

Chloe Wei
May 14, 2020
IT FDN 100 A
Module 05 Assignment 05

GitHub: <https://github.com/cvarw/IntroToProg-Python>

To Do List

Introduction

As with last week's Home Inventory assignment, this week's script will read from and write to a text file called, 'ToDoList.txt'. The data will be stored as a list that will be modified via the user interface, UI, then stored back into the text file. Unlike previous assignments, I will not have an example of the UI nor the text file but must use "Task" and "Priority" as the dictionary keys. All code must be inserted into the "TODO" section of the script and functions are used at this time.

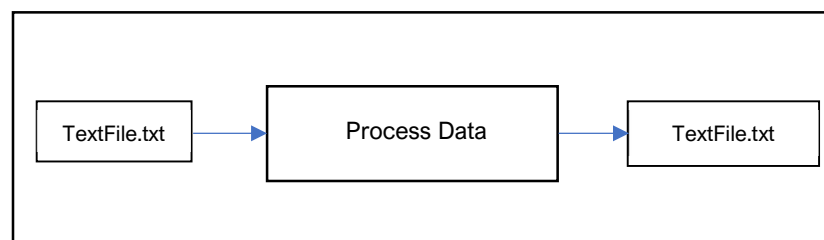


Figure 1: To Do List

Formulating a Plan and Solution

After reviewing this week's lecture, reviewing the web sources for Lists and Dictionaries, and working through the examples in the Python Programming textbook, I knew I had to first write data into a text file for testing out the code. After understanding what the given data variables represented, I worked my way down the code by assigning value to the list variables, `lstTable` and `dicRow`, in figure 2. I added the text file, 'ToDoList.txt', to the same work folder then added data via the 'ToDo' script.

```
strData = open(objFile, "r")
for row in strData:
    lstTable = row.split(",")
    dicRow = {"Task": lstTable[0], "Priority": lstTable[1].strip()}
    lstTable.append(dicRow)
strData.close()
```

Figure 2 Read data from text file

To do this, I referred to last weeks 'HomeInventory.py' file and changed the variables to the given declared variables in the Data section. Then I looked online for how to separate the data based on the keys and found on the site: <https://docs.python.org/3/tutorial/datastructures.html>, more information on how to manipulate lists and dictionaries. Aesthetically, I prefer short and

slim lines of code and the approach to loop through the dictionary worked well for me. So instead of assigning the dictionary as a row to a variable, I used the letters, 't' and 'v' to represent the two keys associated with the index of the list read from 'ToDoList.txt' (figure 3 below).

```
strData = open(objFile, "r")
for row in strData:
    t, v = row.split(",")
    dicRow = {"Task": t, "Priority": v.strip()}
    lstTable.append(dicRow)
strData.close()
```

Figure 3 revised read data from file

Now that I have the dictionaries (dicRow) stored as a list to the variable lstTable, I can now proceed to the Input/Output, I/O, portion of the script.

Using the given menu, pseudo-code, last and this week's notes on lists, files, and dictionaries, I used the for-loop to work through the lstTable and print out the rows to the UI as seen in figure 4.

```
for row in lstTable:
    print("\t", row["Task"], ", ", row["Priority"])
```

Figure 4 Code for generating current list

To visually remove the braces and keys from the list of dictionaries when displaying data in the UI, I just pulled the values from each row in the lstTable. Figure 5 has a portion of sample data added through numerous tests.

```
pseudocode , 5
code , 3
test , 3
debug , 3
test , 3
GitHub , 5
```

Figure 5 Data from file

I learned from a fellow classmate last week (and in the recent live sessions), that input variables were not necessary for creating lists and decreases extra typing. However, I prefer assigning inputs to variables, so the new task and its priority level were associated with the local variables, 'strT' and 'strP', respectively. Using list's built-in append-function, the new task and associated priority ranking were then added to the end of the lstTable stack as a new dicRow. To be sure it is stored how I wanted it, I used the for-loop to display the updated list and commented out later (figure 6).

```

strT = input("Enter your new task: ")
strP = input("It's priority [1 to 5]: ")
lstTable.append({"Task": strT, "Priority": strP})
for row in lstTable:
    print("\t", row)

```

Figure 6 Add new data

The next item to code for the 'ToDoList' script is to enable to the user to remove the new entry. At first, I considered a way of finding the end of the stack and removing the most recent `dicRow` (figure 7). But after watching this week's live session, I learned there was a more effective way of removing dictionaries from the `lstTable` (figure 8).

```

remItm = lstTable[len(lstTable)-2:]
eleItm = remItm[0], remItm[1]
lstTable = [ele for ele in lstTable not in eleItm]
print("\t" + lstTable[0] + " - " + lstTable[1] + "\n")
print(" Removed new item from the ToDoList.")

```

Figure 7 Initial code to remove item

Because `lstTable` acts as a collection of dictionaries, Randal's solution to search through each dictionary and remove the match to the input made more sense that my initial solution just made more sense. His code allows not just the new task & priority level to be removed, but previous task stored in the list as well! The code (figure 8) will display the task that was removed and if it is not in the list, will display that the task is not in the list and return to the menu of options.

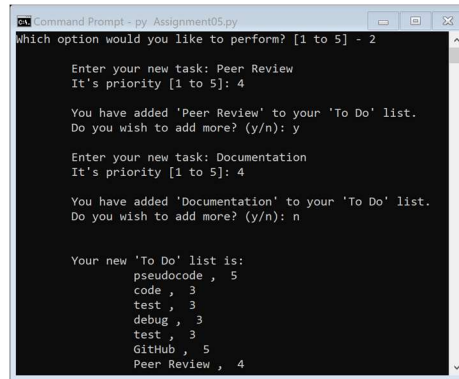
```

strI = str(input("What task do you want to remove: "))
for row in lstTable:
    if row["Task"].lower() == strI.lower():
        lstTable.remove(row)
        print("Task '" + strI + "' was removed from your list.")
    if row["Task"].lower() != strI.lower():
        print("\tTask is not in 'To Do' list.")
    else:
        continue
# for row in lstTable:
#     print("\t\t", row["Task"], ", ", row["Priority"])
continue

```

Figure 8 Improved removal of task

Taking my classmates advice to heart, I went ahead and augmented the code to give proper feedback for the adding and removal of data. As you can see in the 'Add new item...' section of the code, I put the retrieval of new data into a loop (figure 9). After a new task has been added, the UI will prompt if there is an additional task to enter. If not, then it will break out of the loop and print the new data list followed by the main menu prompt.



```

Command Prompt - py Assignment05.py
Which option would you like to perform? [1 to 5] - 2

Enter your new task: Peer Review
It's priority [1 to 5]: 4

You have added 'Peer Review' to your 'To Do' list.
Do you wish to add more? (y/n): y

Enter your new task: Documentation
It's priority [1 to 5]: 4

You have added 'Documentation' to your 'To Do' list.
Do you wish to add more? (y/n): n

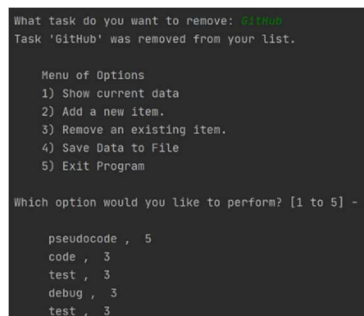
Your new 'To Do' list is:
pseudocode , 5
code , 3
test , 3
debug , 3
test , 3
GitHub , 5
Peer Review , 4

```

Figure 9 Adding new item Command prompt

Once (all) the new data has been added to the list, the UI will display the new task as well as the updated list.

Instead of removing multiple items at once, I decided to stay with removal of one at a time. Just as an insurance against accidentally deleting the wrong task.



```

What task do you want to remove: GitHub
Task 'GitHub' was removed from your 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

pseudocode , 5
code , 3
test , 3
debug , 3
test , 3

```

Figure 10 PyCharm removal of one task

Now you will get feedback of what item was removed, which you could add it back to the list if that was a mistake or just move on to saving the data.

Now that the 'ToDoList' is able to open and store the data from the file to memory, display the stored data and new data entered, add and remove data, I can now address how to store the data back to the associated file. Writing back to the file is somewhat universal for data collections, so I used the same methods and loops for the Home Inventory script and our notes. However, instead of referencing the indices of the list, I used the keys of the dictionary to determine how the data will be stored. I also attached the newline character to the "priority" key instead of a separate string element. This make it easier to strip the character when importing the data back into memory through the "ToDoList" script and eliminated an unnecessary element for each dictionary stored.

```

strData = open(objFile, "w")
for row in lstTable:
    strData.write(str(row["Task"]) +
                  "," + str(row["Priority"] + "\n"))
strData.close()
print("All tasks saved to ToDoList.txt!")

```

Figure 11 Save data to file

Summary

Through this week's assignment, I was able to utilize PyCharm's debugging tool and perform tests through the debugging console. Using the debugger along with following the steps from the starter code, helped make each selection of the menu manageable. There were times, that I would need to create new python files in PyCharm to work through how to add and remove dictionaries while keeping the integrity of the main script intact. As well as how to add the list of dictionaries back to the file. I found writing the data back to a file the most difficult aspect of lists and dictionaries. I do feel I had some growth understanding the relationship of dictionaries to lists as well as the flexibility of lists for data manipulations as well. After working my way towards the write to file, I went back to address the appearance of the display of list data and other potential modifications to the code. I prefer Github when it comes to storing code. If I only have my ipad available, I can still make additions or changes to code regardless of the OS. It was a nicer way to share our data.