using ‘pop()’.

When displaying appointments the ‘show()’ function is called and all the booked appointments for the week are displayed. This is done by iterating over the ‘days’ array, checking if a day has any appointments, if it does it prints the name of the array and all the appointments contained within it.

In summary in this implementation each of the defined functions operation upon the global variable ‘days’, adding, removing or printing tuples, in each of the arrays contained in it.

# Object-Oriented Implementation of Calender Program

In the Object Oriented implementation of the program there are two classes, ‘Appointment’ and ‘Calender’.

The Appointment class has an init method taking a day, a start time and an end time, and a string method used for printing appointment objects.

The Calender class has an init method taking just an array, a string method used for printing the calender object, an add method for adding appointments to the calender taking in an appointment and a remove method for removing appointments from the calender.

In the main of the program the array that is used to initialise the calender class is defined, it is an array called ‘days’ containing seven arrays, each named after a day of the week. Each day in the ‘days’ array stores a list of appointment objects.

When adding an appointment the user is prompted to input a day, start time and end time in the main. The Calender class then attempts the instantiate a new Appointment using the ‘add’ method. If the day and times are valid and do not overlap with other appointments a new appointment object is created and appended to the Calender’s ‘days’ array.

When removing a day the ‘remove’ method in the class function is called and the user is prompted to enter a day. If the day entered is valid all the appointments for that day are listed and the user is asked which one they want to remove. The specified appointment object is then removed using the ‘del’ python function.

When displaying the appointments the main simply calls the python ‘print()’ function on calender object, doing this allows the Calender to call its ‘str’ method in which it is defined how the object will be printed. In this case it iterates through the ‘days’ list to find any day that contains appointments, then iterates through this day and calls the python ‘print()’ function on each appointment object. The appointment ‘str’ method prints the day of the appointment, the start time and the end time.

In summary; in this implementation an object containing its own variables and methods is created for each appointment which are stored in a calender object which again contains it’s own variables and methods.

# Comparison of Both Implementations

In both of the programs a main loop is used to wait for commands in the form of one letter inputs. Each of these inputs executes a function.

In the imperative implementation these functions operate on the global variable array ‘days’, defined at the start of the program. This is contrast with the OO implementation as the functions are contained within classes defined in the program and they operate on objects. For example when removing an appointment from the calender in the imperative style the function operates on the ‘days’ list and removes a tuple from a sub-list. Whereas in the OO style the function operates on the calender object then removes an instance of an appointment from a sub-list.

How the Appointments are represented differ between the two implementations. In the OO style they are represented with objects, this is useful as template can be defined in it’s class as to how appointments should be created(what parameters and variables it has), how it should be printed and what functions should be able to operate on it.

In the imperative implementation it is not so clear to see how appointments are represented, we have to look at the ‘add\_appt()’ function to see that they are being stored as tuples in a sub-list.

How the Calender is represented also differs between the two implementations. Again in the OO style the calender is an object taking a list of lists as its parameters.

Again in contrast to the imperative implementation it is not so clear to see how the calender is defined. It is implemented using a list of lists like in the OO implementation but only after looking through the code does it become clear what the functions are operating on.

In these ways the OO implementation offers many advantages over the imperative implementation:

Code is more readable as it clear what each of the classes are and what its methods do.

It also allowed me to easily define abstract data types that model the real world well eg. An Appointment and a Calender, instead of applying functions data structures.

If the program were required to be modified or have more features to added it would be easier in the OO style as its modularity allows it to be easily adapted to work as part of a larger system or modified as a change inside a class will not affect any other part of the program.