Josh Embury
11/10/2019
Foundations of Programming: Python
Assignment05

Working with Lists and Dictionaries

Introduction

Before diving into the scripting, it was important to review the resources recommended by Prof. Root. This included his Module 5 material video and class notes. Additionally, he provided links to external webpages and YouTube videos detailing the same information but presented in a different way. This variety of material was useful in understanding the use of lists, dictionaries, and their use in writing data to a file while interacting with a user. This document provides an outline of the steps required to write a script necessary to allow a user to write and interact with data stored in a file using Python.

The goal of this assignment was to practice editing a script created by another developer. Additionally, the goal of the assignment was similar to the previous two: interact with data stored in a text filed and handle the user's inputs in interacting with the data. To complete the assignment, dictionaries were used along with lists to create a table of values, which acted as the temporary storage mode between the user's inputs and the data stored in the text file.

Importing Data from a File

The program developed in this assignment is a To-Do List manager. The first step of developing the script was to import data from a file. This was done by, first, checking to see if there is a file with data in it. If there was no data in the ToDoList.txt file, the user was then prompted accordingly. Alternatively, if there was a previously saved set of tasks and priorities in the To-Do List file, the file was traversed in a for loop. Going line by line, each row of the file was converted into a list, with elements split by a comma. Then, this list was converted into a dictionary and given the appropriate header for each index of the list (ie the first index was given "Task" and the second, "Priority"). Finally, this dictionary was appended to the existing list "table" to tabulate the rows of data. By doing this, I was able to create a list of dictionaries which can be used as a table of data to temporarily store the existing data, and use it when interacting with the user as described in subsequent sections of this document. The code for this section of the program can be seen in Figure 1.

```
# -- Processing -- #
# Step 1 - When the program starts, load the any data you have
# in a text file called ToDoList.txt into a python Dictionary.
print("Loading existing To Do List...")
File = open(objFile, "a")
File = open(objFile, "r")
if not File.read():
    print("Current To Do List empty.")
else:
    File = open(objFile, "r")
    for row in File:
        strData = row.split(",")
        dicRow = {"Task": strData[0], "Priority": strData[1].strip()}
        lstTable.append(dicRow)
    print("Load successful.")
File.close()
```

Figure 1: Script for importing existing data

Displaying Existing Data

The program provides the handy functionality of displaying the current information stored in the To-Do List. This is implemented by first checking to see if the list is empty (if so, alert the user as such) and then traversing through the list table and printing the contents of each row. The script used for implementing this functionality can be seen in Figure 2.

```
    # Step 3 - Show the current items in the table
    if (strChoice.strip() == '1'):
        if not lstTable:
            print("To Do List is Empty! Please enter tasks and priorities!")
        else:
            for row in lstTable:
                print(row)
        continue
```

Figure 2: Script for displaying existing data

Adding and Removing Data

Next the functionality of adding and removing data to the existing To-Do list was implemented. After a main menu was displayed to the user, giving options of how they'd like to proceed, when selecting the "Add New Task" feature, the program asks for an input of a Task and a Priority. This is simply done using the Input() function. These two inputs are then added to a new dictionary with the appropriate headings and are appended to the list table as described in the previous section. Similarly, when the user selects to "Remove Data from List," they are prompted for input as to which task they'd like to remove. The program then traverses over the list table until the user's input matches the value stored under the Task header, at which point the Remove() function is utilized to get rid of the appropriate task. If, however, the task indicated by the user doesn't exist in the current list table, the user is prompted appropriately. The portion of the script used to provide the functionality of adding and removing data can be seen in Figure 3.

```
    # Step 4 - Add a new item to the list/Table
    elif (strChoice.strip() == '2'):
        lstRow = ["", ""]
        lstRow[0] = input("Enter new task: ")
        lstRow[1] = input("Enter priority: ")
        dicRow = {"Task": lstRow[0], "Priority": lstRow[1]}
        lstTable.append(dicRow)
        continue
    # Step 5 - Remove an item from the list/Table
    elif (strChoice.strip() == '3'):
        currentLen = len(lstTable)
        removal = input("Which task would you like to remove? : ")
        for row in lstTable:
            if row["Task"] == removal:
                lstTable.remove(row)
        newLen = len(lstTable)
        if newLen == currentLen:
            print(removal + " does not exist in To Do List")
        else:
            print("Task removed!")
        continue
```

Figure 3: Script for adding or removing data to/from the To-Do List

Writing Data to a File

In the main menu, the user is allowed to select an option for “Writing the Data to a File.” When selected, the information stored temporarly in the list table is then written to the ToDoList.txt file saved in the same location as the program. This is implemented by opening the file with the write (“w”) option of the Open() function. Next, the list table is traversed over with a For loop, row by row, and, using the write method, each row is written to the file. After the For loop is completed, the file is then closed and the user is prompted that the data has been written to the file. The portion of the script created for implementing this functionality can be seen in Figure 4.

```
    # Step 6 - Save tasks to the ToDoList.txt file
    elif (strChoice.strip() == '4'):
        File = open(objFile, "w")
        for row in lstTable:
            File.write(row["Task"] + ", " + row["Priority"] + "\n")
        File.close()
        print("Tasks written to file!")
        continue
```

Figure 4: Script used to save the current data to a file

Putting it all Together

When running the program, the execution is done with a while loop so the user can perform any of the operations shown in the main menu until the user selects the “Exit” option, at which point the program breaks the loop and terminates. Sample executions of the program can be seen in Figures 5 and 6.

```
Loading existing To Do List...
Load successful.

    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

{'Task': 'Work', 'Priority': '1'}
{'Task': 'Go For a Run', 'Priority': '3'}
{'Task': 'Fantasy Football', '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
```

Figure 5a: Sample execution of program in PyCharm

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

Enter new task: Homework
Enter priority: 2

    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

Tasks written to file!

    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

Exiting program!
```

Figure 5b: Sample execution of program in PyCharm

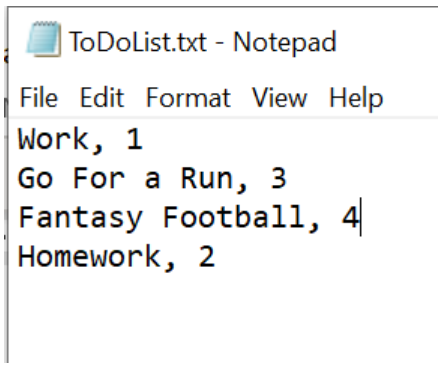

Figure 5c: Resultant text file after sample program execution.

```
C:\_PythonClass\Assignment05>python Assigment05_Starter.py
Loading existing To Do List...
Load successful.

    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

{'Task': 'Work', 'Priority': '1'}
{'Task': 'Go For a Run', 'Priority': '3'}
{'Task': 'Fantasy Football', 'Priority': '4'}
{'Task': 'Homework', 'Priority': '2'}

    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

Which task would you like to remove? : Fantasy Football
Task removed!

    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
```

Figure 6a: Sample execution of program in command line.

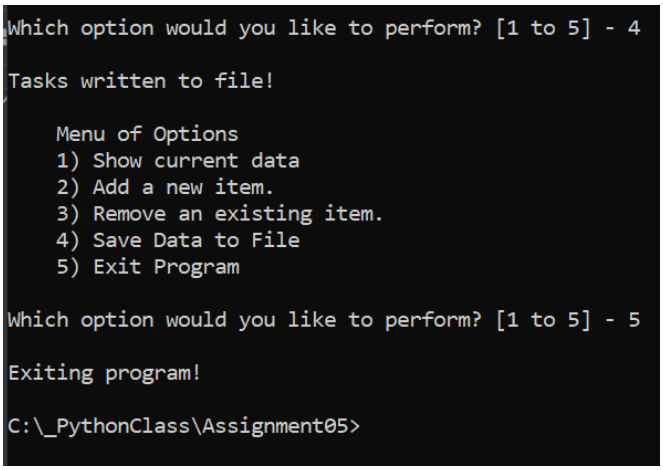

Figure 6b: Sample execution of program in command line.

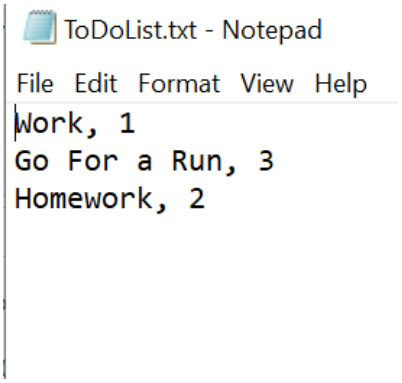

Figure 7: Resultant text file from sample execution.

Summary

This document demonstrated the process for editing a script to handle a user's modifications to a To-Do List stored in a text file. This was implemented using lists and dictionaries in order to efficiently store the data temporarily before ultimately writing the information to a text file. As a note, it was important to keep the code added to the .py file as organized as possible in order to facilitate good collaboration with the author of the script. To do so, meaningful comments were added, and code was broken up into logical sections within the script. Unfortunately, functions were not allowed to be used in the implementation of this program. If functions could be added, further organization would be added and redundancy of code would be minimized.