Clock in/Clock Out

- Gary Davis
- Python Free Time Project
- 1/27/2022

This project was the start of something that I plan on adding to and upgrading later. As it stands now, the project is a series of short programs that work fine individually. Ultimately the plan is to integrate them all into one larger project and have them act as functions to be called instead of separate programs that run one at a time. The individual programs are outlined below:

EmployeeClassv2.py

This is the most recent iteration of the Employee Class that I built. It contains a class function (the highlight of the program) that creates employee objects to that they can be added to/arranged in an array. This was the beginning phase of the program's life cycle. After some short testing, I realized that whenever the program starts up the array starts fresh and therefore defeats the purpose of having the program run for several days at a time. Eventually this program was replaced with the more modular idea it is today which involves storing the employee information in a database.

AddToRecords.py

This is the eldest of the programs that were written for the database idea. This function, (I figured) would be the simplest, so I started here. When I wrote the program the idea was to be able to create a database that was a text file (in this case it is Employee.txt) and simply append a new line with a new entry in it that follows the same naming conventions that the other entries do. This program accomplishes this.

Password.py

The second program that I wrote was this one. The purpose of this was to match a username to its corresponding password in the database. It required the ability to read data from the text file (in this case passwords.txt) and match the information in the name column to the data in the password column. I kept things relatively simple in the database here. I used the same names that I used in the employee.txt file and gave all the guys one password [1234] and gave all the girls a different password [abcd]. This program iterates through the list searching or the string that the user input. If it finds a match it raises a flag that signals "found it" and asks the user for the password. If it doesn't find the name, it will return that the combination of characters{name} is not in the database. [[this search algorithm needs work. It currently will grab every instance of that string and prompt the user more than once to input the password if the name appears more than once. Will be fixed in future updates]] Finally, if the user puts in the right password the program will display a “Password matched” message. If not, it will display a “password *not* matched” message.

RemoveFromRecords.py

This was the next program that was written. After I learned how to iterate through the list, I figured the next logical step was to use that algorithm to build a program that could remove entries from the list. That’s what this program does. Its relatively short compared to the other programs. This one simply iterates through the list looking for the entry that the user put in when the program started up and checks to see if it is in the database. If it doesn’t find a match then it doesn’t do anything other than let the user know that there is not a match in the database. However, if it does find a match it lets the user know that the input has been deleted. It also removes the entry from (for now) test2.txt.

EditRecords.py

This was the most complicated of all the different programs in this project. This one does exactly as the name describes. The problem though was that each record has several different parts, including a username, a password, admin rights, position, and hours. This meant that I would need to be able to update the strings on a more targeted level than just adding or removing entire lines. Instead I would need to update the characters in the strings themselves. Eventually I got it all working.

As of now, the program starts and asks the user what they would like to change. When it receives the numeric input associated with the options available, it will then ask the user who’s they would like to change. When it receives the input, the program will iterate through the [Employees.txt] database. If it doesn’t find the name that was input, It will return that the name was not found and end the program and end. If it does find the name it will replace the characters in the columns available for the name with the input from the user. After that it will rewrite the record at the bottom of the list. The same process works for all the other edit functions. “Change what” “For who?” “Are they in there?” “Update”. All of these functions check the input from the user to keep them from inputting strings that can not be added to the database. For example, the name field can not be more than 10 characters. It checks this and numeric inputs for lines like “What do you want to change” at the beginning. When a string is accepted and the necessary line has been found it updates the necessary database(s).

Employees.txt

This is just a test file that contains the class parameters for all of the employees. “Name” “position” “Admin rights” “hours worked. All of the entries are organized based on character counts.

Passwords.txt

This is a list much like the Employees list. This one just contains all of the same names that are in the Employees.txt file and one password associated with each of these.

Test.txt & Test2.txt

These two files were created to test different outputs instead of updating and potentially destroying the two primary text files.

Conclusion

As stated in the introduction the long-term plan is to create an overhead program that utilizes the different parts of these individual programs as functions. There will most probably be some sort of TKinter integration for graphical interface. I’m considering now that I may spend the next week of free time learning about either TKinter or Kotlin so that I can find a way to integrate other programming languages together to strengthen this program’s presentation and execution.