**Name:**  Mohammed Qadri

**Date:** 2023-08-07

**Course:** IT FDN 110 A

**GitHub URL:** <https://github.com/mqadri22/IntroToProg-Python>

**Assignment 05 – Tuples, Lists, & Dictionaries**

**Introduction**

The objective of Assignment 05 is to gain familiarity with different collection types and their usage in Python, with particular attention paid to dictionaries. This would accomplished through the creation of a script that builds a Task/Priority To Do list and modifies it based on user command and input.

**Basic Code Structure**

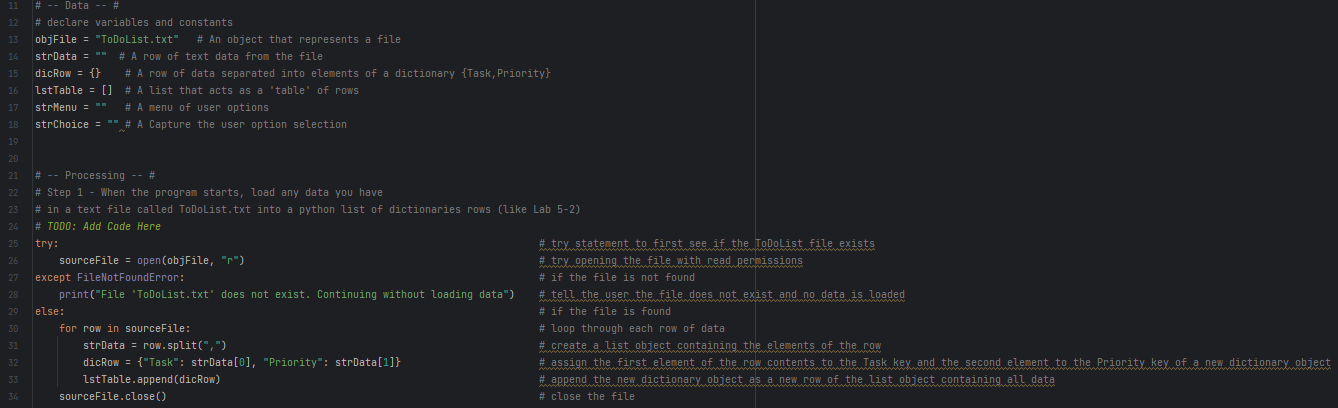
The basic structure of the script was already provided with the starter file that was provided with the assignment. It was advised that custom functions not be created for this assignment, so the script begins with the declaration of variables, which is followed by the reading in of existing data from a text file, and then ends with a while loop that runs repeatedly based on user inputs until the user decides to terminate.

Five options or methods are contained within the while loop: displaying currently recorded data, adding new user provided data to the current list, removing items from the list, saving the list to a file, and exiting the program. Development of each stage of the script is described in the following sections of this report.

**Processing: Loading Data from Files, Dictionaries**

The “Processing” section of this script is intended to load data into memory that is saved persistently in a text file. This loaded data would then be available for modification in subsequent sections of the script. Utilizing lessons learned from previous assignments, opening the file was fairly trivial via the open(…,“r”) function. New to this module are dictionaries and error handling, which are both used in this section. As required, the individual task and priority pairs are stored as dictionary objects that are then iteratively appended to the existing list object. When writing the script, the possibility of the “ToDoList.txt” file not existing was considered, which led the implementation of a try-except statement that handled situations where the necessary text file was present or not.

Once the existing data is read in from the file, the script continues onto the next section. Figure 1 below shows the section of the script intended to initializing variables and then loading data into a dictionary object from an existing file.



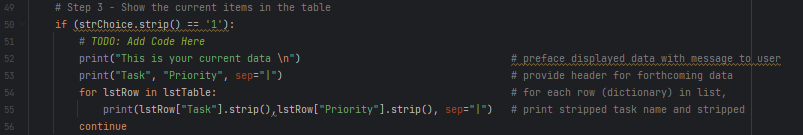
*Figure 1. Script to initialize variables and load in existing data*

**Input/Output: Working with Dictionaries**

The majority of this script falls under the Input/Output section. This section encompasses the presenting of a menu to the user and performing actions (displaying current data, adding new data, removing data, saving data, and exiting) based on user response to the menu options.

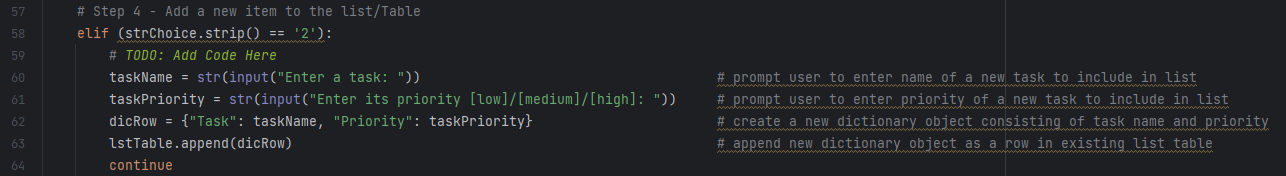
The displaying of the menu was already accomplished in the provided script template. User input in response to the menu was also already being stored as a string object. The subsequent functionalities largely dealt with dictionaries and their associated built-in functions.

The section for displaying current items in the table was accomplished by calling each element of each dictionary object (i.e., each row in the list table) via its key, followed by concatenating and then printing a string using these elements. See Figure 2 below for the section of the script that handled display of data.



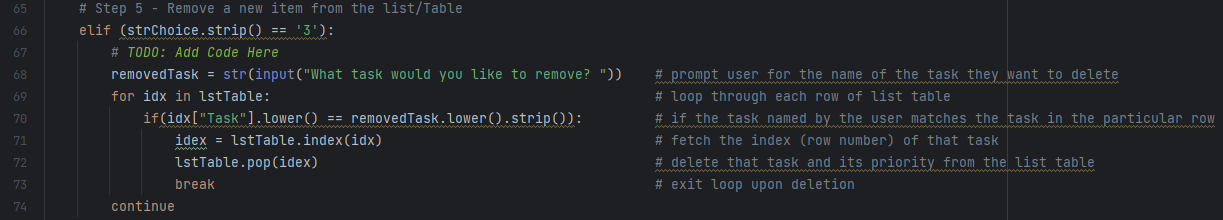
*Figure 2. Segment of script for displaying current data*

The next functionality or option in the menu was to add a new item to the existing list table. This required to the retrieval of input from the user in the form of a task name and task priority. This was accomplished fairly simply via the input() function. These user provided inputs were then fed into a dictionary object with keys similar to the existing table (i.e., “Task” and “Priority”). This dictionary object would then be added as a new row to the existing list table via the append function. This section closely resembled the process of loading existing data shown in Figure 1. Figure 3 shows the section of the script that handled the addition of new data to the list of tasks and priorities.



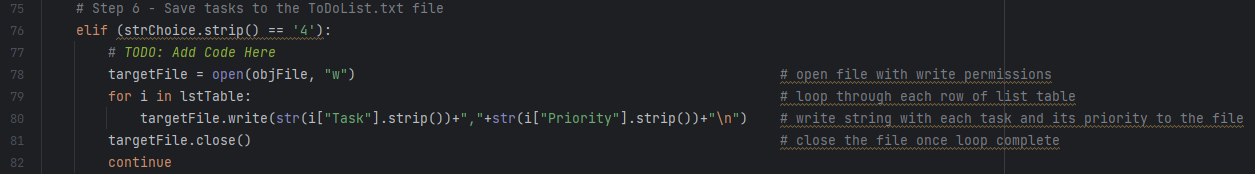
*Figure 3. Segment of script for adding new data to the existing list*

This segment was followed by one via which an item could be removed via user command. The assignment description was not explicit as to what should be removed (most recent entry, user-specified entry, etc.), so it was assumed that the removed entry would have to be specified by the user. To accomplish this, the user was prompted to input the name of a task they wished to remove. Then, the elements of each dictionary object in the list table were looped through, with the “Task” keyed item being compared to the user input. When a match was found, the index of that element was saved to a variable, which was then fed to the listObject.pop() function for removal. Figure 4 shows the section of the script dedicated to removing tasks from the list table.



*Figure 4. Segment of script for removing data from the list table*

The last nontrivial functionality of the script was intended to save the list table to a file upon user command. Using lessons learned from previous sections and previous assignments, this was accomplished fairly simply by first opening a file with write permissions, looping through each dictionary object within the list, extracting the elements for each key within each dictionary object, and writing a concatenated string to the file before closing it after processing the final dictionary object. Figure 5 shows the code for this segment of the script.

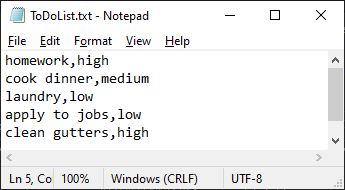


*Figure 5. Segment of script for saving list data to a file*

The final portion of the script exits the program. The “break” statement provided with the template accomplishes this already, so was left as such.

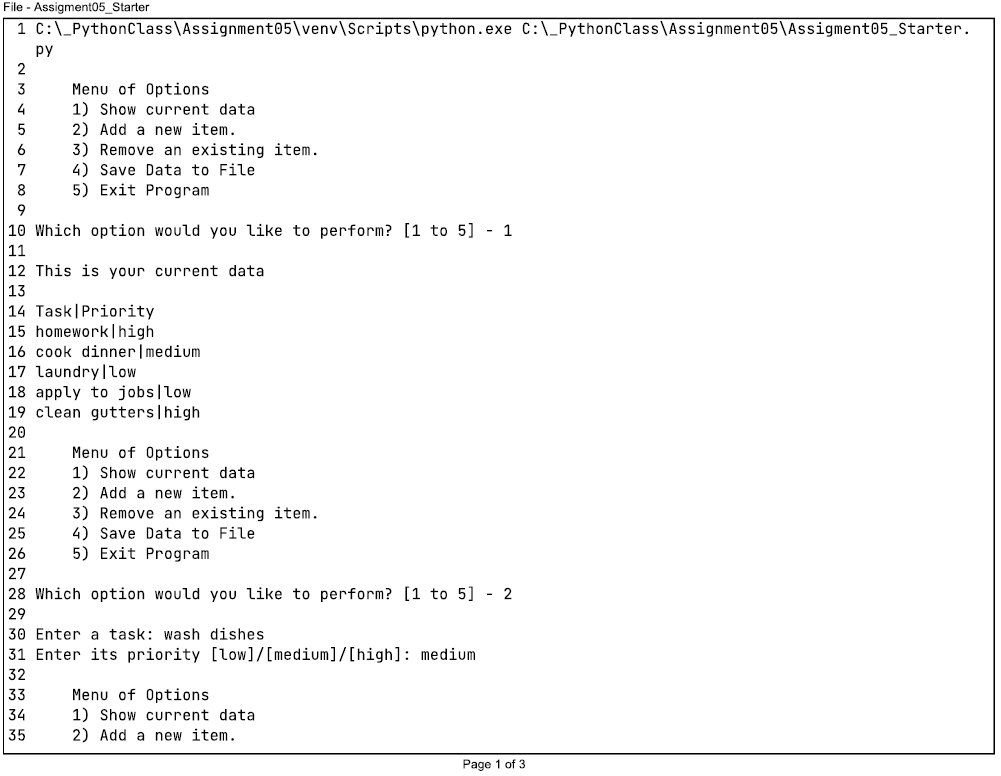
**Results**

A baseline ToDoList.txt file was created and populated with a list of tasks and their priorities, as shown in Figure 6.

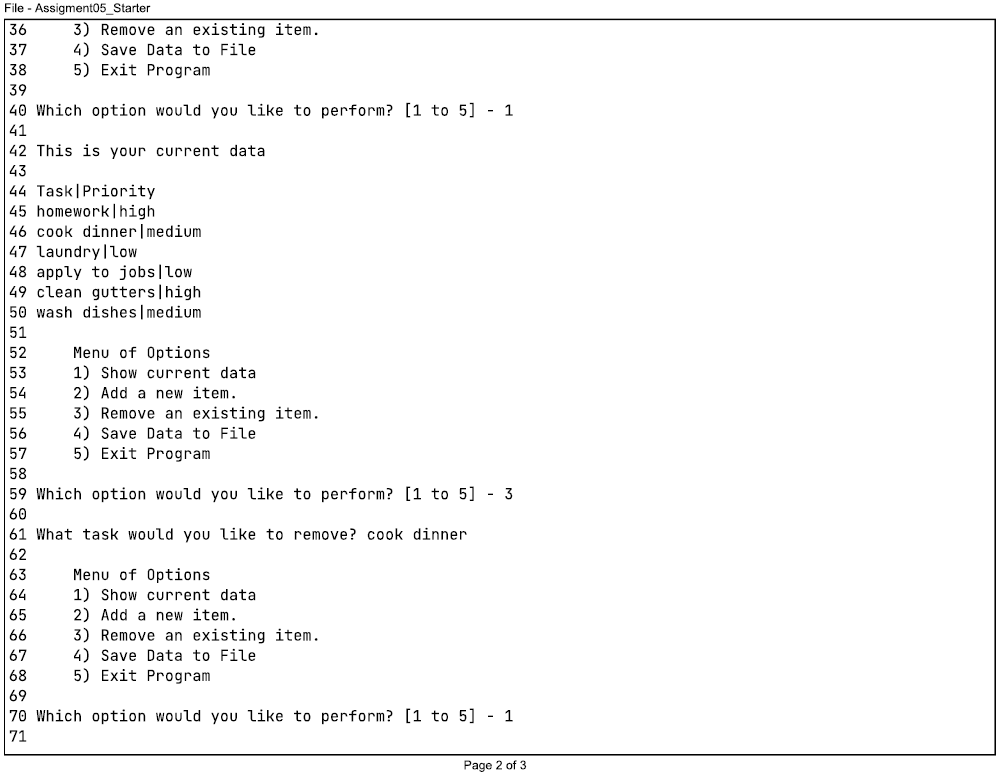


*Figure 6. Original baseline ToDoList.txt file*

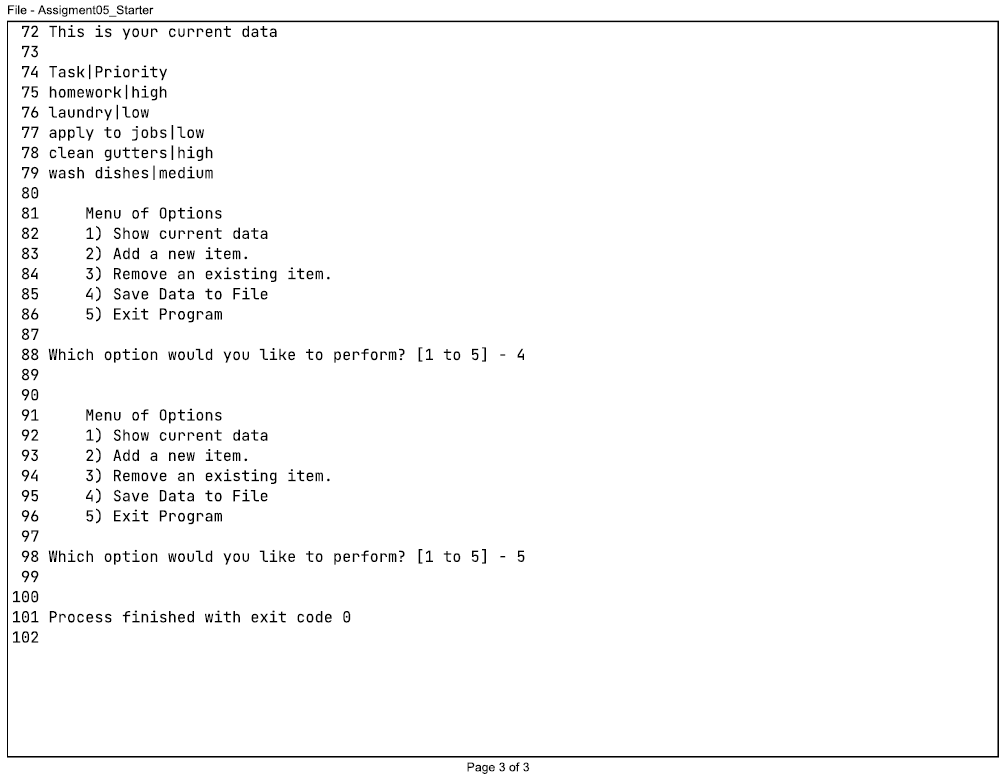
Figure 7.1 through 7.3 below show a print out of the PyCharm console as the script is run in PyCharm.



*Figure 7.1. Script running in PyCharm (1 of 3)*

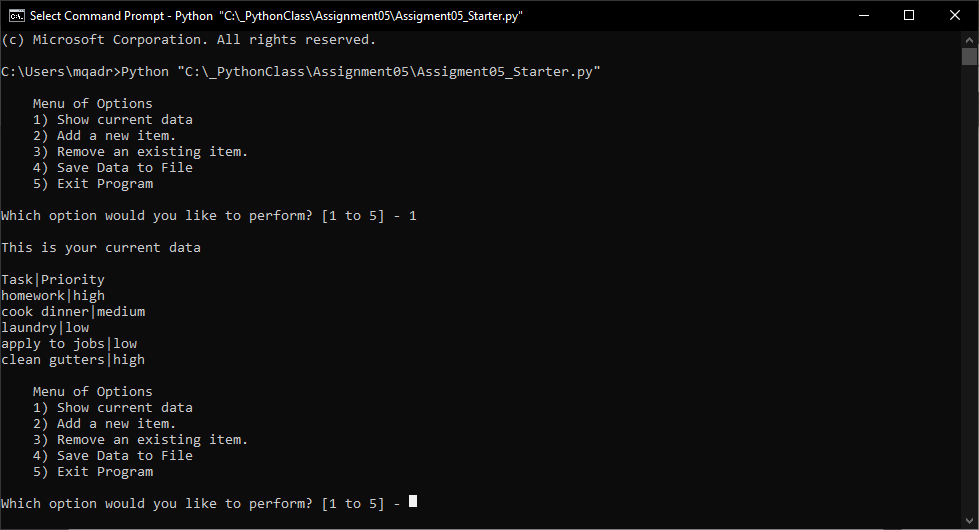


*Figure 7.2. Script running in PyCharm (2 of 3)*

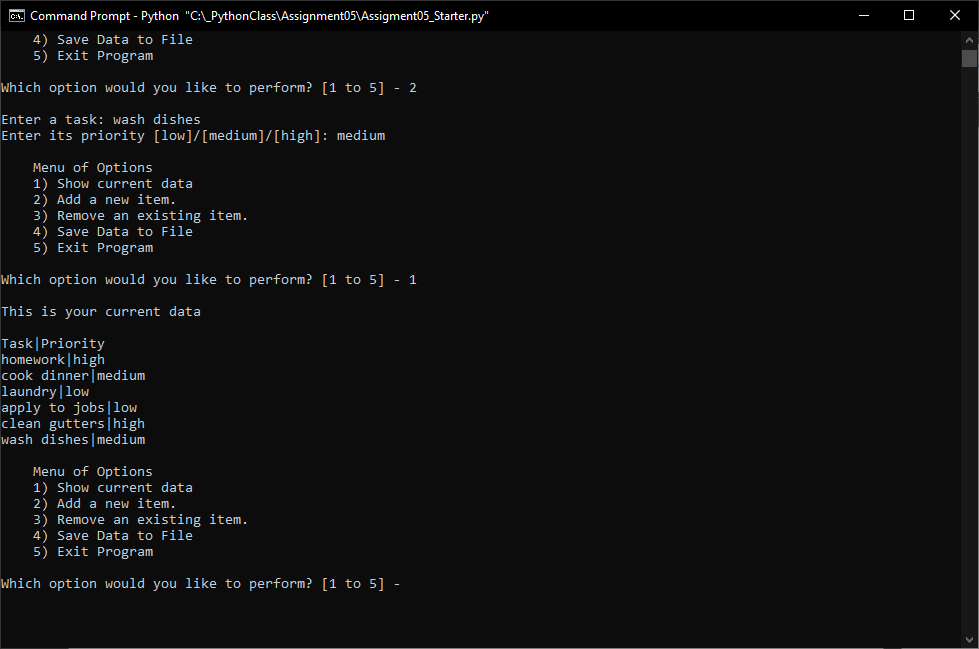


*Figure 7.3. Script running in PyCharm (3 of 3)*

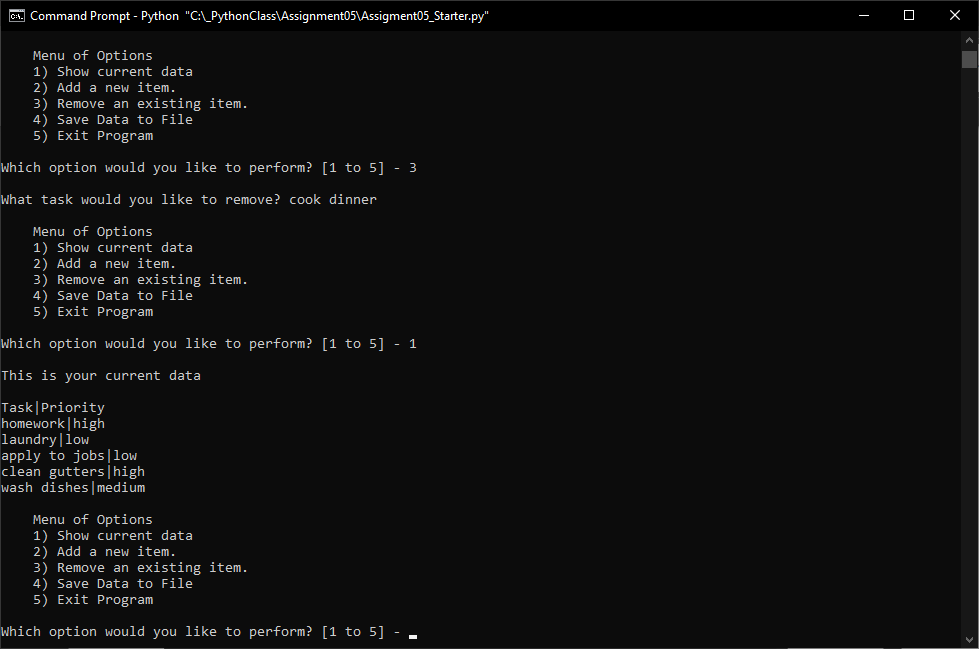
Figures 8.1 through 8.4 below show the script running in the command window, using the same baseline file and inputs as when run in PyCharm.



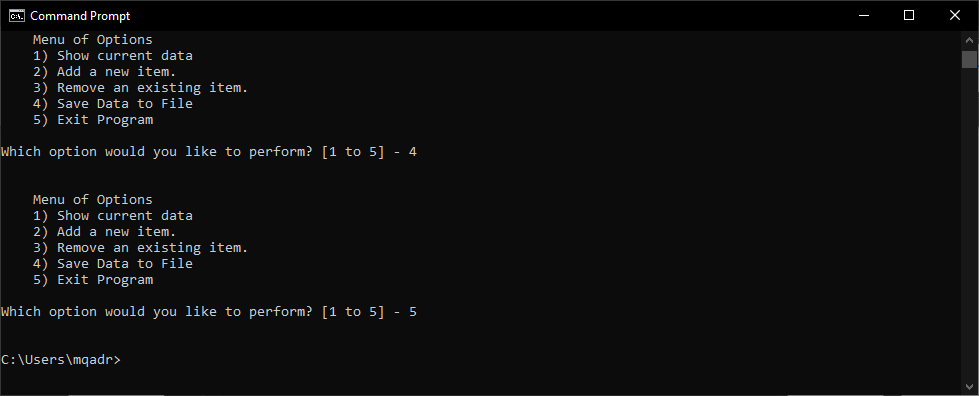
*Figure 8.1. Script running in command window (1 of 4)*



*Figure 8.2. Script running in command window (2 of 4)*

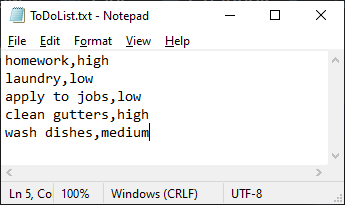


*Figure 8.3. Script running in command window (3 of 4)*



*Figure 8.4. Script running in command window (4 of 4)*

Figure 9 below shows the ToDoList.txt file that is output as a result of running the script as shown above using the baseline file shown in Figure 6. As can be seen in Figures 6 through 9, the output for each user-input command as well as the script overall are as expected.



*Figure 9. Output ToDoList.txt file*

**Conclusion**

The objectives of this assignment were achieved. The script was successfully written and generated the correct outputs while familiarity was gained with lists, dictionary objects, error handling, and GitHub. This script was slightly less challenging to write than the previous assignment (04), primarily due to the fact that familiarity had already been gained with the syntax and functionality of different collection types. Overall, the assignment was a good learning experience and provided further example of useful rudimentary functions that can be written in Python.