Name: Katie Savard

Date: 5/17/2023

Course: Intro to Python

**Assignment 05**

**Introduction**:

Not only did Assignment 05 further our programming knowledge, but it also required us to review and updated existing code. Assignment05 started by providing existing code that the programmer would have to build upon. This came with a new set of advantages and disadvantages. You will see in following text that the programmer had to take on a “peer review” type role to validate and update the existing code to get a fully functioning script. This assignment also challenged the programmer to use dictionaries instead of lists. The use of dictionaries required keys instead of positions do denote order and that will be apparent in the code base.

**Creating the Program:**

To start the program, I first reviewed the existing program. Because there was existing code, I had to slow down and really understand what had been done before me. The script seemed straightforward, a menu of options with some actions that we had mostly seen before. The assignment also required that some text be pre-loaded into the txt file. I went ahead and added two random tasks with a priority level to start.

Once I felt comfortable, I started programming the first set of requirements. It become clear that the prior user was missing the objFile variable. The user had strFile = ToDoList.txt, but something needed to represent the actual file. Therefore, I defined objFile = None to support.

Next, I created a series of code that converted the previously loaded entries into a dictionary and then printed them at the start of the script. It seemed redundant to include this in the script because the user could simply enter “1” to get the list again. However, this code allowed the user to see the existing list at the start of the program. This code appears like this and opens the txt file in read only mode:

objFile = open(strFile, "r")  
for row in objFile:  
 strData = row.split(',') # each position would be a single letter if not for split  
 dicRow = {"Task": strData[0].strip(), "Priority": strData[1].strip()}  
 lstTable.append(dicRow)  
 print(dicRow["Task"] + ', ' + dicRow["Priority"]) # not including this, but keeping code for reference  
objFile.close()

*Figure 1: Initial code to display existing to do list*

This section defined the keys for the dictionary throughout the entire script. These keys were “Task” and “Priority”.

After this initial set up, the program starts a while loops and goes into a series of if statements that corresponds to the menu selections. The first selection, “1”, tells the user what is currently in our Table (defined as lstTable). Because I added the existing entries to the lstTable, user selection 1 will display those same options plus anything new added by the user in the program.

Selection 2 allows the user to add new items to the list. This is very similar code to some of the projects we have already worked on. The only difference is that we are working with a dictionary instead of a list. The user’s entry gets formatted into a dictionary and then added to lstTable via the += command. I leveraged the dictionary keys from the start of the script in this loop.

Next, I started working on the code to remove a task. I thought a lot about the structure of this section. I was thinking of adding a column to lstTable the provided some type of numbering like 1, 2, 3, etc for each entry. The user would view the list and simply type the row they wanted to remove by using the number ranking. That would require me to re-structure the list table, dictionary, and txt file. Instead, I opted for a less elegant solution and asked the user to type the task name. The script would then find that task in the table. When it was found, it would remove the task. If it was not found, the script would say “not found”. Because this script was written by using a for loop, each time the program ran through a line of the table, it would say “not found” or it would find it and delete the item. For example, if the line the user was trying to remove was the third entry, the user would see two instances of “not found” before seeing “removed”. This is shown in Figure 3 below:

A screen shot of a computer

Description automatically generated with low confidence

*Figure 3: Example if the user interface when removing a task*

This section is where I learned how to remove items from table. Similar to Table.append(), I used Table.remove() to achieve this.

The final two menu options were less complex and covered material from prior Modules. Option 4 had the code save the lstTable to the objFile. In this case, I wrote code to open the txt file and save each entry. For each row in the table (for row in table:), I “wrote” to the txt file. The results of one example are shown in Figure 4 below:

A screen shot of a computer

Description automatically generated with medium confidence

*Figure 4: Example of the final txt file for a user*

After saving the user entries, option 5 provided the option to get out. This section broke out of the loop by using “break” and told the user it was ending via print(“bye!”). The program stopped and the user would have to reload the script to start over at this point.

**Lessons Learned:**

After completing the module, I prefer working with someone else’s code. While it took some upfront work to understand the current script, it was simpler and easier to build upon. I enjoyed having a framework that provided a clear outline of the functionality the script was intended to have. While the framework provided the very simple, if statement, it was a helpful starting point.

As a novice programmer, I was thinking about the pros and cons of using a dictionary vs a list. I believe that in this application, a list would have work just as well as the dictionary. The dictionary appeared to have provided the same benefits and using a key instead of position, was irrelevant in the script. I can see the need to use a dictionary when trying to define multiple columns that easier to remember by using words. 0, 1,2, etc is very hard to remember and a dictionary would alleviate that problem.

I had a bit of confusion on how to start the program. Step 1 said “load any data you have in a text file called ToDoList.txt”. To me, this meant that I would manually pre-write a couple tasks in the txt file. Then the subsequent code would build upon those manual entries. However, I debated whether to have the code write that text in, vs me manually. The code for this would be similar to the code present under option 5:

objFile = open(strFile, "w")  
for row in lstTable:  
 objFile.write(str(row["Task"]) + "," + str(row["Priority"]) + "\n")

Figure 5: Example of writing an entry to the txt file

**Conclusion**:

There are many areas of this final script that can be improved upon with further python knowledge. I know there are ways to make the script cleaner and more effective for a user. However, this type of programming knowledge is an essential foundation for what is to come in future modules.