

Seb Clendenning-Jimenez  
 Foundations of Programming: Python  
 November 14<sup>th</sup>, 2022  
 Assignment 05  
<https://github.com/scljimenez/IntroToProg-Python>

## To-Do List Script: Taking User Input into Dictionaries and Lists

### 1.1 Introduction:

This paper discusses the To-Do list script. It presents the user with a list of menu options, allowing them to add, remove, and print out data. Using dictionaries and lists, the data is saved to rows inside a table. The `.read()` function allows existing data to be loaded from the `.txt` file into the script and the `.write()` function writes and saves the user input data from the script into the `.txt` file.

### 2.1 Declaring Variables:

The original variables in the starter code file are empty strings, lists, and dictionaries. The empty dictionary **dicRow** is modified so that it contains two keys, 'task' and 'priority', which can then be used to access user input data later. **objFile** is set to the `ToDoList.txt` file, which will be used to store any existing data before the script is run and any data that the user has added when they run the script. This code can be seen in **Figure 1**:

```

14  # -- Data -- #
15  # declare variables and constants
16  objFile = "ToDoList.txt" # An object that represents a file
17  strData = "" # A row of text data from the file
18  dicRow = {'Task', 'Priority'} # A row of data separated into elements of a dictionary {Task,Priority}
19  lstTable = [] # A list that acts as a 'table' of rows
20  strMenu = "" # A menu of user options
21  strChoice = "" # Captures the user option selection

```

**Figure 1** — Variables for this script, including the dictionary with two keys.

### 2.2 Loading in Existing Data:

To load in any existing data, the variable **readFile** was created, which opens the `.txt` file and reads it. Since any existing task and its priority are separated by a comma in the `.txt` file, the for-loop loops through each row of data and splits those values whenever it reaches a comma. This code can be seen in **Figure 2**:

```

23  # -- Processing -- #
24  # Step 1 - When the program starts, load any data you have
25  # in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
26  readFile = open(objFile, 'r')
27  for row in readFile:
28      lstRow = row.split(',')
29      dicRow = {'Task':lstRow[0].strip(), 'Priority':lstRow[1].strip()}
30      lstTable.append(dicRow)
31  readFile.close()

```

**Figure 2** — Initial for-loop that loads in existing data from `.txt` file and appends it to an empty table.

The **lstRow** variable stores each comma-separated value loaded in from the original .txt file. Inside the for-loop, the variable **dicRow** has been re-written so that the 0<sup>th</sup> value of each comma-separated value from **lstRow** is assigned to the 'Task' key of the dictionary, while the 1<sup>st</sup> value is assigned to the 'Priority' key. The values from the .txt file now form a dictionary pair, for example: {'Task': 'read book', 'Priority: low'}.

The dictionary row is then appended to the empty list, **lstTable**. Appending **dicRow** to the **lstTable** allows existing data to be loaded into the script and be accessed should the user want their previous data printed out to them. Once the for-loop has read and loaded in all existing data in the .txt file, the .txt file and its read function is closed.

### 2.3 Printing Existing Data & Adding New Data:

The user menu allows five options: show the current user data, add new tasks, remove existing tasks, save data, and exit the script.

The string variable, **strChoice** is set to take in user input, asking them which option of the five they would like to perform. If **strChoice** equals the string '1', then a for-loop within that if statement is run. This for-loop checks the previous list, **lstTable**, to see if there is any data stored in it. If so, each row is printed out to the user. This can be seen in **Figure 3**:

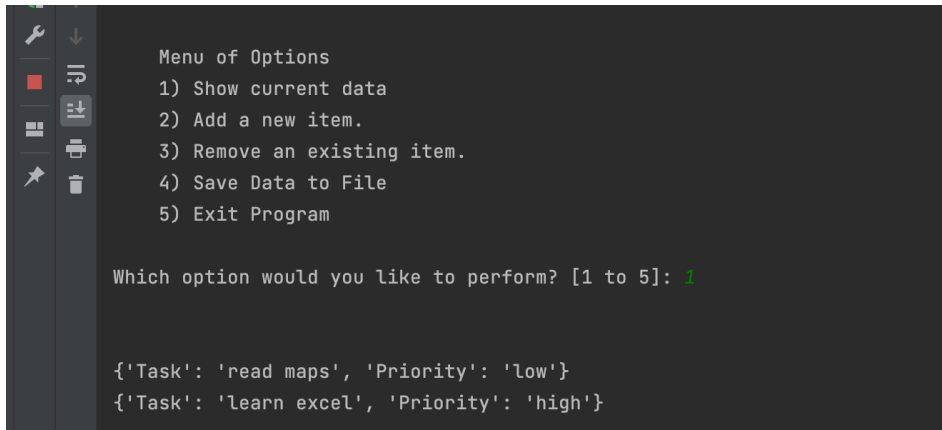
```

32  # -- Input/Output -- #
33  # Step 2 - Display a menu of choices to the user
34  while (True):
35      print("""
36          Menu of Options
37          1) Show current data
38          2) Add a new item.
39          3) Remove an existing item.
40          4) Save Data to File
41          5) Exit Program
42      """)
43      strChoice = str(input("Which option would you like to perform? [1 to 5]: "))
44      print('\n') # adding a new line for looks
45
46      # Step 3 - Show the current items in the table
47      if (strChoice.strip() == '1'):
48          for row in lstTable:
49              print(row) # prints loaded in data to user
50              continue

```

**Figure 3** — User menu and option 1, which prints out current data.

Since **lstTable** contains a nested dictionary inside of it, the printed-out rows will include the 'Task' and 'Priority' keys alongside each value that has been assigned to that key. An example can be seen in **Figure 4**, in which the tasks 'read maps' and 'learn excel' previously existed in the .txt file and were printed out following an input of '1' to check against **strChoice**:



```

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

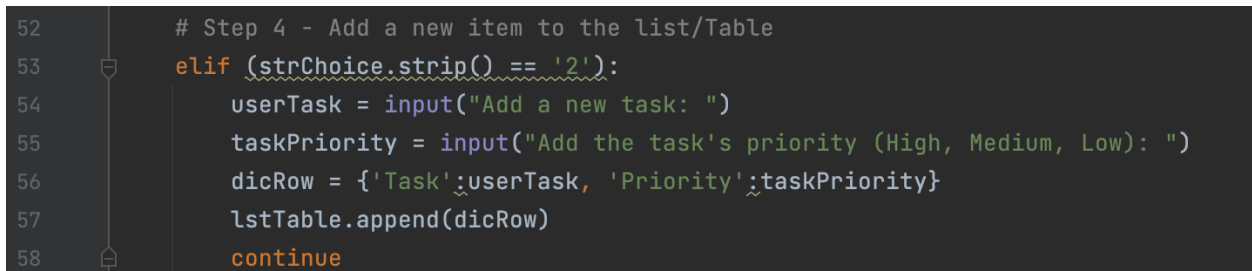
{'Task': 'read maps', 'Priority': 'low'}
{'Task': 'learn excel', 'Priority': 'high'}

```

**Figure 4 — Existing data printed out in the console (having been loaded in from the original .txt file)**

The next option provided to the user is to allow them to input their own data. For this `elif` statement to run, the user must input ‘2’ when prompted by the input statement in **strChoice**.

Two new variables were also created, **userTask** and **taskPriority**, which prompt the user to input the name of the task they want to complete and its priority (high, medium, low), respectively. Once these strings have been input and stored into their respective variables, they are then assigned to their respective dictionary key-value pairs. In the **dicRow** dictionary, **userTask** is paired as the value for the key ‘Task’ while **taskPriority** is paired to the key ‘Priority’. Once those variables have been assigned to their respective values in the dictionary, **dicRow** is then appended to **lstTable**. **Figure 5** shows this code block:



```

52      # Step 4 - Add a new item to the list/Table
53      elif (strChoice.strip() == '2'):
54          userTask = input("Add a new task: ")
55          taskPriority = input("Add the task's priority (High, Medium, Low): ")
56          dicRow = {'Task': userTask, 'Priority': taskPriority}
57          lstTable.append(dicRow)
58          continue

```

**Figure 5 — User input statements that prompt them to input a task name and its priority, which are then appended to the existing list.**

## 2.4 Removing & Saving Data:

Once the user has input ‘3’ from the menu options, the script goes into the second `elif` statement.

For each row, the script then checks that user’s task also occurs as an existing task name stored within the Key:Value pairs within **lstTable**. If the task within **lstTable** matches the user’s input, then the `.remove()` function deletes that row from the list. Although the `if` statement only uses key’s characters to confirm the task’s occurrence in **lstTable**, the entire row including the task’s priority is also deleted from the table. Each dictionary Key:Value pair forms one row of ‘data’ within the table.

```

60         # Step 5 - Remove a new item from the list/Table
61         elif (strChoice.strip() == '3'):
62             strInput = input("Input Task Name to remove it: ")
63             for row in lstTable:
64                 if row['Task'].lower() == strInput.lower():
65                     lstTable.remove(row)
66                     print("Task has been removed.")
67                     print(lstTable) # print updated table to show task has been removed
68                     continue

```

**Figure 6** — Removing data by looping through the existing rows `lstTable` to check if the user input-task occurs in the current list.

Once that row has been deleted, two print statements are presented to the user. The first print() statement on line 66 confirms to the user that the task has been removed and the second statement prints out the new `lstTable`, which shows the new, updated table to the user.

Option four allows the user to save the data they've input to the .txt file. The variable `writeFile` opens `objFile` and asks it to write any information to that file. A for-loop loops through each row in `lstTable` and writes each row that contains the 'Task' and 'Priority' keys and values to the .txt file.

Once the for-loop has written each row of data to the .txt file, the `.close()` function closes the .txt file and saves the new data to the .txt file. Until the user inputs '5' when the menu option is presented to them again, the while() loop will continue to print out the menu. Once they input '5', this will trigger the last elif statement, which breaks out of the while loop. This code can be seen in **Figure 7**:

```

69         # Step 6 - Save tasks to the ToDoList.txt file
70         elif (strChoice.strip() == '4'):
71             writeFile = open(objFile, 'w')
72             for row in lstTable: # loop through each row and write each task and priority to .txt file
73                 writeFile.write(row['Task'] + ', ' + row['Priority'] + '\n')
74                 continue
75             print("Tasks have been saved to file.")
76             writeFile.close() # closes the file
77             # Step 7 - Exit program
78             elif (strChoice.strip() == '5'):
79                 break # and Exit the program
80

```

**Figure 7** — Saving new data to the .txt file by looping through each row.

### 3. Running Script in PyCharm & Command Shell:

**Figure 8** shows the script being run in the Mac OS command shell while **Figure 9** shows it being run in PyCharm. The script loads the pre-existing data {read maps: low} and {learn excel:high}. In the Mac OS run, it adds 'write up chapter' tasks and removes the 'read maps' task while in the PyCharm run it removes the 'write up chapter' and adds the 'schedule meeting' task:

```

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': 'read maps', 'Priority': 'low'}
{'Task': 'learn excel', 'Priority': 'high'}

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

Add a new task: write up chapter
Add the task's priority (High, Medium, Low): high

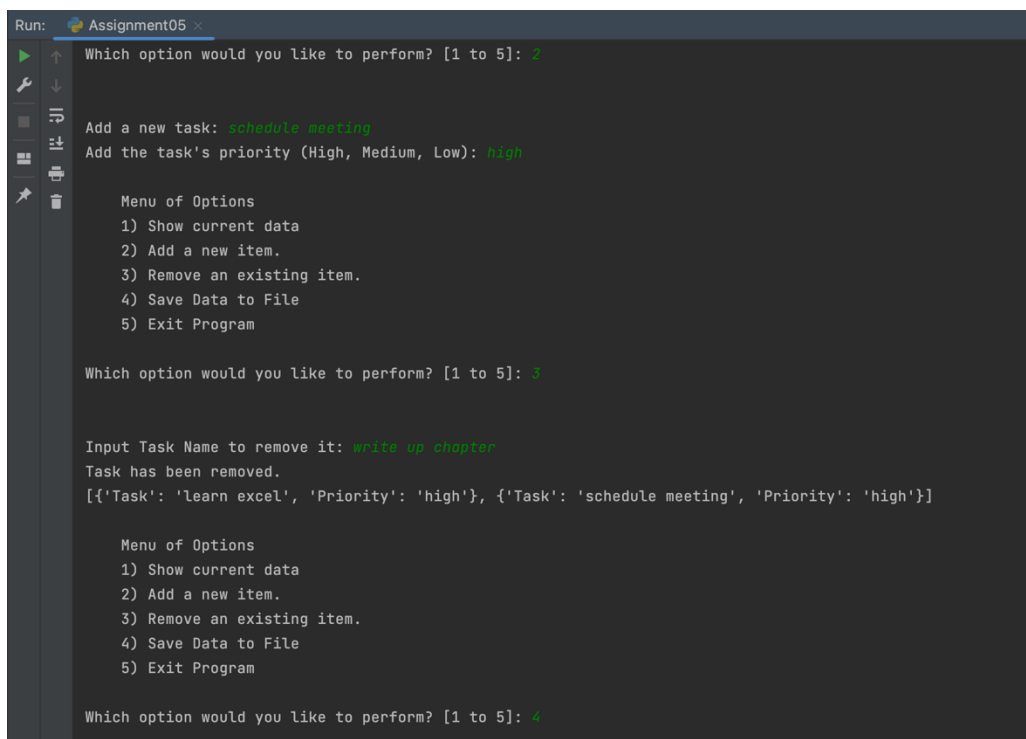
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program

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

Input Task Name to remove it: read maps
Task has been removed.
[{'Task': 'learn excel', 'Priority': 'high'}, {'Task': 'write up chapter', 'Priority': 'high'}]

```

**Figure 8 — The script being run in the Mac OS command shell.**



```

Run: Assignment05
Which option would you like to perform? [1 to 5]: 2

Add a new task: schedule meeting
Add the task's priority (High, Medium, Low): high

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

Input Task Name to remove it: write up chapter
Task has been removed.
[{'Task': 'learn excel', 'Priority': 'high'}, {'Task': 'schedule meeting', 'Priority': 'high'}]

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

```

**Figure 9 — Script running in PyCharm.**

**Figures 10 and 11** show the .txt file results from above, first for the command shell and second for PyCharm:



```
learn excel, high
write up chapter, high
|
```

*Figure 10 — Data saved from running the script in the command shell.*



```
learn excel, high
schedule meeting, high
```

*Figure 11 — Data saved from running the script in PyCharm.*

### 3.1 Summary:

This paper has looked at the To-Do list script which uses dictionaries and lists to store key:value data and present it to the user. It provides the user with a menu of options so that they can input the necessary tasks and the priority of each task.