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.

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


# -- Processing -- #
# Step 1 - When the program starts, load any data you have
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
# TODO: Add Code Here
try:                                                        # try statement to first see if the ToDoList file exists
    sourceFile = open(objFile, "r")                         # try opening the file with read permissions
except FileNotFoundError:                                   # if the file is not found
    print("File 'ToDoList.txt' does not exist. Continuing without loading data")    # tell the user the file does not exist and no data is loaded
else:                                                       # if the file is found
    for row in sourceFile:                                  # loop through each row of data
        strData = row.split(",")                            # create a list object containing the elements of the row
        dicRow = {"Task": strData[0], "Priority": strData[1]}   # assign the first element of the row contents to the Task key and the second element to the Priority key of a new dictionary object
        lstTable.append(dicRow)                             # append the new dictionary object as a new row of the list object containing all data
    sourceFile.close()                                      # close the 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.

```python
    # Step 3 - Show the current items in the table
    if (strChoice.strip() == '1'):
        # TODO: Add Code Here
        print("This is your current data \n")                           # preface displayed data with message to user
        print("Task", "Priority", sep="|")                              # provide header for forthcoming data
        for lstRow in lstTable:                                         # for each row (dictionary) in list,
            print(lstRow["Task"].strip(),lstRow["Priority"].strip(), sep="|")   # print stripped task name and stripped
        continue
```

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.

```
    # Step 4 - Add a new item to the list/Table
    elif (strChoice.strip() == '2'):
        # TODO: Add Code Here
        taskName = str(input("Enter a task: "))                                  # prompt user to enter name of a new task to include in list
        taskPriority = str(input("Enter its priority [low]/[medium]/[high]: "))  # prompt user to enter priority of a new task to include in list
        dicRow = {"Task": taskName, "Priority": taskPriority}                    # create a new dictionary object consisting of task name and priority
        lstTable.append(dicRow)                                                  # append new dictionary object as a row in existing list table
        continue
```

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.

```
    # Step 5 - Remove a new item from the list/Table
    elif (strChoice.strip() == '3'):
        # TODO: Add Code Here
        removedTask = str(input("What task would you like to remove? "))    # prompt user for the name of the task they want to delete
        for idx in lstTable:                                                # loop through each row of list table
            if(idx["Task"].lower() == removedTask.lower().strip()):         # if the task named by the user matches the task in the particular row
                idex = lstTable.index(idx)                                  # fetch the index (row number) of that task
                lstTable.pop(idex)                                          # delete that task and its priority from the list table
                break                                                       # exit loop upon deletion
        continue
```

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.

```
    # Step 6 - Save tasks to the ToDoList.txt file
    elif (strChoice.strip() == '4'):
        # TODO: Add Code Here
        targetFile = open(objFile, "w")                                                 # open file with write permissions
        for i in lstTable:                                                              # loop through each row of list table
            targetFile.write(str(i["Task"].strip())+","+str(i["Priority"].strip())+"\n")  # write string with each task and its priority to the file
        targetFile.close()                                                              # close the file once loop complete
        continue
```

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.

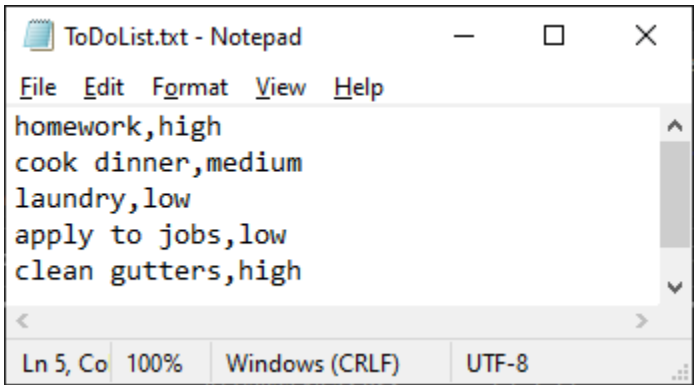

Figure 6. Original baseline ToDoList.txt file

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

File - Assigment05_Starter

```
C:\_PythonClass\Assignment05\venv\Scripts\python.exe 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

This is your current data

Task|Priority
homework|high
cook dinner|medium
laundry|low
apply to jobs|low
clean gutters|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

Enter a task: wash dishes
Enter its priority [low]/[medium]/[high]: medium

    Menu of Options
    1) Show current data
    2) Add a new item.
```

Page 1 of 3

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

File - Assigment05_Starter

```
    3) Remove an existing item.
    4) Save Data to File
    5) Exit Program

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

This is your current data

Task|Priority
homework|high
cook dinner|medium
laundry|low
apply to jobs|low
clean gutters|high
wash dishes|medium

    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

What task would you like to remove? cook dinner

    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

```

Page 2 of 3

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

File - Assigment05_Starter

```
 72 This is your current data
 73
 74 Task|Priority
 75 homework|high
 76 laundry|low
 77 apply to jobs|low
 78 clean gutters|high
 79 wash dishes|medium
 80
 81     Menu of Options
 82     1) Show current data
 83     2) Add a new item.
 84     3) Remove an existing item.
 85     4) Save Data to File
 86     5) Exit Program
 87
 88 Which option would you like to perform? [1 to 5] - 4
 89
 90
 91     Menu of Options
 92     1) Show current data
 93     2) Add a new item.
 94     3) Remove an existing item.
 95     4) Save Data to File
 96     5) Exit Program
 97
 98 Which option would you like to perform? [1 to 5] - 5
 99
100
101 Process finished with exit code 0
102
```

Page 3 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.

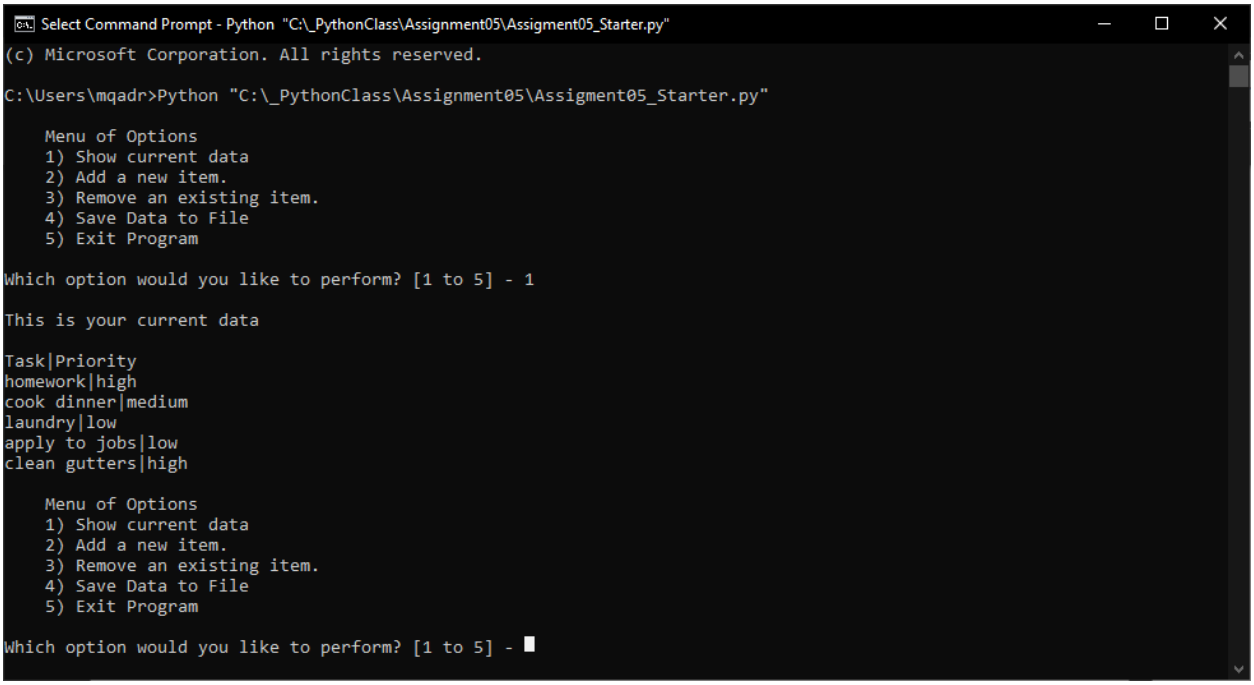

Figure 8.1. Script running in command window (1 of 4)

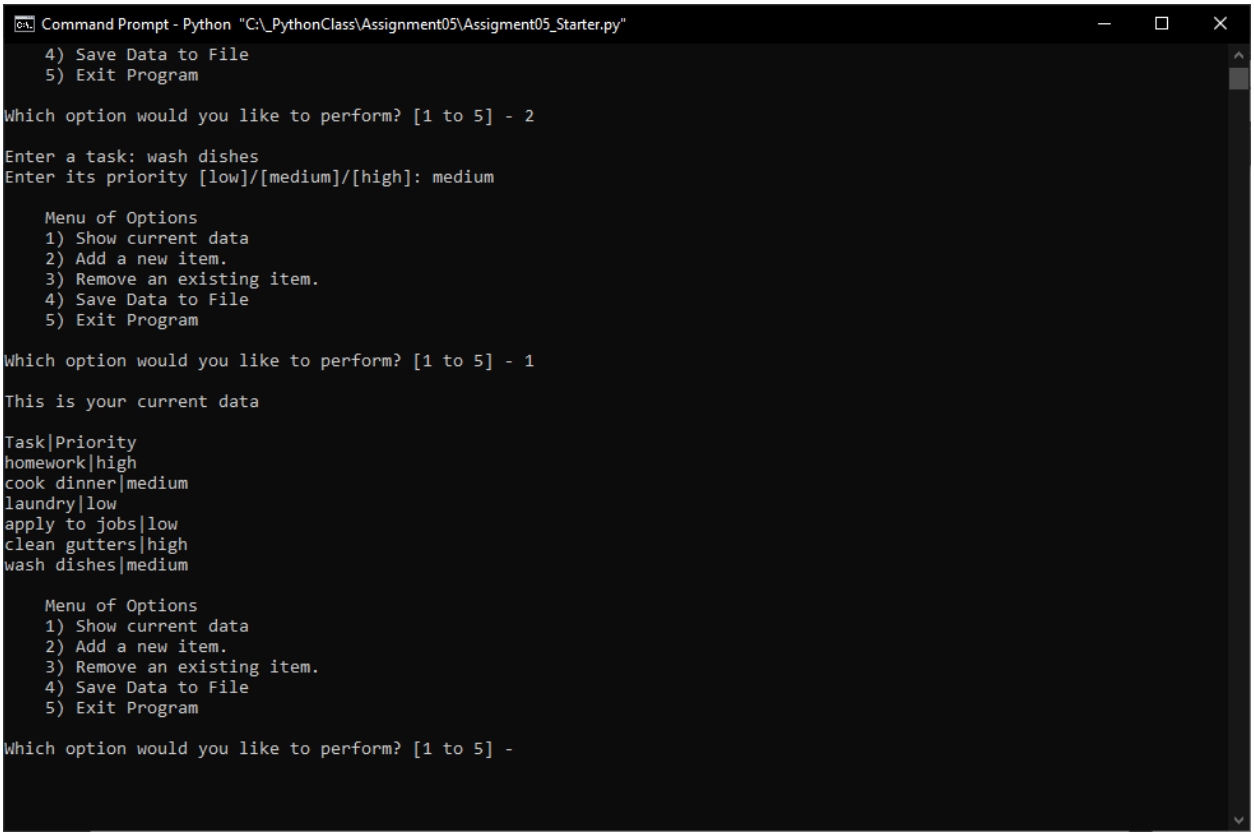

Figure 8.2. Script running in command window (2 of 4)

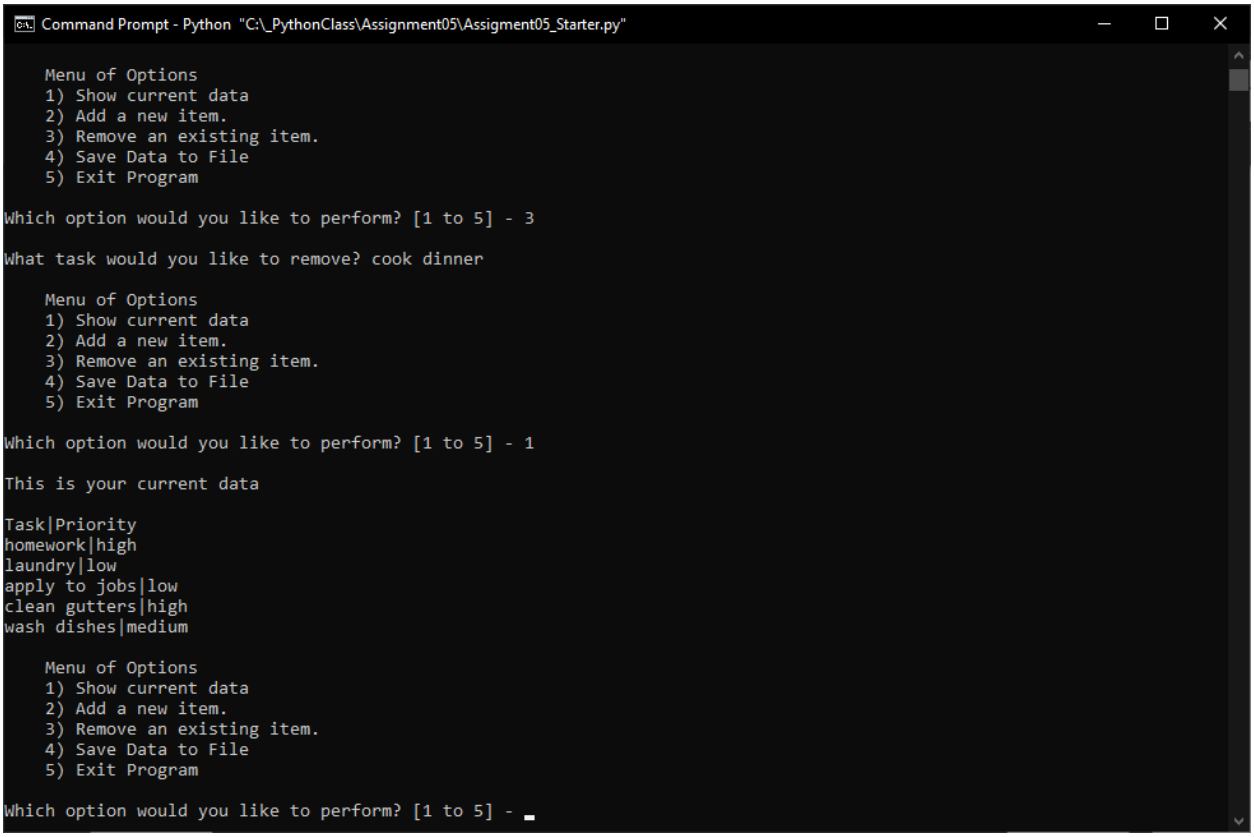

Figure 8.3. Script running in command window (3 of 4)

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.

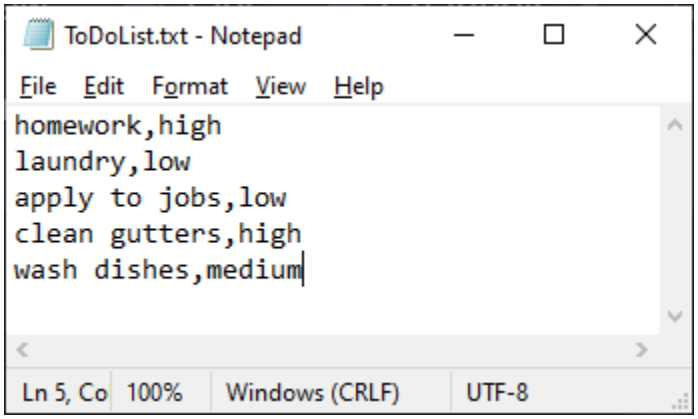

Figure 9. Output ToDoList.txt file

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.