

Writing Files to Dictionaries

Introduction

So far, we have been working on writing strings and lists to file. Since the `write()` method used to write an object to file takes a string as its argument, we could directly insert strings to write them to file, and for lists, we could use indexes to access strings within lists to write them in. In the case of dictionaries, we would have to extract strings from the dictionaries using *keys* rather than *indexes*, as dictionaries use character subscripts rather than numerical positional subscripts to identify their values.

In this Assignment 05, we build on our previous assignments and labs that ask the user to add or edit data in a file. However, this time, we first read the data from a file and then add or edit data before saving the data to the same file. We are provided a starter script that already has the menu of options for the user listed out and provides the framework that we will be writing out code in. What is left is for us to load the data, and then fill out what will happen when the user inputs each menu option from 1 -5.

Reading Data from a File

When we start the project, the file that we will save to, called 'ToDoList.txt', is empty. To avoid an error when we run the command to open said file, we start off by using try and except. If the file exists, the program will run, otherwise, the exception part of the script will catch the error before it executes, and it will print out that the file is not found, but you can create a new file (Figure 1).

```
# Read/load data from file
try:    # if file exists
    objToDo = open(objFile, "r")
    for row in objToDo:
        task, priority = row.split(",")
        dicRow = {"Task": task, "Priority": priority.strip()}
        lstTable.append(dicRow)
    objToDo.close()
except:    # if file does not exist
    print("File not found. You can create a new file when you save.")
```

Figure 1: Reading the data from a file that may or may not exist

Choosing from a Menu of Options

The starter script for this assignment already provides us with 5 menu options:

```
while (True):
    print("""
    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
    """)
    strChoice = str(input("Which option would you like to perform? [1 to 5] -
    "))
    print() # adding a new line for looks
```

Figure 2: Menu of Options—Part of Starter Code

The variable `strChoice` captures the users input on which item is chosen. We use the “if” conditional statement for each choice from 1- 5. I added an else statement below the 5 if statements representing the menu choices; this else statement will capture incorrect entries and remind the user to choose a valid menu option.

`strChoice == 1`

The ‘if `strChoice.strip() == '1'`’ statement is removing any accidental spaces the user may have inputted and then comparing that to the ‘1’ string. If this is true, then it runs a *for* loop through the list variable `lstTable`, and for each row in the table, it prints out the first and second value from the dictionary row. (I use the *title()* method with `row["Task"]` to ensure a consistent capitalization of the first letter of all listed tasks.) I use `row["Task"]` and `row["Priority"]` as the method of value retrieval because “Task” and “Priority” are the keys for our dictionary row.

If `lstTable` is empty, as it would be the first time we run the program and no ‘ToDoList.txt’ file exists, then only the column headings will be displayed.

Once this if statement finishes executing, the program goes back to the beginning of the *while True* loop. (The “continue” is not necessary here, but clarifies the code.)

```
if (strChoice.strip() == '1'):
    # TODO: Add Code Here - Show current Data
    print("Task" + " | " + "Priority")
    for row in lstTable:
        print(row["Task"].title() + "|" + row["Priority"])
    continue
```

Figure 3: Choosing the first menu option

strChoice == 2

If `strChoice.strip() == 2`, the user is choosing to add a task and its priority. I therefore add code to allow the user to input the task and then another input function to allow the user to input the priority of the task. Both of these entries are captured in variables `strTask` and `strPriority`. I then add them into the dictionary `dicRow` under the keys "Task" and "Priority". Then, the dictionary entry is appended to the `lstTable` variable. I finish off this choice by displaying the updated table.

```
# Step 4 - Add a new item to the list/Table
elif (strChoice.strip() == '2'):
    # TODO: Add Code Here - Add New Item
    strTask = input("Enter Task to be Added: ")
    strPriority = input("Enter Priority Number of Added Task: ")
    dicRow = {"Task": strTask, "Priority": strPriority}
    lstTable.append(dicRow)
    print("Your item was added to the list.")
    print("Task" + " | " + "Priority")
    for row in lstTable:
        print(row["Task"].title() + "|" + row["Priority"])
    continue
```

Figure 4: Choosing the second menu option

strChoice == 3

In the third choice, the user is choosing to remove an existing row from the table. Here, I ask the user which task they want to remove. The associated priority will also be removed along with the task. The user enters the tasks. Using the *for* loop, I iterate over each row of the table and check to see if the task is the same as `row["Task"]`. If it is, I simply remove that row, using `lstTable.remove(row)`.

I cannot simply use the *else* statement to capture when the task user inputs does not exist in the rows of the table. If I do, it would print out my exception statement that the task is not found for all the rows that the inputted task to be removed (*strRemove* in the code) is not in a row. I only want to print this statement if *strRemove* is not found in `lstTable`. To do this, I initialize an integer *intAbsent* and sent it to 1. When I am checking if `strRemove == row["Task"]`, I set `intAbsent = intAbsent*0` for ever time the if condition is executed. This ensures that a single instance of `strRemove == row["Task"]` will result in *intAbsent* permanently changed to 0. The *else* statement then states that for the rows that the if condition does not execute, `intAbsent = intAbsent*1`. The *else* statement was not strictly necessary, but completes the logic more explicitly. Then, once we complete the *for* loop, the main script continues on to the if statement that if `intAbsent == 1`, which means that the if statement was never true and *strRemove* was not found in any of the rows of `lstTable`, then the inputted task was not found and the user should check the existing items and choose a valid task.

Whether *strRemove* was inputted correctly or not, this block of script ends with a printout of the updated items – if an item was successfully removed, this will show the user the new table, and if not, it will show the user the eligible items that the user can choose from to remove on a retry.

Figure 5 below illustrates the script for if the user enters this menu option.

```

elif (strChoice.strip() == '3'):
    # TODO: Add Code Here - Remove an Existing Item
    strRemove = input("Which task do you want to remove? ")
    strRemove = strRemove.strip()
    intAbsent = 1 # initializing variable to capture if
strRemove is not present 0/1
    for row in lstTable:
        if row["Task"].lower() == strRemove.lower():
            intAbsent = intAbsent*0
            lstTable.remove(row)
            print(strRemove.title() + " was removed. Updated list below.")
        else:
            intAbsent = intAbsent*1
    if intAbsent == 1:
        print("Task not found. See below for updated list of tasks.")
    # Show updated list or remind user of existing items
    print("Task" + " | " + "Priority")
    for row in lstTable:
        print(row["Task"].title() + "|" + row["Priority"])

    continue

```

Figure 5: Choosing the third menu option

strChoice == 4

If the user is done with all changes, s/he will want to save this data. Here, I simply use the `open()` function to open the file. I iterate over each row of `lstTable` and extracting the values using the “Task” and “Priority” keys, I write the data using the `ObjToDo.write()`. After iterating over each row and extracting the strings to be saved (separated by a comma and ending each row with a new line), I close the file and print out “Tasks Saved!”

```

# Step 6 - Save tasks to the ToDoToDoList.txt file
elif (strChoice.strip() == '4'):
    # TODO: Add Code Here - Save tasks to ToDoList
    objToDo = open(objFile, "w")
    for row in lstTable:
        objToDo.write(row["Task"] + ',' + row["Priority"] + '\n')
    objToDo.close()
    print("Tasks Saved!")
    continue

```

Figure 6: Saving the Data

strChoice == 5

This last option simply allows the user to exit the program. If chosen, it prints out a simple message that the user is exiting the program and it breaks out of the `while True` block.

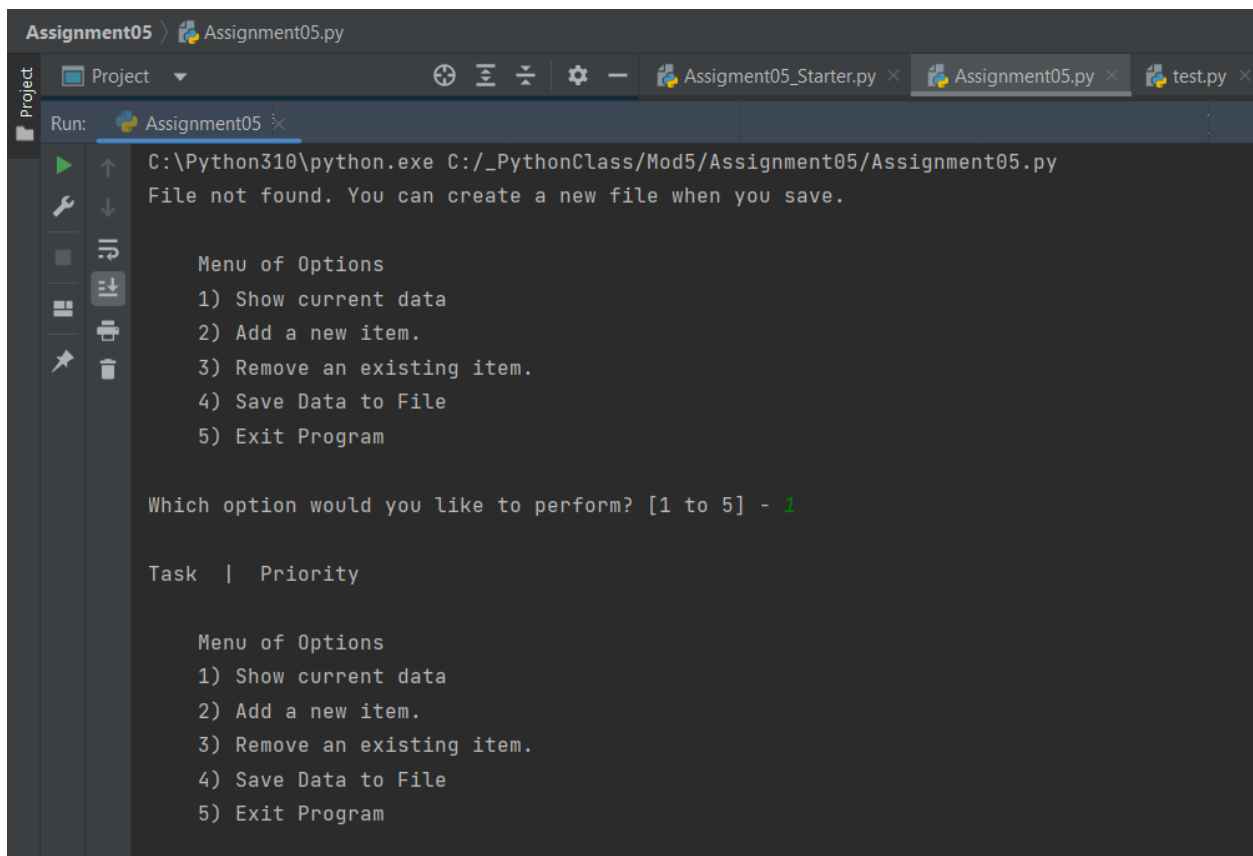
We can also see in *Figure 7* that after the last *elif* statement, there is an *else* statement to capture if the user makes an ineligible entry. Then, the user will be reminded to choose from the menu options 1-5.

```
# Step 7 - Exit program
elif (strChoice.strip() == '5'):
    # TODO: Add Code Here --Exit Program
    print("You are exiting the program.")
    break # and Exit the program
# Catch for errors in strChoice input
else:
    print("Input not recognized. Please choose numbers 1, 2, 3, 4, or 5.")
```

Figure 7: The fifth choice and the *else* statement

Trial Run of the Program

Figures 8 below illustrates a trial run of the program, starting from when no 'ToDoList.txt' file exists. It prints out the message that the file is not found but the user can create it.



```
Assignment05 > Assignment05.py
Run: Assignment05
C:\Python310\python.exe C:/_PythonClass/Mod5/Assignment05/Assignment05.py
File not found. You can create a new file when you save.

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

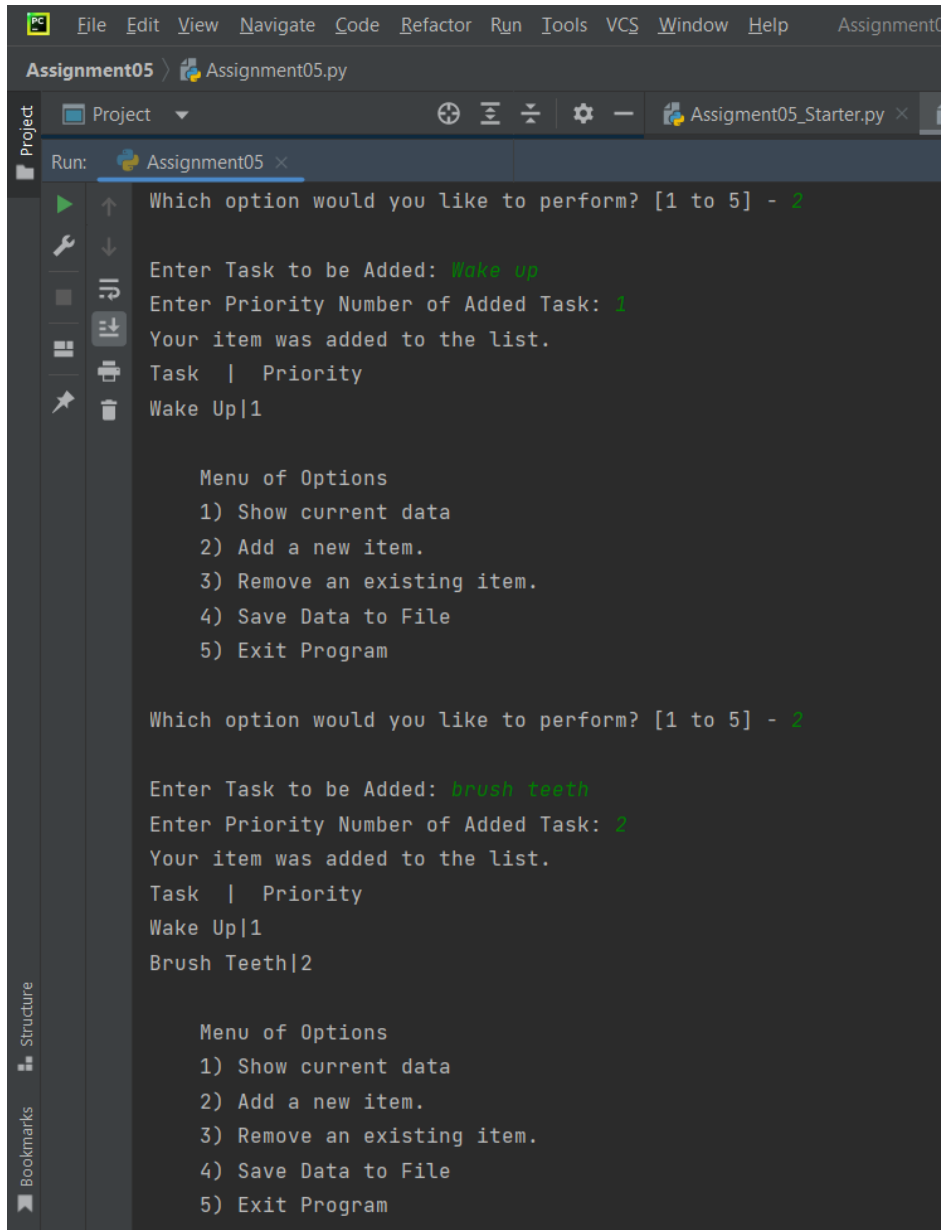
Task | Priority

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
```

Figure 8: 'File Not Found' and an Empty Display data with empty *lstTable*

Selecting Option 1 at this point also displays an empty table with just the headings as *lstTable* is empty.

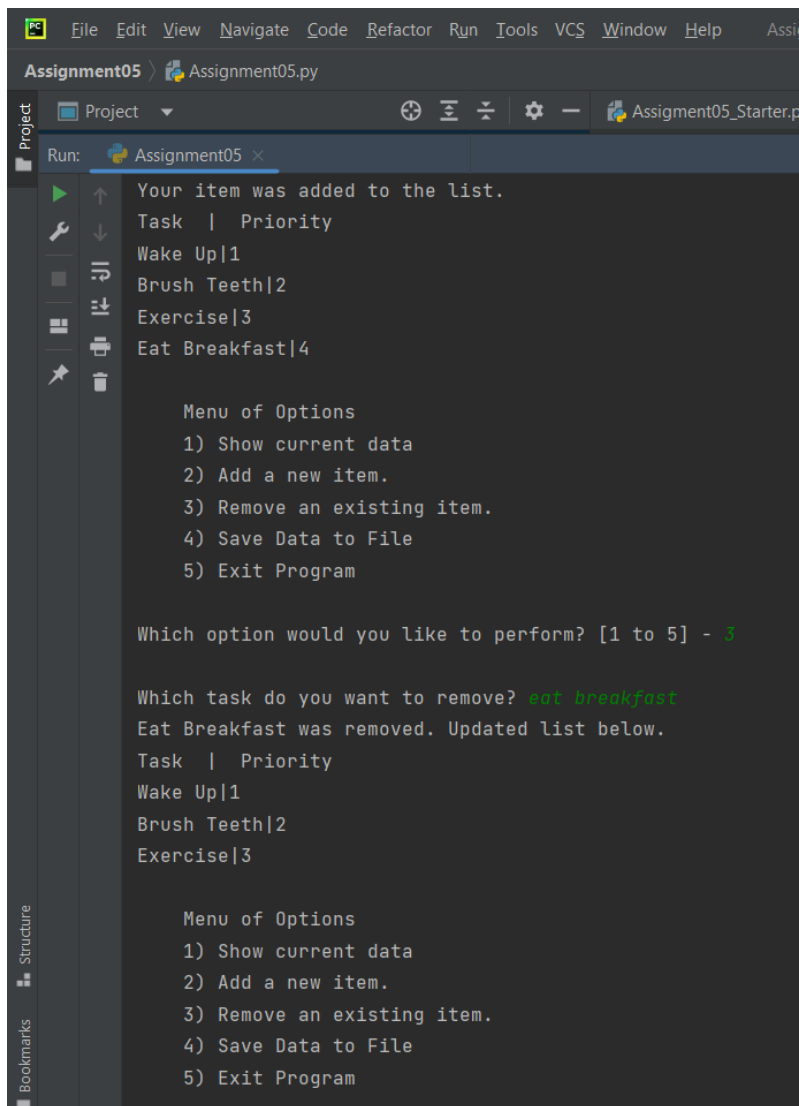
Figure 9 shows samples of adding values for the keys tasks and priority to a dictionary and appending it to the *lstTable*. The *lstTable* is updated and displayed with each addition.



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help Assignment05
Assignment05 > Assignment05.py
Project
Run: Assignment05 x
Assignment05_Starter.py x
Which option would you like to perform? [1 to 5] - 2
Enter Task to be Added: Wake up
Enter Priority Number of Added Task: 1
Your item was added to the list.
Task | Priority
Wake Up|1
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 Task to be Added: brush teeth
Enter Priority Number of Added Task: 2
Your item was added to the list.
Task | Priority
Wake Up|1
Brush Teeth|2
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
```

Figure 9: Example of Adding Data

Figure 10 illustrates an example of removing data. As we can see, 'Eat Breakfast' was a task in the table, but now the user chooses to remove it. The updated list with the removed item is displayed afterwards.



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help
Assignment05 Assignment05.py
Project
Run: Assignment05 x
Your item was added to the list.
Task | Priority
Wake Up|1
Brush Teeth|2
Exercise|3
Eat Breakfast|4

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

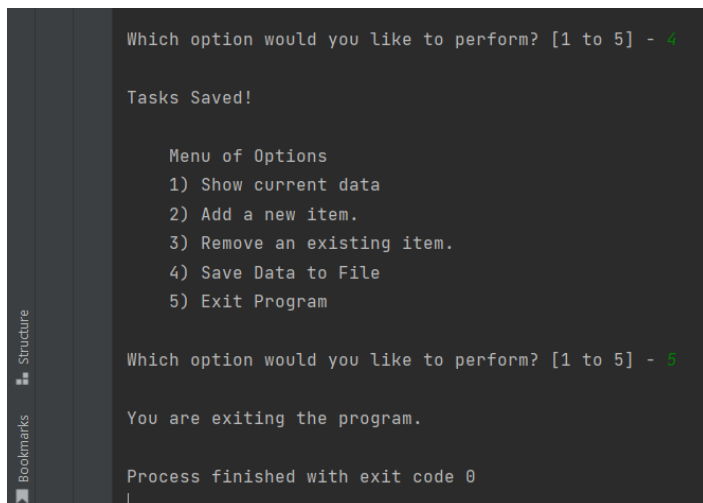
Which option would you like to perform? [1 to 5] - 3

Which task do you want to remove? eat breakfast
Eat Breakfast was removed. Updated list below.
Task | Priority
Wake Up|1
Brush Teeth|2
Exercise|3

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
```

Figure 10: Removing items and updating the table

Saving the file or exiting the program are the remaining options. We see these being executed in Figure 11 below. Once the file is save, the program prints out “Task Saved.” When the user selects to exit the program, the program prints “You are exiting the program.”



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

Tasks Saved!

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

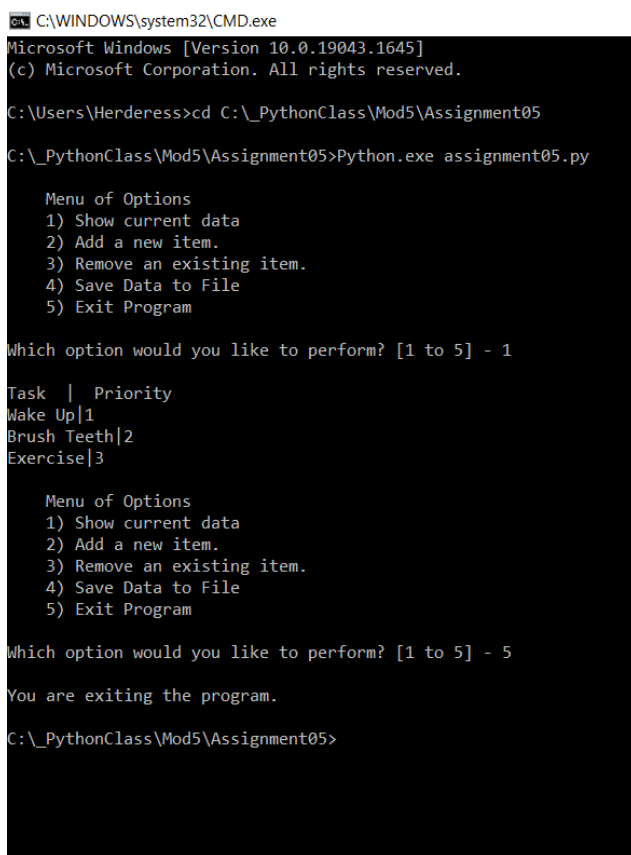
Which option would you like to perform? [1 to 5] - 5

You are exiting the program.

Process finished with exit code 0
```

Figure 11: Save the File and Exit

Figure 12 provides an image of the program running in the Windows Command Shell.



```
C:\WINDOWS\system32\CMD.exe
Microsoft Windows [Version 10.0.19043.1645]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Herderess>cd C:\_PythonClass\Mod5\Assignment05

C:\_PythonClass\Mod5\Assignment05>Python.exe assignment05.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

Task | Priority
Wake Up|1
Brush Teeth|2
Exercise|3

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

You are exiting the program.

C:\_PythonClass\Mod5\Assignment05>
```

Figure 12: Script running in Windows Console

Figure 13 show the data file 'ToDoList.txt' that our edited data is saved to. We can see that each row of data is separated by a comma.



Figure 13: 'ToDoList.txt' after running script

Summary

In this assignment, we learned to use try and except so that foreseen errors could be captured and the user could be clearly told what was going on and what to expect. We then used a while loop and a menu of options as in previous assignments to allow the user to choose what actions to take, but this time we practiced adding data or removing data that would be/was saved in a list of dictionaries. We use keys to retrieve values from dictionaries whether to compare them to user input in *if/elif* statements or to store them in data files. As always, we gave the user option to exit the while loop and exit the program.