**Development**

It was during a free session that the idea of creating this project popped in my heads. I realized that my school administrators spend a great deal of time checking who is absent and who is present, then in coming up with proxies, etc. I wondered how all this worked. Also, I occasionally saw my school peons work hard walking long distances for searching teachers.

So the first step towards the commencement of the project was the understanding of how this system worked. Initially I made a Teacher class which contained teacher attributes such as her/his name, Iekly schedule, her/his status of being present, etc. Soon I developed algorithms and functions to just ‘place’ teachers in their classes during the respective sessions. Then I developed Teacher’s methods like update\_status(), which change the teacher’s position whenever a session expires and a new session starts. I took the help of ‘datetime’ module to perform two simple tasks: identifying the present weekday and calculating the number of seconds passed since the school day started. I then divided the latter by 1800 (the number of seconds in a typical session), and then incremented the result by 1 to get the ongoing session. Later I developed functions to generate proxies and identify classes which required monitoring during particular sessions.

I did these by implementing lists and dictionaries. In my program, an individual proxy is a list which contains information such as who will take the proxy, in whose place this proxy is assigned, etc. The code runs a for loop on all absent teachers’ schedules and a nested for loop over all the present teachers’ schedules. Whenever the code found that a present teacher is free, it assigned her/his proxy for the absent teacher in the parent loop. In addition to these, there are other proxies possible for the present/absent teacher during a particular session. These proxies get store in the proxy suggestions list (named s\_proxies). I used dictionaries to store classes which required monitoring. The keys are the classes and the values are lists which contain the sessions when no teacher would be present in the key classes.

Next I created the teacherSeek() function, which searches for teachers. The function runs a for loop over all the present teachers’ schedule, and whenever it finds that a teacher is present in a particular place during a particular session (which are passed as arguments), it returns the teacher instance. The find\_free\_teachers() function works similarly. This function finds free teachers during a particular session.

Finally, I made the interface of the program like a SQL shell: you type your queries about a particular class or teacher or session and the computer responds; or the computer suggests you proxies and you give commands to assign them, etc. I felt that this was the best way to keep the interface of my program simple.

**Data Files and Their Structures**

The individual teacher instances are stored as binary pickle files. These files have names corresponding to the teachers’ names. Whenever the program starts, these pickle files get loaded. There’s a separate binary file named ‘Names.pkl’, which stores the names a list containing the names of the teachers. The code runs a file loop on this list and loads the corresponding teacher instance one by one.

Next, I created a Stats object, which contains each day’s ‘statistics’. Whenever the user quits my program, the out() function creates a Stats instance and stores the day’s data (i.e., this Stats instance) into a binary file named by the date of the day. The Stats object has attributes like: the day’s proxies, lists of present and absent teachers, lists of classes which require monitoring, etc. This feature is very convenient when the user runs my program again during the same day – the program doesn’t ask for attendance once again. The day’s data is already loaded in the program, by loading the day’s Stats file. This is also useful when the user wants to view past days’ records. The displayStats() function takes a date (as a string) as an argument, and then prints the day’s ‘statistics’.