Linda Shmait
8/9/2022
Foundations Of Programming: Python
Assignment 05
https://github.com/lzs425/IntroToProg-Python

To Do List

Introduction

In this article, I will review the process to create a Python script that will allow a user to choose several options from a menu to view, add, remove, and export items to a to do list. This script will continue running through options until the user selects that they would like to close. This project uses dictionaries, lists, file writing, and associated methods, along with print() and input() functions. When run, this code should look like Figure 1 below, with variations based on menu options and entries.

```
C:\_PythonClass\Assignment05>python "C:\_PythonClass\Assignment05\Assigment05_Starter.py"

    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] - 1

reading,1
homework,2
Cleaning,3
```

Figure 1. Portion of Completed Script Display

Script

The script consists of data, processing, and input/output sections. As we were provided the template, most of our code was added to the input/output section. The first section is the data section, in which we declare the variables we will use in the code. The next section reads data from a text file and load it into a list of dictionary items, display the menu of options to the user, and store their response. The last section, the input/output, allows the user to select a series of actions that the script will carry out based on the menu section.

Data

The data section houses the variables that will be used in the script. By declaring them at the start, someone looking at the code will have a general idea of what the script will do. It also allows us to provide initial values and determine type. For example, the variable “dicRow” was declared as an empty dictionary, as denoted by the curly brackets. In this case, many of the variables were declared already. I added strTask and strPriority as variables whose inputs would be added to the list of tasks.

Processing

The processing section of this script pulls data from an external ToDoList.txt file and loads it into a table. This table will have rows made of dictionaries, similar to a list but leveraging key subscripts instead of indices, as seen in Figure 2 below. We need the code to read all lines of the file, which is why after I get the file using the open() function in the read format, I use a for loop to cycle through all lines of data and create a new table.

Within the loop to add the text file data to a table of dictionaries, I am separating the task and priority using the split() function for data stored in each row of the text file in line 32. I then store it in a dictionary in line 33 and creating the keys "Task" and "Priority". The strip() function is used on the lstRow index 1 to get rid of any carriage returns. For each dictionary I create, I then append it to the lstTable using the append() function, and the for loop will continue this cycle until all rows of data in the text file have been read and my lstTable has all the data in lines of dictionaries.

```
26  # -- Processing -- #
27  # Step 1 - When the program starts, load the data you have
28  # in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
29  # TODO: Add Code Here
30  objFile = open("ToDoList.txt", "r")
31  for row in objFile:
32      lstRow = row.split(",")
33      dicRow = {"Task": lstRow[0], "Priority": lstRow[1].strip()}
34      lstTable.append(dicRow)
35  objFile.close()
```

Figure 2. Loading Data into a List of Dictionaries

Input/Output

Much of the structure of the Input/Output section was provided. Its overarching structure is a while loop with nested if statements. While(True) in this case would continue to run until it gets to a break. Within that while loop, the code displays the menu using the print function, captures the user input, and based on that routes to a particular section of code by way of a series of if/elif/else statements. Only Option 5 has the break option, so the user can continue looping through the options until they choose to exit.

Option 1: Show Current Data

When the user inputs 1, they are selecting to print any existing data we have loaded in the table lstTable, as seen in Figure 3. As we have in previous assignments, I used a for loop to go through each row and print. The advantage of dictionaries is evident in line 54 of Figure 3, where within the print function I am able to use the key, providing more meaningful information as to what I want to print, rather than an index value that requires more knowledge of the original table to understand. Once this is complete, line 55's "continue" will put the user back to the main menu to input their next option.

```
51      if (strChoice.strip() == '1'):
52          # TODO: Add Code Here
53          for row in lstTable:
54              print(row["Task"] + "," + row["Priority"])
55          continue
```

Figure 3. Printing a table of dictionaries

Option 2: Add a New Item to the List

If the user wants to input a new task and priority level into the to do list stored in lstTable, they will select option 2 from the menu. The script will then route to the code displayed in Figure 4. I use two variables previously declared in the Data section, strTask and strPriority, and setting them equal to the inputs for task and priority respectively in rows 59 and 60. I can then store them in dicRow in line 61, and append the newly created dicRow to lstTble in line 62. It is convenient again to be able to reference the key, and ensure that I am inputting the correct variables with the corresponding keys.

```
56      # Step 4 - Add a new item to the list/Table
57      elif (strChoice.strip() == '2'):
58          # TODO: Add Code Here
59          strTask = input("Task: ")
60          strPriority = input("Priority: ")
61          dicRow = {"Task": strTask, "Priority": strPriority}
62          lstTable.append(dicRow)
63          continue
```

Figure 4. Appending a Dictionary Row to a Table

Option 3: Remove an Existing Item

If the user wants to remove a task and enters '3' for option 3, the elif statement in Figure 5 on line 65 will be true. The code will prompt an input from the user for the task they want to be removed, stores it in a variable, and then a for loop goes through the table of to do items and checks for matches.

The lower() function is used to ensure uniformity of letter case and improve the chances of an intended match. The remove() function is responsible for eliminating the matching item to the user input. Because I have the print statement within the loop, I use a true/false if statement based on the variable Found to avoid printing the 'not found' note for every row the for loop goes through. The user will then get one message that the input was either removed or not found, and then looped back to the menu using continue.

```
64      # Step 5 - Remove a new item from the list/Table
65      elif (strChoice.strip() == '3'):
66          # TODO: Add Code Here
67          strItem = input("Task to Remove: ")
68          Found = False
69          for row in lstTable:
70              if row["Task"].lower() == strItem.lower():
71                  lstTable.remove(row)
72                  Found = True
73                  print(strTask + " has been removed from the list.")
74                  continue
75          if Found == False:
76              print(strTask + " not found.")
77          continue
```

Figure 5. Removing an Item from Tasks

Option 4: Saving Tasks to the ToDoList File

In similar processes to our previous assignment, the user can choose to store their data in a text file. The code for this process first opens the file in write mode as seen in line 81 of Figure 6 below. The for loop that follows then turns the dictionary rows into strings separated by a comma, adding a carriage return. This transformation back into a string can be seen in line 83, putting the data in the format that the code in the Processing section basically undoes to return it to a dictionary.

```
79      elif (strChoice.strip() == '4'):
80          # TODO: Add Code Here
81          objFile = open("ToDoList.txt", "w")
82          for dicRow in lstTable:
83              objFile.write(str(dicRow["Task"]) + "," + str(dicRow["Priority"]) + "\n")
84          objFile.close()
85          print("Your data was saved to ToDoList.txt.")
86          continue
```

Figure 6. Saving Tasks to the Text File

Option 5: Exit

The last choice the user has in the menu is to exit the program. This is very similar to Assignment 04's exit logic. Using an input(), the user is prompted to confirm the exit request, and if they enter 'y' or 'Y', the if statement logic and lower() will route the script to the break, ending the while loop that started the Input/Output section.

Testing

While I was testing each section of the script as I went, my last check is to run through the program in Command Prompt as well as PyCharm. Once I changed the directory to Assignment 05's location, I ran through the script in Command Prompt, an example of which is in Figure 7 below. Iverified that all options performed as intended, and checked the ToDoList.txt file for the correct data.

```
Command Prompt
Microsoft Windows [Version 10.0.19043.1826]
(c) Microsoft Corporation. All rights reserved.

C:\Users\lshma>cd C:\_PythonClass\Assignment05

C:\_PythonClass\Assignment05>python  "C:\_PythonClass\Assignment05\Assigment05_Starter.py"

    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] - 1

reading,1
homework,2
Cleaning,3

    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] - 2

Task: Writing
Priority: 4

    Menu of Options
    1) Show current data
    2) Add a new item.
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

Which option would you like to perform? [1 to 5] - 3

Task to Remove: cleaning
cleaning has been removed from the list.
```

Figure 7. Command Prompt Testing

Summary

This script used dictionaries to allow data to be referenced by key instead of index, and the strip() function to remove extra spaces and carriage returns, as well as using the functions and methods from previous assignments. While/break, for loops, and if statements were key to the structure of this script as well. In all, the script ToDoList will allow a user to view, add, remove, and save a to do list with priority level.

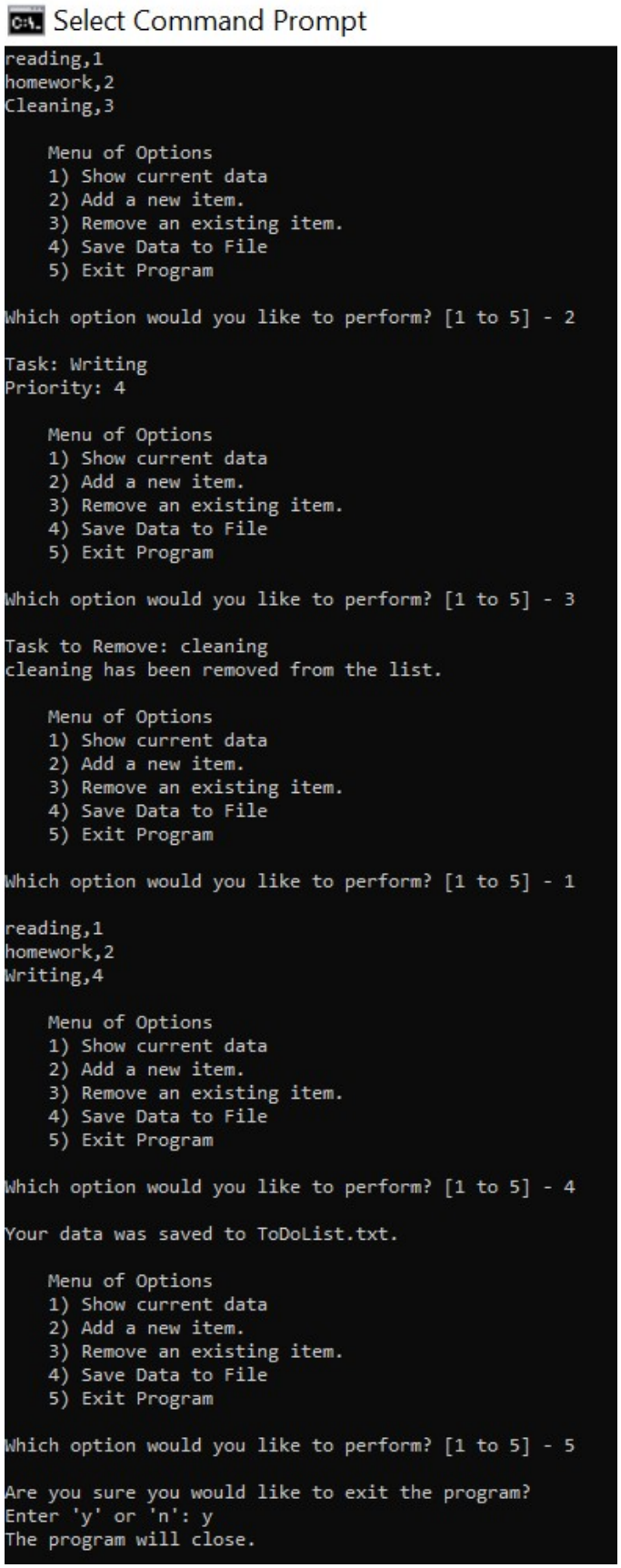
Select Command Prompt
reading,1
homework,2
Cleaning,3
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program
Which option would you like to perform? [1 to 5] - 2
Task: Writing
Priority: 4
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program
Which option would you like to perform? [1 to 5] - 3
Task to Remove: cleaning
cleaning has been removed from the list.
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program
Which option would you like to perform? [1 to 5] - 1
reading,1
homework,2
Writing,4
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program
Which option would you like to perform? [1 to 5] - 4
Your data was saved to ToDoList.txt.
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program
Which option would you like to perform? [1 to 5] - 5
Are you sure you would like to exit the program?
Enter 'y' or 'n': y
The program will close.

Figure 10. Test Results from Command Prompt

Summary

I was able to create a script that provides a menu of options to a user, routes them based on their selection to take input and store it in a table, displays a table, or writes the data to a file. I added some messages to notify the user of bad entries, as well as notify them that they are exiting the program right before the loop breaks. I then tested the script in both PyCharm and the Command Prompt to ensure that it was running as intended.