Eric Forster

Nov. 14th, 2020

IT FDN 110 A Au 20

Assignment 05

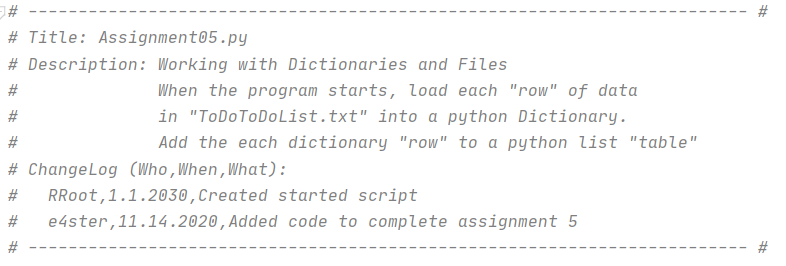
**Storing Data with Dictionaries**

**Introduction**

This week I learned how dictionaries differ from lists, and the difference between indexes and keys. Dictionaries are very handy because many data sets only include two items, the name of an item, and a value for that item. Just like other sequences of data, dictionaries can be nested inside lists to create multiple dimensions like a table. In this week’s assignment, I added on to a script started by Randal, and made it meet the goal. Like previous weeks it gives the user a menu to choose from, and helps the user create a text file full of tasks and priority levels for those tasks.

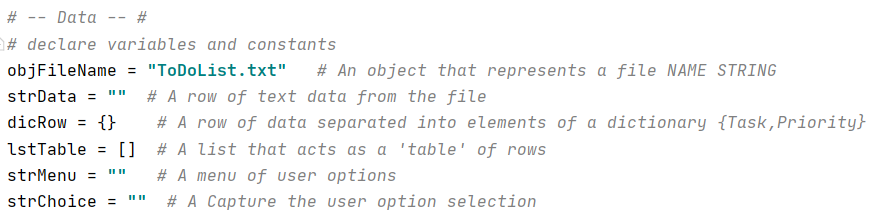
**Step 1: Header, Data, Processing**

Like previous scripts, I updated the header to start things off:



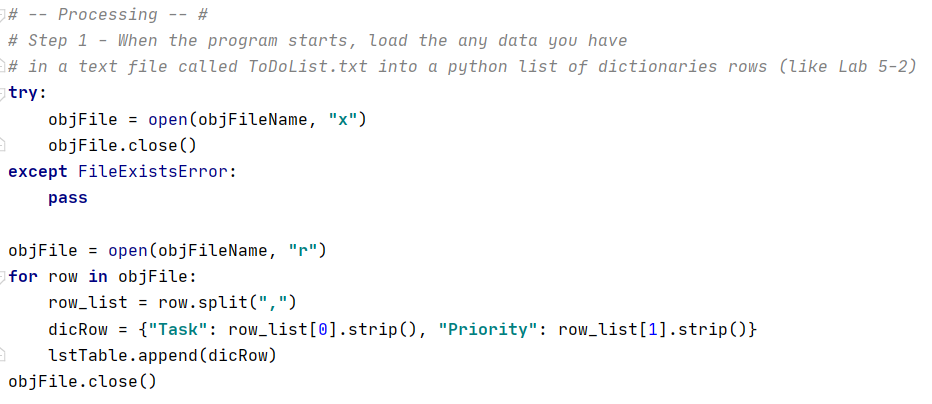
***Figure 1.1: Header from the script.***

This is the first assignment that the script is broken down into three main sections that is industry standard. Data, Processing, and Presentation (I/O). This follows a concept called ‘separation of concerns’ which means organizing the code by standard chunks that all future developers will understand. In the Data section the main variables in the script are listed and created:



***Figure 1.2: Declaring variables at the start.***

This showed what type of data and sequences I would be working with later in the program. The next part was processing data between the hard-drive and memory. The goal was to grab the data we already had stored in our ToDoList document, and store each row as a dictionary, and then store those dictionaries in a list. At first I had trouble because the file didn’t exist yet, but I used a concept call Try-Except in order to combat this. After edits, the script first tried to create the file in case it is the first execution, and then it tried to the read the file and was successful.

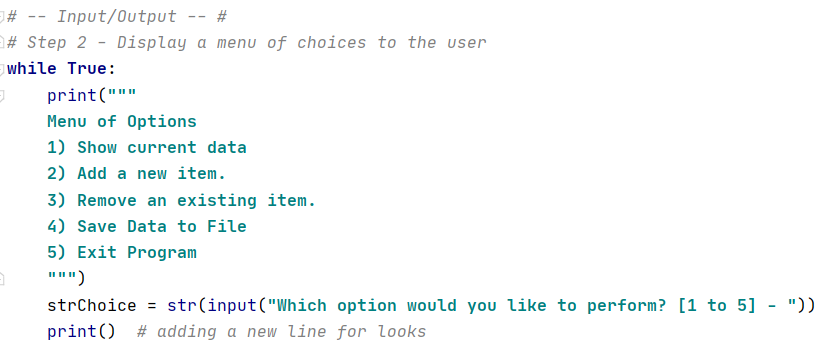


***Figure 1.3: Grabbing data from the text file.***

In the ‘for’ loop, I looked at each row in the text file and created a list of two items by splitting the string at the comma. Then I created a dictionary and put those two items as the ‘values’ for the Task and Priority ‘keys’. Then this dictionary was stored as an element in my over-arching ‘lstTable’. This list was used to store and manipulate all of the user’s data in memory while the program was running.

**Step 2: Display the Menu**

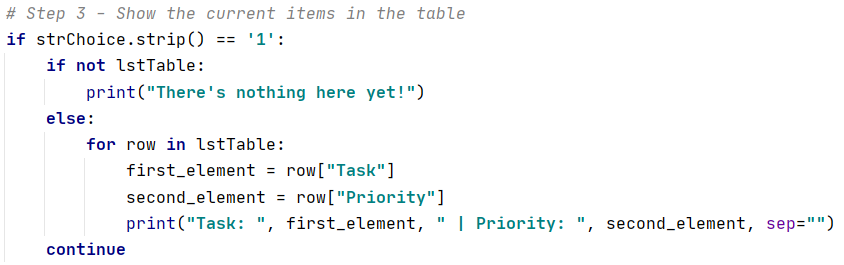
The next section started the presentation portion of the script. I started with the ‘while True’ loop, and printed the options. It then asked for the user to type and option.



***Figure 2.1: Displaying the menu.***

**Step 3: Show Current Data**

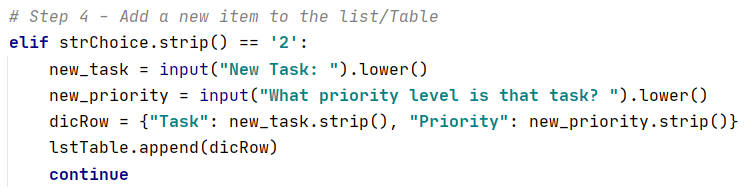
For this section I looked through each element (dictionary) in the ‘lstTable’, and pulled values from the dictionary. To do this I called out the key as the argument instead of an index value in order to locate the data I needed in the dictionary. Then I took those values and placed them in a string to print. Then the ‘for’ loop iterated through the next elements in ‘lstTable.’



***Figure 3.1: Printing current data.***

**Step 4: Add to the To Do List**

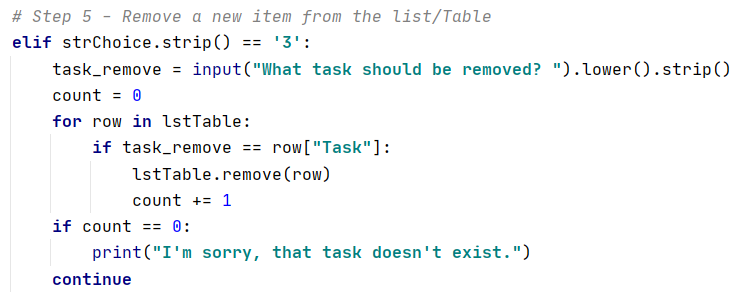
In order for the user to add a new Task and Priority to the list, I first asked the user for those values with the input() function. Next I created a short dictionary with ‘Task’ and ‘Priority’ as the keys, and inserted the user’s input as the respective variables. Then I simply appended this dictionary as a new element in my lstTable. By using the .strip() and .lower() methods, I was more likely to have standardized strings.



***Figure 4.1: Appending new data to the list.***

**Step 5: Removing an Item**

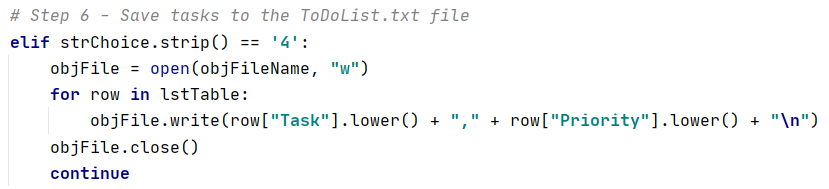
For this choice, I started by asking the user what item they would like to remove. This is essentially asking to type a ‘key’ from the current data set. Once I had that string from the user, I compared it to each Task value in each dictionary within the list. If it found a match, it removed the line item by using the .remove() method. If it didn’t find a match, it let the user know it couldn’t find the string they typed. I did this by raising a counter variable if it found a match, and then checking to see if the counter was zero after the ‘for’ loop was done.



***Figure 5.1: Removing a line item.***

**Step 6: Save Tasks to the Text File**

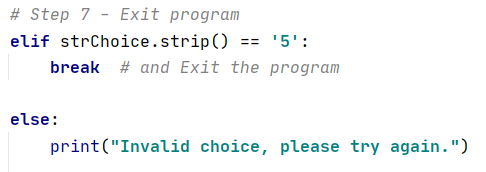
For this user choice, I started by opening the text file in ‘write’ mode. This cleared all the data in the file, however all of the data was stored in memory in the lstTable I created. This avoided adding duplicate items to the file. I used another ‘for’ loop to iterate through each dictionary in my list, and extracted the Task value, the Priority value, and then placed those values into a string which I wrote into the file using the .write() method. Once the ‘for’ loop was done, I closed the text file for later use.



***Figure 6.1: Saving to the text file.***

**Step 7: Exiting the Program**

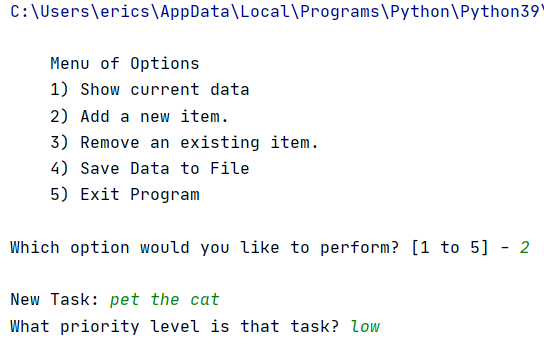
Last but not least, when the user chose ‘5’ the script broke out of the ‘while’ loop and the program ended. I also added an ‘else’ statement at the end in case they entered an invalid number.



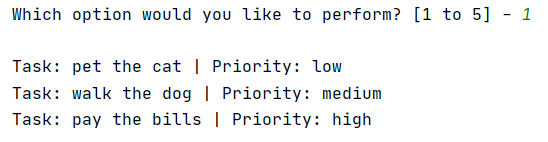
***Figure 7.1: Exiting the script.***

**Step 8: Running the Program**

Here was the program in PyCharm. It started by displaying the menu, and I added three rows of data to the to do list.

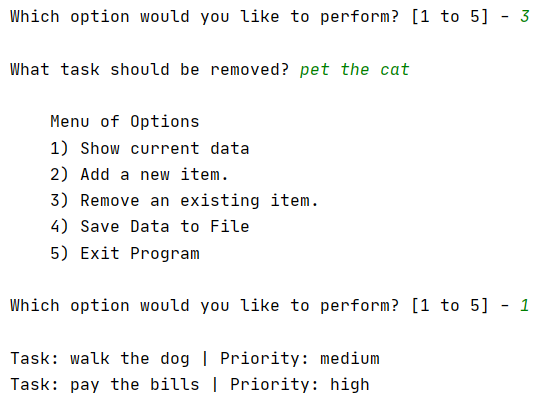


***Figure 8.1: Menu in PyCharm.***



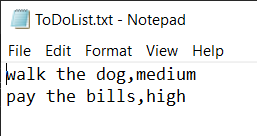
***Figure 8.2: Displaying current data.***

Once I chose option ‘3’, and typed in a task to remove, the item was gone from the list when I chose option ‘1’ again.



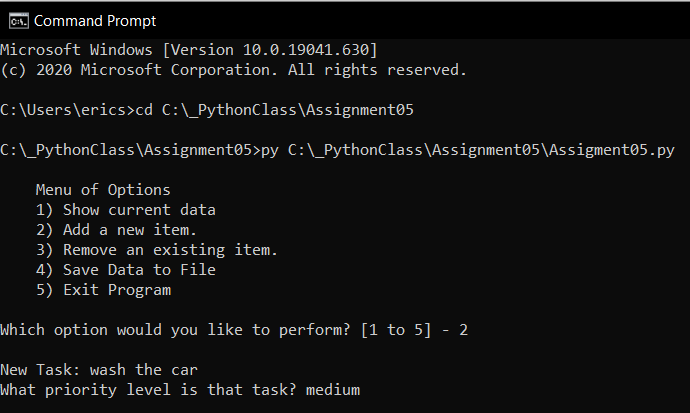
***Figure 8.3: Removing an item.***

After choosing option ‘4’ and then exiting the program, it added the data that was stored in memory to the text file I created.



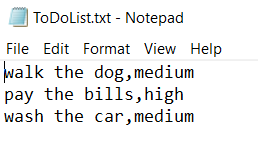
***Figure 8.4: Data saved to the text file.***

Here is the program running from the command prompt:



***Figure 8.5: Adding item from the command prompt.***

As seen below, once I chose option ‘4’ from the command prompt, the data was added to the text file.



***Figure 8.6: Item added to text file from command prompt.***

**Summary:**

Combining dictionaries with lists opens up a wide range of data storage capabilities. The toughest part of this assignment was wrapping my mind around calling values within a dictionary, within a list. Once that was understood, it was very easy to gather and save data with the text file. While writing this assignment I could see that a function could have streamlined some of these operations, such as opening and closing files etc. All of that processing could have been called by a single object name. Also, it was difficult to use someone else’s variables. In a script much larger than this, I can see how incorrectly named variables could drive a future developer mad. I am looking forward to expanding on all of this knowledge next time.