Jon Larson

08-08-2022

Foundations of Programming: Python

Assignment 05

https://github.com/jtlarson/IntroToProg-Python

Working with Existing Code

# Introduction

This paper will detail the modification of an existing script to complete the desired functionality. The goal is to produce a complete program which uses lists and dictionaries to store data loaded from a file, display and modify the data, and write it back to the file if desired. The description of the coding process will be divided into sections covering the development of different functional elements of the completed program.

# Existing Code Structure

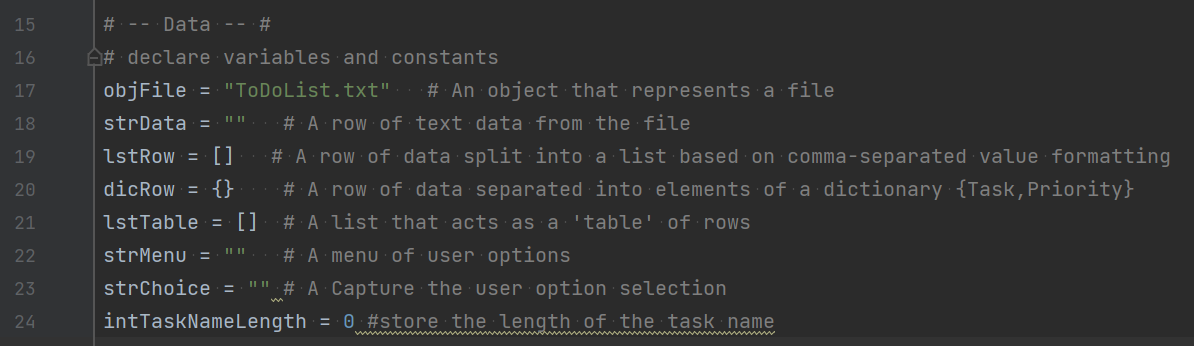
The assignment requires that we start with an existing code template and fill in marked “TODO” sections with additional code to complete the functionality described in the pseudo code. The code template contains an incomplete header that makes it clear we need to load data from the “ToDoList.txt” file into row dictionaries, then add those dictionaries in a list table (Figure 1).

Text

Description automatically generated

**Figure 1 – Template description**

The code template also contains a “Data” section with a list of defined and commented variables (Figure 2). I added the ‘listRow’ variable to hold rows of data pulled out of the ‘ToDoList.txt’ file, and the ‘intTaskNameLength’ for use in displaying the data. Since I am modifying an existing program, I adopted the same naming scheme (‘camelCase’ with type prefix) for new variables.



**Figure 2 – Declared variables section with added ‘listRow’**

Following the ‘Data’ (declaration of variables) section, there are a total of six code sections with “TODO” comments. You might notice that PyCharm automatically highlights comments that contain “TODO” for easy identification. In this template, the “TODO” sections include pseudo code instructions for loading the data from “ToDoList.txt,” and the functions required for each of the menu options which provides the main structure of the program (Figure 3):



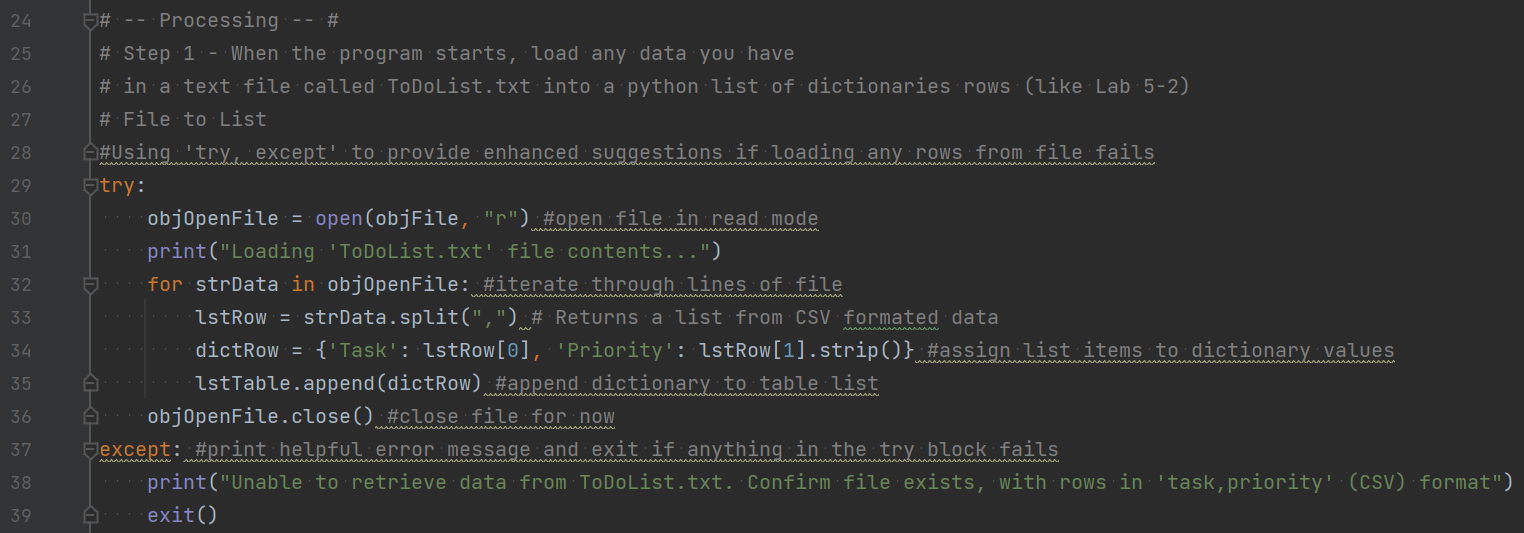
**Figure 3 - TODO segments for loading data and performing user-selected tasks.**

# Code Explanation

The comments for the “TODO” segments make it clear that we will need to first load the contents of the ‘ToDoList.txt’ file into a ‘list of dictionaries.’ Since the ‘ToDoList.txt’ file is not provided with this assignment, I created the file manually and added a few lines of ‘task,priority’ data to the file.

## Loading the Data

My code for loading the ‘ToDoList.txt’ file makes use of the ‘try, except’ functions which were introduced this week (Figure 4). The code for opening the file and extracting the rows into dictionaries and a table list is all contained within the ‘try:’ code block. If that fails, the code within the ‘except:’ code block prints an enhanced error message that give the user a better idea of what might have gone wrong, and then exits the program.

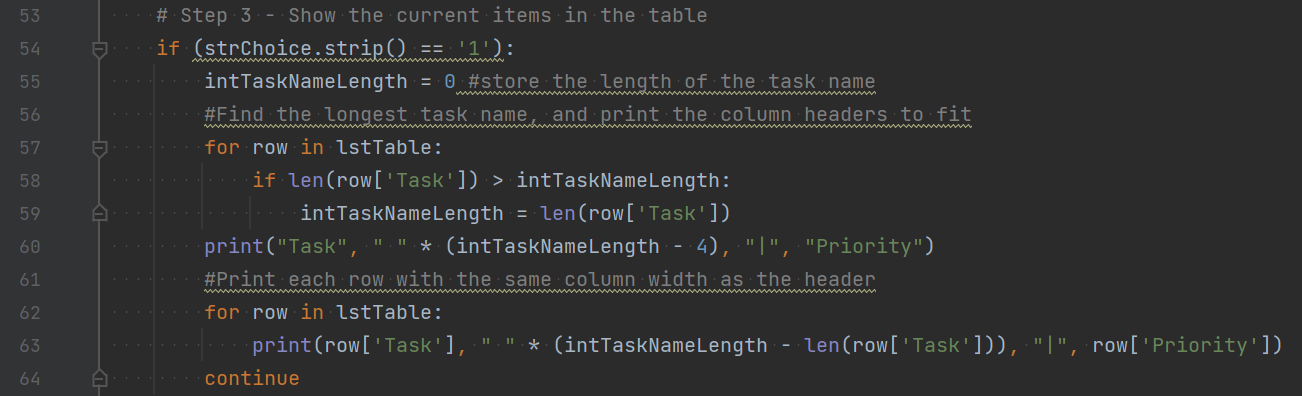


**Figure 4 - Processing section uses try, except to handle errors**

Within the ‘try:’ block, the ‘ToDoList.txt’ file is opened in read mode. A ‘for’ loop iterates through the rows of the file and splits the rows into list items based on the commas found in each row. The list items are then assigned as values to a dictionary with keys named “Task” and “Priority,” and that dictionary is appended to the ‘lstTable’. If this ‘import process’ doesn’t turn up any missing/extra/invalid items in the file and the file is able to close successfully, then the script continues on to the ‘Main menu’. If there is an error—such as a file row that doesn’t have two comma-separated values—then the script will switch to the ‘except:’ statement which prints an error message and exists immediately.

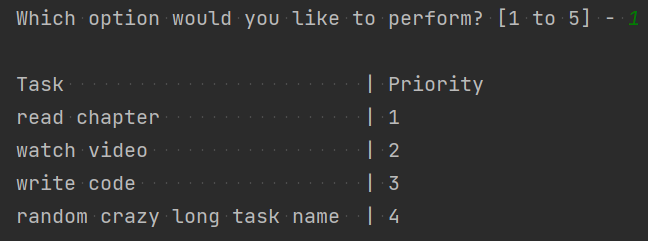
## Displaying Current Data

The ‘while’ loop that powers the main menu doesn’t have a ‘TODO’, so that code is left as is, and we move on to the code that performs the first menu option: “Show current data.” My code for this section is in Figure 5:



**Figure 5 - Code for "showing current data"**

In this section, I chose to use two ‘for’ loops to examine the existing data. The first ‘for’ loop scans the number of characters of each ‘task’ value and stores the largest value. I use this value to print a “Task | Priority” header that is customized to fit the longest task name so that the “|” column separators and “Priority column are always aligned vertically. The second ‘for’ loop prints the ‘task’ and ‘Priority’ values with a custom spacing that takes the length of the task name into account. The benefit of this can be seen in Figure 6, where the display of tasks and priorities is still aligned in spite of having a “random crazy long task name”:



**Figure 6 - Displaying current data in PyCharm**

## Adding a New Item

The second menu option gives the user an opportunity to add new data to the table. The user is asked to input a new task and priority, both of which are ‘stripped’ and stored as strings. I then assign these new values to the ‘dictRow’ dictionary (using the ‘replace’ method to filter out commas entered by the user) and append the dictionary to the ‘lstTable’ (Figure 7). I would also note that while I chose to use numerical indicators of ‘priority’ in my display example (Figure 6), a string such as ‘high’ or ‘low’ would also work with this code since the priority values are processed as strings.

Text

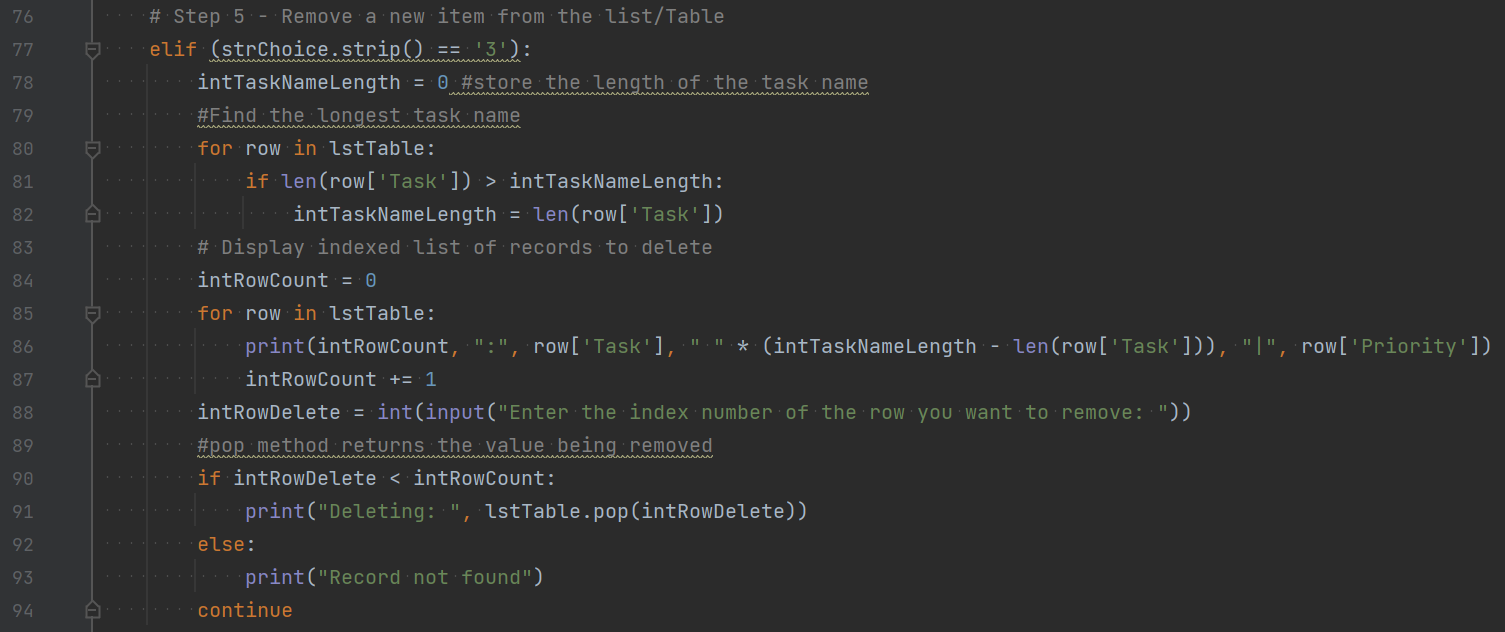
Description automatically generated

**Figure 7 – Code for “Adding a new item”**

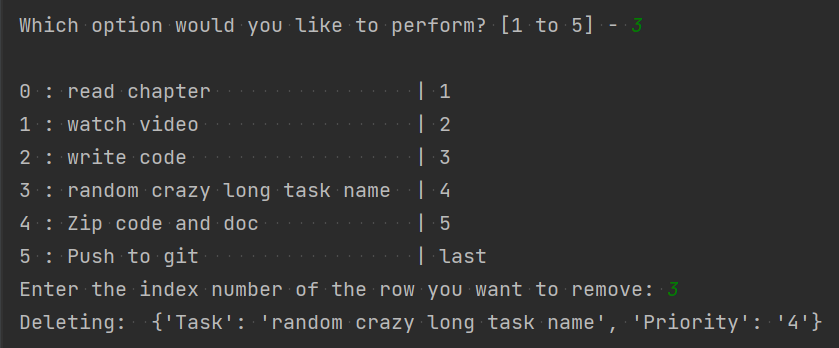
## Removing an Existing Item

The third menu option allows the user to remove an existing item from the table. Since I want to show the table, I re-use the code that sets the ‘intTaskNameLength’ again. This would normally be a good opportunity to package this code as a custom function, but the assignment specifies that we should not use functions in this assignment.

To give the user a reference index to guide their selection, I created a new variable ‘intRowCount’ that is used within a ‘for’ loop to display the index of each row (Figure 8). The user is then presented with an selection prompt for the desired index number. If they enter a valid index number, I use the ‘pop’ method within a print command to simultaneously display and remove the desired record (Figure 9). If they enter an invalid number, then the user is directed back to the main menu with an error message.



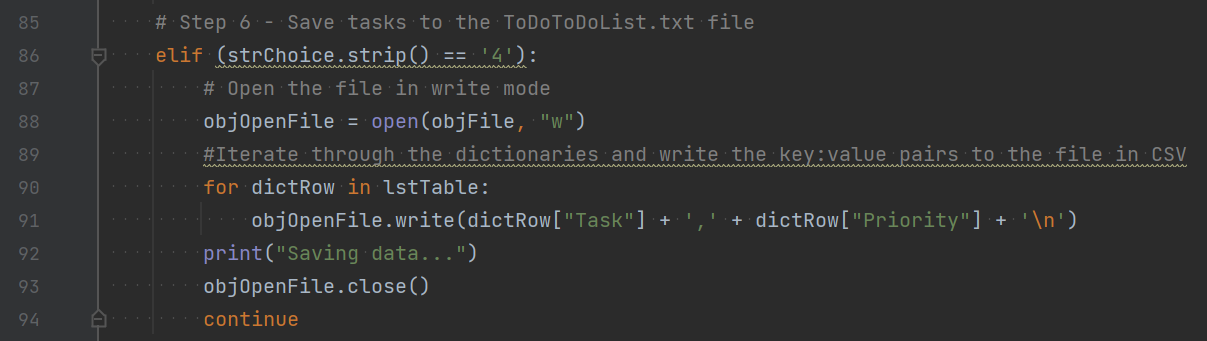
**Figure 8 - Code to "Remove an existing item"**



**Figure 9 - Example of deletion process in PyCharm**

## Save Data to File

To save the data back to the ‘ToDoList.txt’ file, I open the file in ‘w’ mode, which will overwrite the contents of the file. This is OK because we imported the full contents of the file at the beginning.



**Figure 10 - Code to save data to ‘ToDoList.txt’**

The ‘for’ loop in lines 90-91 (Figure 10) iterates through the dictionary rows. The values are referenced via the dictionary keys and written back to the file with a comma separator and “\n” at the end of each line. The code section ends with a simple verification that the data is saved and closing the file.

## Exiting the program

While this ‘elif’ statement included a “TODO” comment, there really isn’t anything that needs to be done. So I simply print a farewell message and leave the ‘break’ as-is so that the ‘for’ loop is ended.

# Running the code

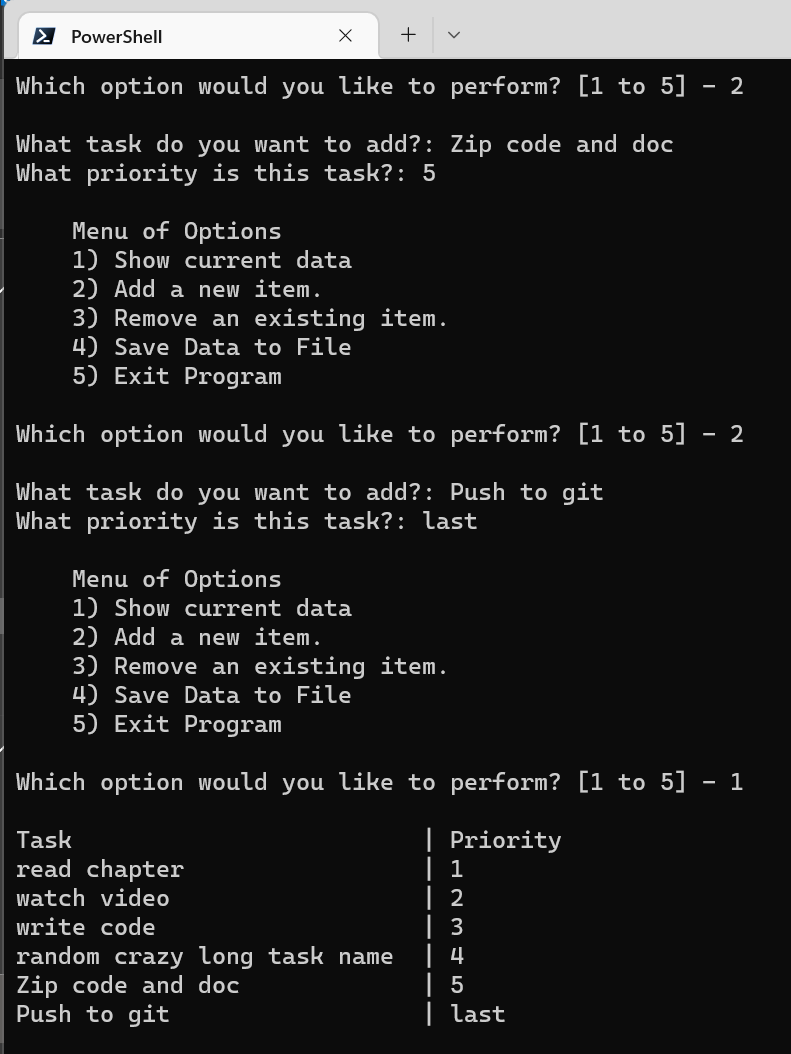
Below are screenshots of the program working within PowerShell. To start, Figure 11 shows the initial menu and display of current data:

Text

Description automatically generated

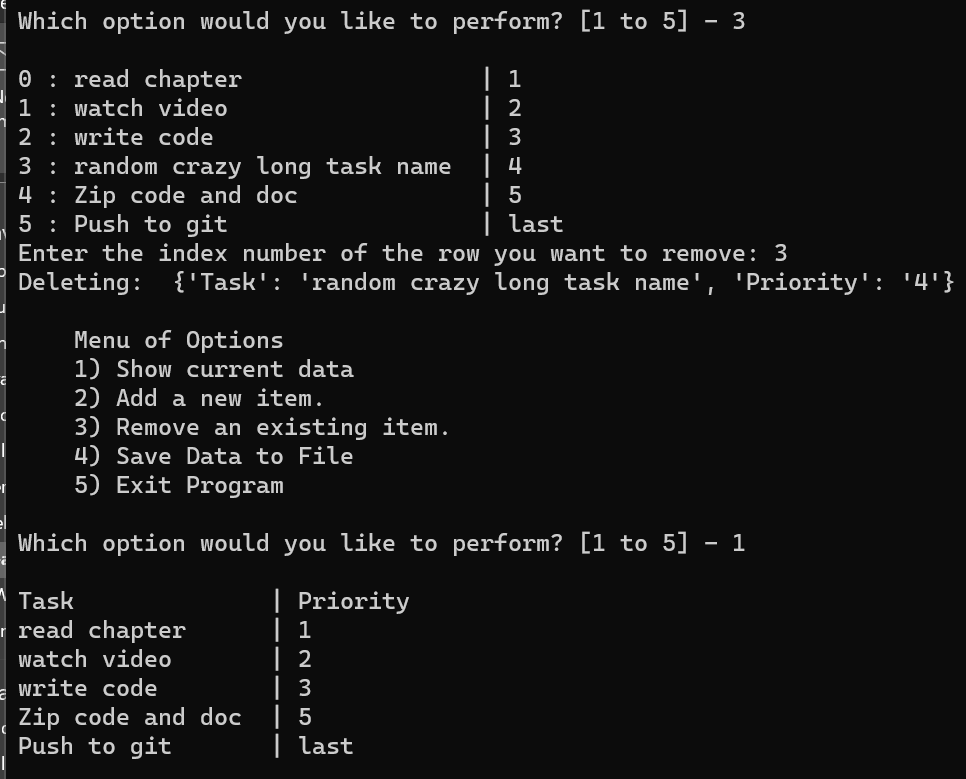
**Figure 11 - Menu and displaying current data in PowerShell**

I then enter a couple new tasks and priorities and display the changes in Figure 12:



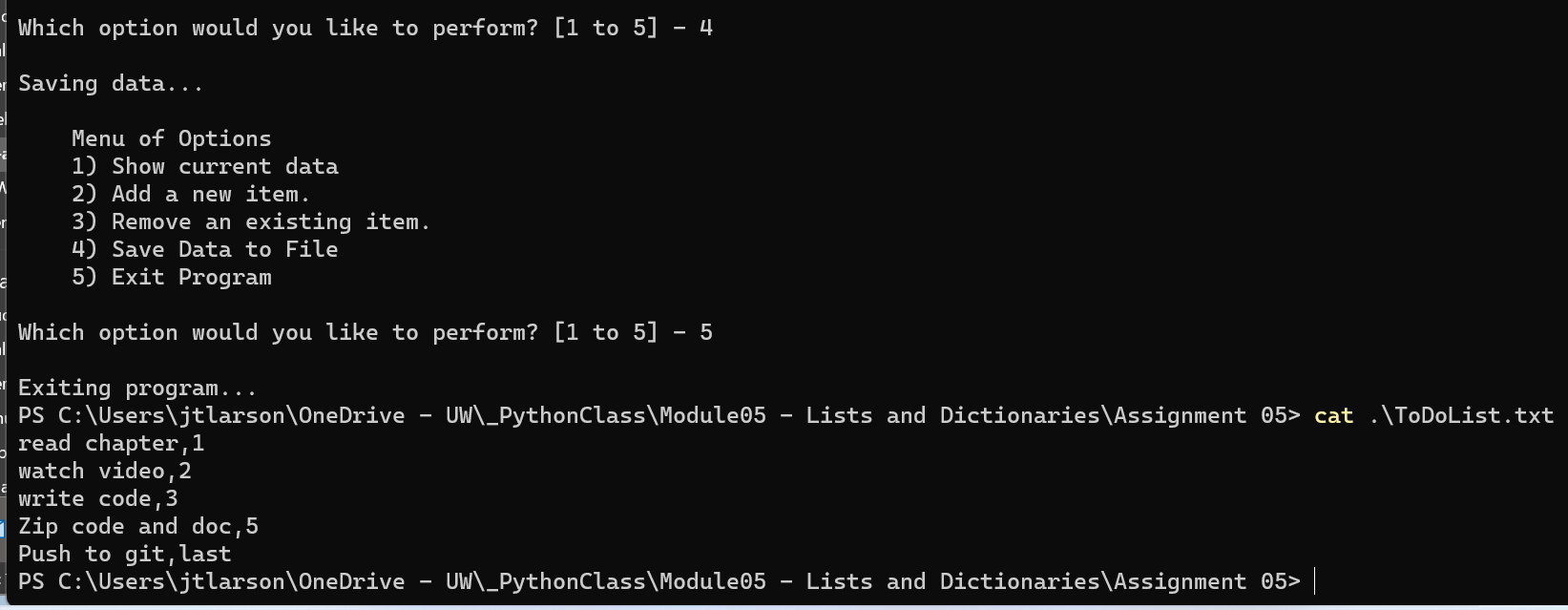
**Figure 12 - Adding and displaying data in PowerShell**

I remove a record and display the data again (notice the ‘Task’ column resizes to fit) in Figure 13:



**Figure 13 - Deleting an existing item in PowerShell**

The final screenshot in PowerShell (Figure 14) captures menu options 4-5, and the output from the updated ‘ToDoList.txt’ file:



**Figure 14 - File save, program close, and file contents display**

## Running in PyCharm

The output is identical in PyCharm. Figures 15-17 shows the program operation stages, and Figure 18 shows the resulting text file when we make the same changes as used with PowerShell.

Text

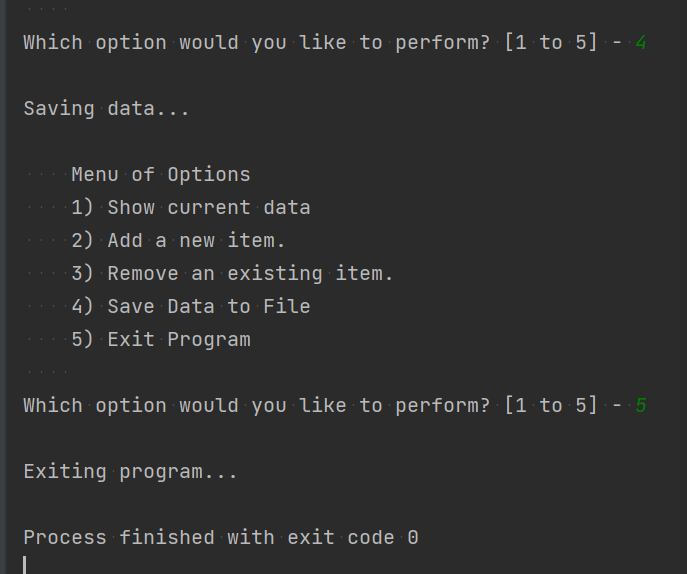
Description automatically generated

**Figure 15 - Opening, displaying, and adding entry in PyCharm**

Text

Description automatically generated

**Figure 16 - Deleting a row in PyCharm**



**Figure 17 - Saving data and closing program in PyCharm**

Text

Description automatically generated

**Figure 18 - ‘ToDoList.txt’ contents after PyCharm program close**

# Summary

In this document I described my evaluation process of the pre-existing code and requirements. I also described the code sections that I added to complete the tasks described in each of the TODO segments. I discussed some features that I chose to use for the first time to help reduce errors from user input (strip() and replace() methods for new records; ‘if’ conditional to confirm valid record deletion) and for better feedback to the user (‘try, except’ during data loading). I then demonstrated the program operation (as well as proper output to the text file) in PowerShell and PyCharm.