

Rufus Ayeni  
November 1, 2020  
IT FDN 110 A  
Assignment 05

## To-Do List Script

### Introduction:

In Assignment05, I modified an existing script that manages a user's "To-Do" list. The script loads data from a file, adds data to a Python list, and returns the data to the file. The managed data is comprised of two columns-- **Task** and **Priority**. The assignment utilizes a newly introduced data structure-- *dictionaries*, and the idea of using dictionaries as elements in a list. This technique allows users to manage data in a table-like format. The script was developed by first determining the requirements, writing pseudocode, writing Python syntax, testing the script, and making adjustments.

### Requirements:

Modify an existing Python script that does the following:

1. Load data from a text file and into a list of dictionaries
2. The data is comprised of two columns-- **Task** (To-do Task) and **Priority**
3. Present the user with the following functionality:
  - a. Show current data
  - b. Add a new item (Task and Priority)
  - c. Remove an existing item
  - d. Save data to file
  - e. Exit program

## Writing Pseudocode (Modified):

After determining the requirements, the following pseudocode was written/modified:

1. Declare and initialize the following variables:
2. Load an existing data into a list of dictionaries:
  - a. Check if file exists
  - b. If file exists, open it and load data into dictionaries
  - c. Close file

To prevent the script from exiting due to an error, use the try and expect clauses.

3. Start while loop with keyword True.
  - a. Present menu of options to user
  - b. Based on chosen option, do either of the following:
    - i. Show current items in table
    - ii. Add new item to table
    - iii. Remove existing item from table
    - iv. Save data to file
    - v. Exit program

## Writing the Python Script:

The following steps were performed to convert the above pseudocode to Python code:

1. The script starts with a comment identifying the name of the script, its description, the original developer (*and contributors*), and the date it was created (*or modified*):

```

1  # -----
2  # Title: Assignment 05
3  # Description: Working with Dictionaries and Files
4  #               When the program starts, load each "row" of data
5  #               in "ToDoList.txt" into a python Dictionary.
6  #               Add the each dictionary "row" to a python list "table"
7  # ChangeLog (Who,When,What) :
8  # RRoot,1.1.2030,Created started script
9  # RAYeni,10.30.2020,Added code to complete assignment 5
10 # -----

```

Figure 1

- Some variables were added and others removed from the initial set of declared variables. The final set is in **Figure 3**. Later, additional local variables were declared in blocks of code following several `if` statements:

```

12  # -- Data -- #
13  # declare variables and constants
14  strFile = "ToDoList.txt" # Variable to reference file name.
15  lstRow = [] # A row of text data from the file
16  dicRow = {} # A row of data separated into elements of a dictionary
17  lstTable = [] # A list that acts as a 'table' of rows
18  strChoice = "" # A Capture the user option selection
19  taskNum = "" # A variable to reference row to remove
20  task = "" # A variable to reference task to do
21  priority = "" # A variable to reference task's priority
22  confirmRm = "" # User confirmation of removal of task

```

**Figure 3**

- The next block of code loads data from the file `strFile` (a variable referencing the file `ToDoList.txt`). If the file exists, the script loads any existing data. However, if the file doesn't exist, it will break the script. To prevent the application from exiting on an error (i.e., *file not found*), the `try` and `except` clauses are used:

```

25  # -- Processing -- #
26  # Step 1 - When the program starts, load any data you have
27  # in a text file called ToDoList.txt into a python list of dictionaries rows
28
29  # MODIFICATION - by RAYeni
30  try:
31      objFile = open(strFile, "r")
32      for row in objFile:
33          lstRow = row.split(",")
34          dicRow = {"Task": lstRow[0], "Priority": lstRow[1].strip()}
35          lstTable.append(dicRow)
36      objFile.close()
37  except:
38      print("")

```

**Figure 4**

4. The `while` loop starts with the keyword `True`, and presents a menu of five options to the user. Each iteration of the loop starts by presenting the menu to the user:

```

46 # -- Input/Output -- #
47 # Step 2 - Display a menu of choices to the user
48 while (True):
49     print("""
50         To-Do List
51
52         Menu of Options
53         1) Show current data
54         2) Add a new item.
55         3) Remove an existing item.
56         4) Save Data to File
57         5) Exit Program
58     """)
59     strChoice = str(input("Which option would you like to perform? [1 to 5] - "))
60     print() # adding a new line for looks

```

Figure 5

5. When the user selects **Option #1** (*Show current data*), the script determines if the table (`lstTable`) is empty or not. If the table is empty, the user is informed that *there are no data to display*. If data are present, they are presented in numbered rows. A **Task No** column was added to give the data a consistent look. Initially, it was only added to **Option #3** (*Remove...*), because the `list.pop()` method can only accept an `int` argument:

```

62 # MODIFICATION - by RAYeni
63 # Step 3 - Show the current items in the table
64 if (strChoice.strip() == '1'):
65     if len(lstTable) == 0: # Action to perform if table is empty
66         print("There are no data to display.")
67     else: # Action to do if table is not empty
68         j = 0 # variable used to number rows
69         print("Task No. | Task \t| Priority\n")
70         for row in lstTable:
71             print(f'{j} | {row["Task"]} \t| {row["Priority"]}')
72             j += 1
73         continue

```

Figure 6

6. The code for **Option #2** (*Add a new item...*), is straightforward. The `input()` function is used to accept data for the **Task** (To-Do Item) and the **Priority** (*High, Medium, or Low*) and store the data in the `task` and `priority` variables. The variables are later used to create a dictionary, which is assigned to the variable `dicRow`, which is subsequently added it to the list, `lstTable` using its `append()` method:

```

75     # MODIFICATION - by RAYeni
76     # Step 4 - Add a new item to the list/Table
77     elif (strChoice.strip() == '2'):
78         task = input("To-Do Item: ") # User enters task
79         priority = input("To-Do Item Priority (High, Medium, Low): ")
80         dicRow = {"Task": task, "Priority": priority} # Put entries in dict.
81         lstTable.append(dicRow) # Append dict to table/nested list
82         continue

```

Figure 7

7. When the user selects **Option #3** (*Remove item...*), rows of data are displayed to the user. Then the user selects the row number that he/she wants to delete. The selection is an `int` that is passed to the `pop()` method, which deletes the row:

```

84     # MODIFICATION - by RAYeni
85     # Step 5 - Remove a new item from the list/Table
86     elif (strChoice.strip() == '3'):
87         i = 0 # local variable to act as counter to number rows
88         print("Task No. | Task \t| Priority\n") # Data header
89         for row in lstTable: # loop through list of dictionaries
90             print(f'{i} | {row["Task"]} \t| {row["Priority"]}')
91             i += 1
92         print() # print new line
93         # Capture row number to delete
94         taskNum = int(input("Select task (0, 1, 2, ..) to remove: "))
95         # Confirm deletion of selected row
96         confirmRm = input(f'Remove Task No. {taskNum}? Enter y or n: ')
97         if confirmRm.lower() == "y": # Condition to remove row
98             lstTable.pop(taskNum) # Remove row
99             print(f'\nTask {taskNum} has been removed.')
100         elif confirmRm.lower() == "n": # Condition to not remove row
101             print(f'\nTask {taskNum} has NOT been removed.')
102         else:
103             print("Incorrect choice. No action taken. ")
104         continue

```

Figure 8

8. The code to save data to a file (**Option #4**) first opens the file `strFile` in write mode ("w"), loops through the list `lstTable`, and then uses the dictionary `get()` method to grab data from each row/dictionary and write them to the file:

```
102 # MODIFICATION - by RAYeni
103 # Step 6 - Save data to the ToDoList.txt file
104 elif (strChoice.strip() == '4'):
105     objFile = open(strFile, "w")
106     for row in lstTable:
107         objFile.write(row.get("Task") + "," + row.get("Priority") + "\n")
108     objFile.close()
109     print("Data saved to", strFile)
110     continue
```

Figure 9

9. When user chooses **Option #5** (*Exit Program...*), the script uses the keyword `break` to end the `while` loop:

```
116 # Step 7 - Exit program
117 elif (strChoice.strip() == '5'):
118     break # Exit while loop
```

Figure 10

10. After the script exits the `while` loop, the user is prompted to press the Enter key to exit the application:

```
120 # MODIFICATION - by RAYeni
121 # Exit script
122 input("\n\nPress the Enter key to exit.")
```

Figure 11

## Execution of Script:

1. To execute the script in PyCharm, you simply right-click the code area and select “**Run** ‘Assignment05.’”

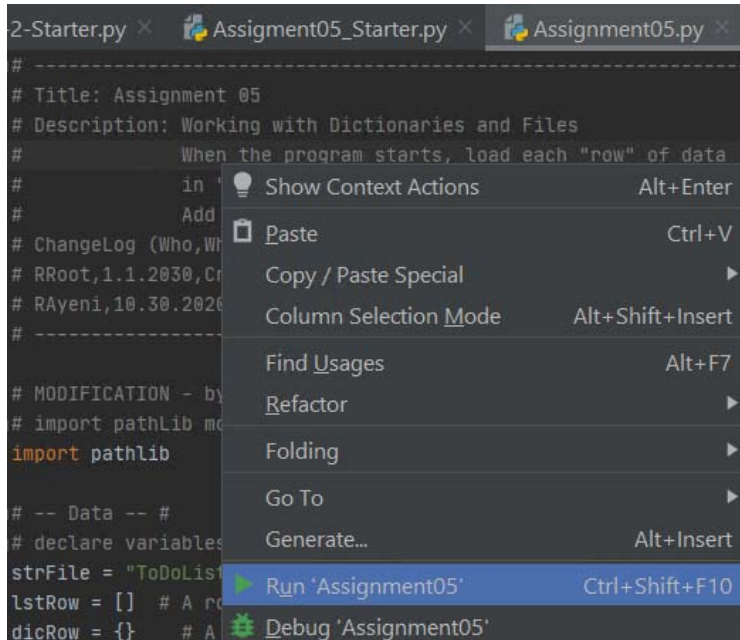


Figure 12

2. Running the script kicks off a command window through which a user can interface with the application via a menu of options:

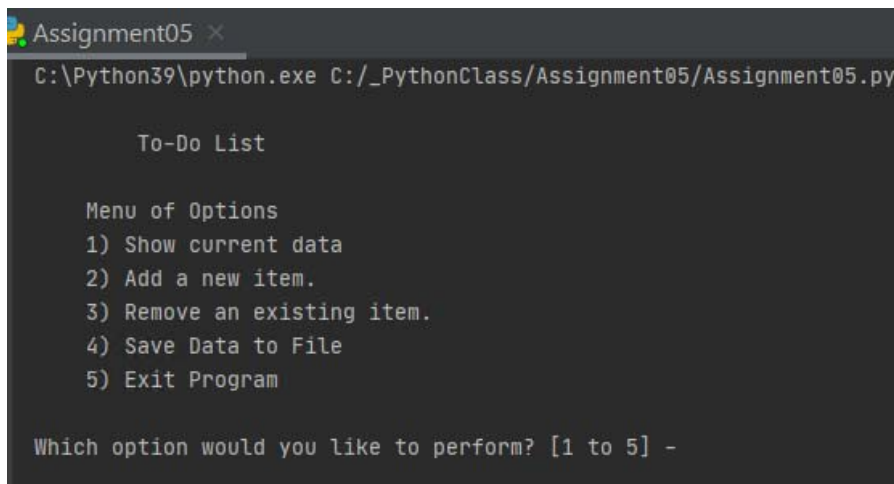
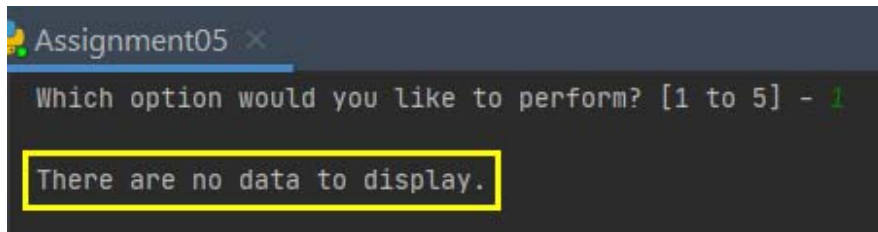


Figure 13

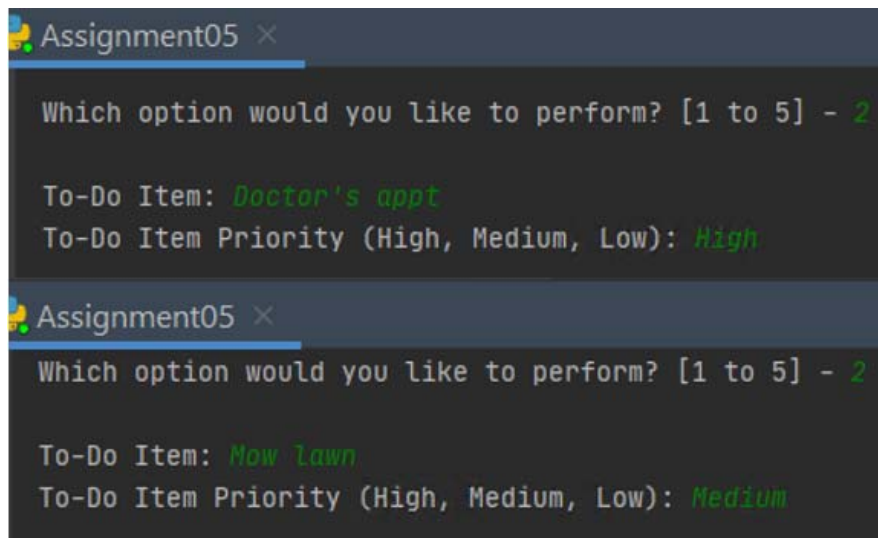
3. In this example, **Option #1** (*Show current data*) returns “There are no data to display,” because the first run of the script has yet to populate the file with data:



```
Assignment05 x
Which option would you like to perform? [1 to 5] - 1
There are no data to display.
```

Figure 14

4. **Option #2** (*Add a new item*), asks the user for a **To-Do Item** and its **Priority**:

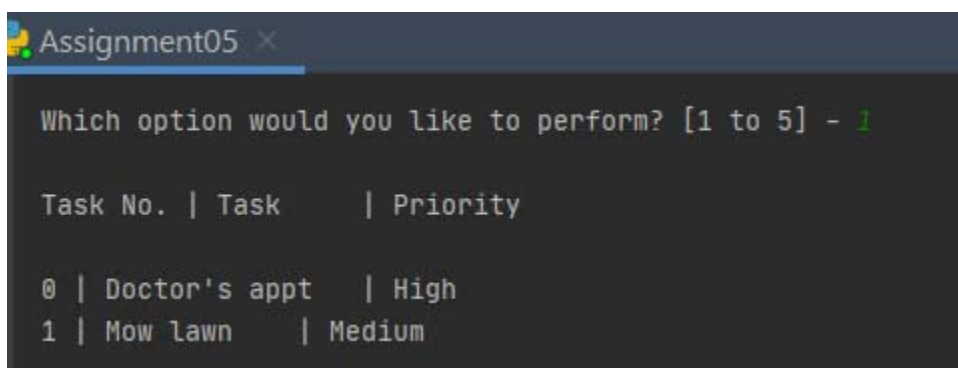


```
Assignment05 x
Which option would you like to perform? [1 to 5] - 2
To-Do Item: Doctor's appt
To-Do Item Priority (High, Medium, Low): High

Assignment05 x
Which option would you like to perform? [1 to 5] - 2
To-Do Item: Mow lawn
To-Do Item Priority (High, Medium, Low): Medium
```

Figure 15

5. Returning to **Option #1** (*Show current data*) confirms that the two entries reside in the table `lstTable`:



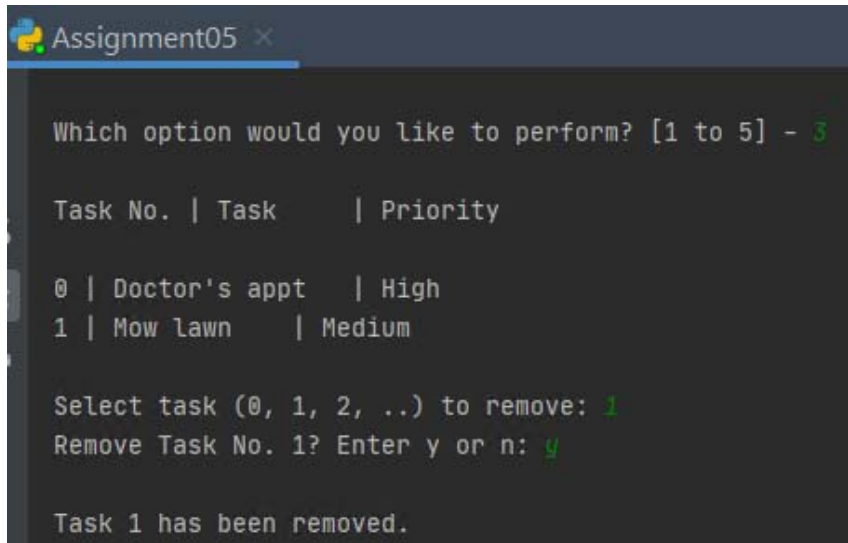
```
Assignment05 x
Which option would you like to perform? [1 to 5] - 1

Task No. | Task      | Priority
0 | Doctor's appt | High
1 | Mow lawn     | Medium
```

Figure 16



6. **Option #3** (*Remove an existing item*) displays the rows of data, and prompts the user to select a row to delete:



```
Assignment05 x
Which option would you like to perform? [1 to 5] - 3

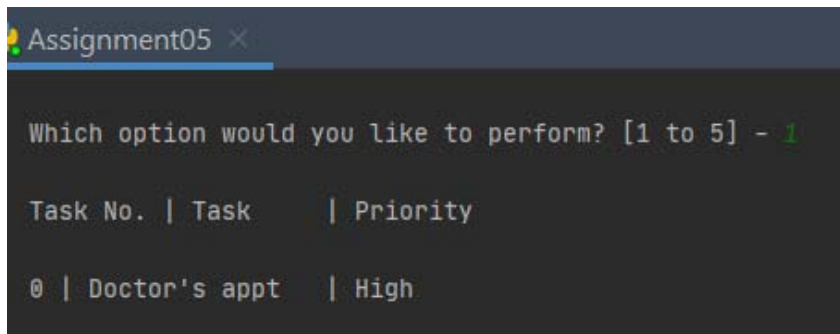
Task No. | Task      | Priority
0 | Doctor's appt | High
1 | Mow lawn     | Medium

Select task (0, 1, 2, ..) to remove: 1
Remove Task No. 1? Enter y or n: y

Task 1 has been removed.
```

Figure 17

7. Returning to **Option #1** (*Show current data*), confirms the row/task (i.e., *1/Mow lawn*) has been deleted:



```
Assignment05 x
Which option would you like to perform? [1 to 5] - 1

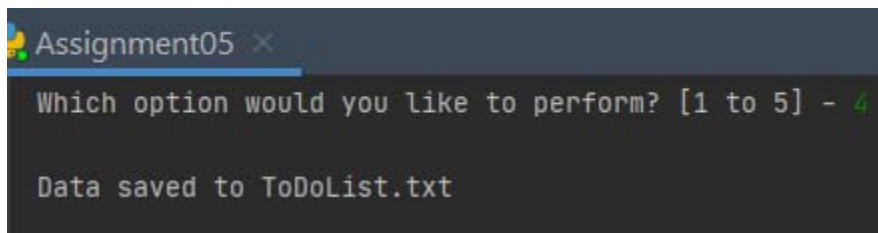
Task No. | Task      | Priority
0 | Doctor's appt | High
1 | Mow lawn     | Medium

Select task (0, 1, 2, ..) to remove: 1
Remove Task No. 1? Enter y or n: y

Task 1 has been removed.
```

Figure 18

8. **Option #4** (*Save Data to File*) gives the user a message confirming the data was saved to the `ToDoList.txt` file:

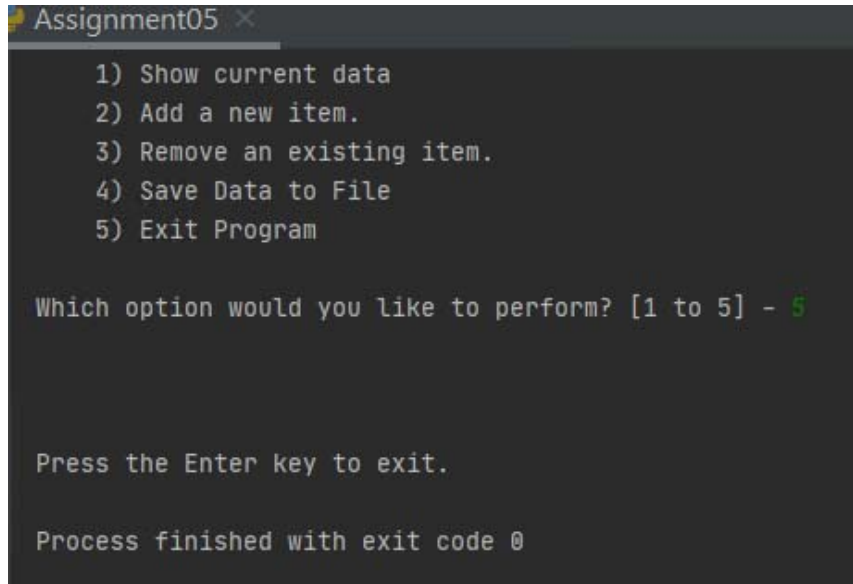


```
Assignment05 x
Which option would you like to perform? [1 to 5] - 4

Data saved to ToDoList.txt
```

Figure 19

9. **Option #5 (Exit Program)** asks the user to press the [Enter](#) key to exit the application:



```
Assignment05 X
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Exit Program

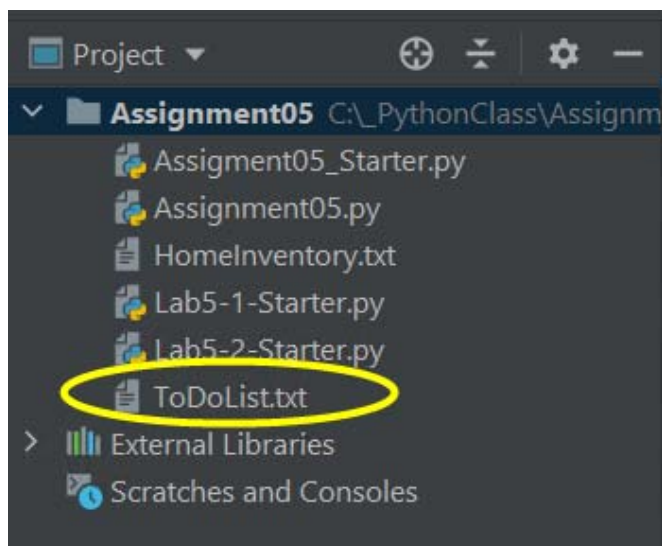
Which option would you like to perform? [1 to 5] - 5

Press the Enter key to exit.

Process finished with exit code 0
```

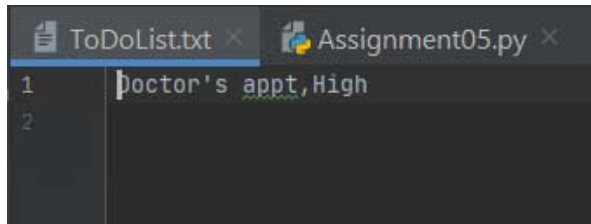
**Figure 20**

10. The Project pane shows that the file [ToDoList.txt](#) was created:



**Figure 21**

11. When the file is opened, in PyCharm, we can see the user's entries were written to the file:



**Figure 22**

12. **Figure 23** shows the execution of the script in Windows a Command Prompt:

```

C:\_PythonClass\Assignment05>python Assignment05.py

To-Do 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] - 2

To-Do Item: Mow lawn
To-Do Item Priority (High, Medium, Low): Low

To-Do 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] - 4

Data saved to ToDoList.txt

To-Do 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] - 5

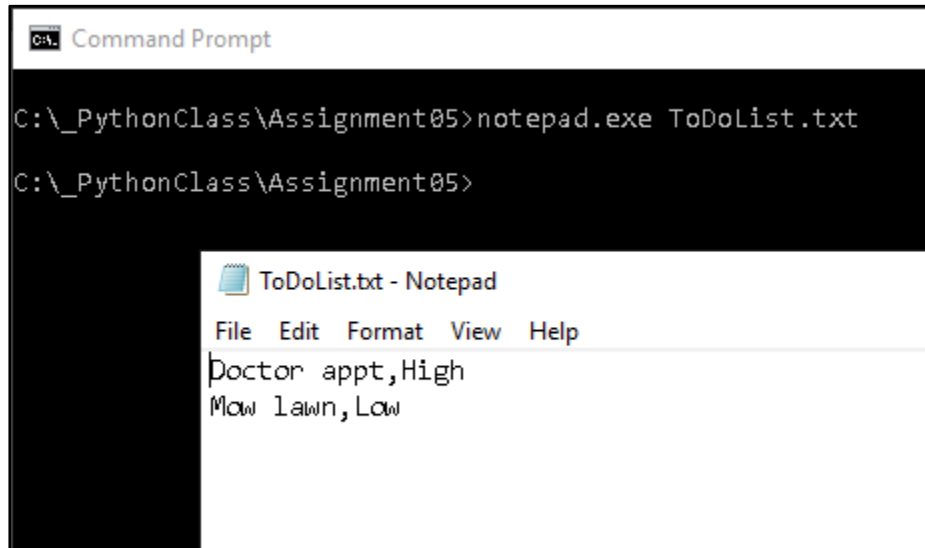
Press the Enter key to exit.

C:\_PythonClass\Assignment05>

```

**Figure 23**

13. **Figure 24** confirms that the data was written to the same file that was created when the application was run previously in PyCharm:



**Figure 24**

## Summary:

In Assignment05, I modified an existing script that manages a “To-Do” list for the user. The script made use of dictionaries to organize data in rows, and made use of lists to store the dictionaries as elements to present the data in a table-like format. The `try` and `except` clauses were used for the first time to prevent the script from exiting on an error (i.e., an attempt to access a file that doesn’t exist).