Task Inventory Text File with Lists

# Introduction

For the fifth assignment, the concepts of lists and dictionaries were explored. A text file storing strings for tasks and priority levels was read and overwritten with data inputs from the user. The practice of “Separations of Concern” allowed the script to be organized into three sections of data declaration, data processing and presentation. Functions were included to group a set of programming statements that can be later referenced by name. Once a skeleton script was created, I was able to save it as a template for later assignments. Finally, I was able to create a GitHub account and loaded my final script and knowledge document to the network server so that it was easier to share my program architecture and concepts with others.

# Lists and Dictionaries

## Lists

**Lists** are tuples but more flexible in that they are not immutable and have functions that allow for further analysis. For formatting purposes, they use brackets instead of parentheses and do not require a “,” to indicate there is only one object in the list. Much like tuples, they can be multi-dimensional and form arrays as lists are nested within other lists. They can be written into text files by having them written column by column or row by row depending on the level of the list in a multi-dimensional list. They have plenty of built-in functions to manipulate the data stored within them. They also can be stored in a .txt file by writing in to the file row by row. The split function allows data to be read from a .txt file row by row.

## Dictionaries

**Dictionaries** use key(character) subscripts with the **brace operators{}** to indicate that a variable should be a dictionary. Dictionary keys act like columns in a database in that they identify a domain of data types. Dictionaries act like rows of data. The dictionary keys and dictionaries combine to form a two-dimensional collection of data. Just like lists, dictionaries have built-in methods like items(), values() and keys().

# Recommended Script Improvements

## Separation of Concerns

The **Separation of Concerns Concept** is a design principle for separating a computer program into distinct sections. Each section addresses a separate concerns, or set of information that affects the code of the program. Most code is divided into three different sections of **data statements, processing and presentation**. These can be organized into which ever way best meets the language and design of the program. This policy does not need to be strictly followed as sometimes the data processing can be broken up by presentation sections.

## Functions

**Functions** group a set of programming statements so they may be referenced by a given name. When Python gets to the line that calls to the function, it jumps to the section where the function is defined. It runs its code within the function and then returns to the line of code where the function was called before continuing its run.

## Script Templates

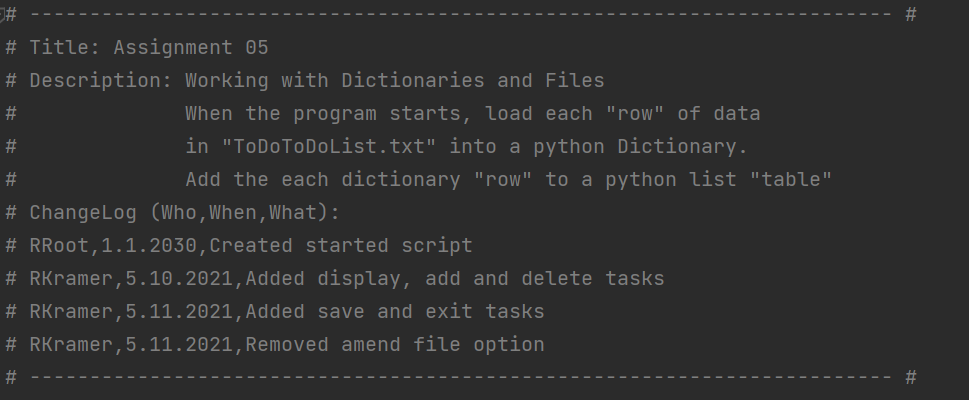
**Script Templates** allow the user to save custom templates that can be saved for later use

## Error Handling

**Error Handling** improves scripts by managing errors. It traps errors with try-except constructs. This allows for personalized or more generalized error messages. These are easier to understand. If the error occurs in the grouped statements, Python moves to another set of statements where one can handle the error in their own way.

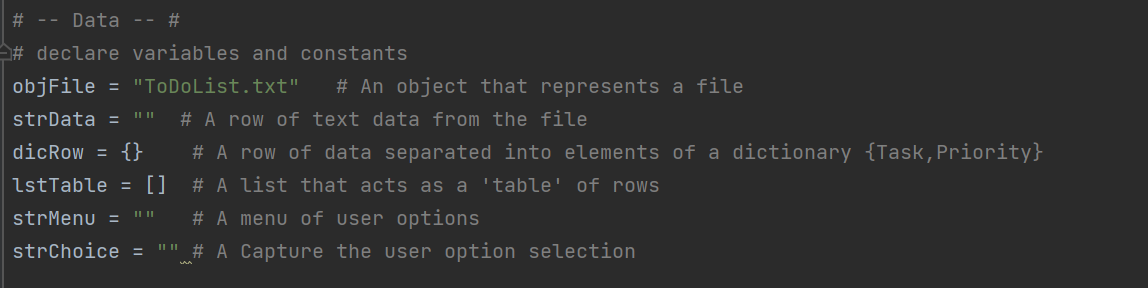
# Code Outlining

I began my script file with the script header (Figure 2). The original code was provided by Professor Root so he was credited with creating the script. To alter the program for this assignment, the first major change was adding the display current data from reading the existing .txt file option. Further changes were made by allowing for the adding and deleting of tasks in the form of rows or rather dictionaries. The save and exit tasks were then added. Finally, to allow for the .txt file to be properly saved and thus read when the program was reran, I removed the amend .txt file feature that I had copied over from the previous assignment as it was creating an error where the .txt file could not identify the rows properly.

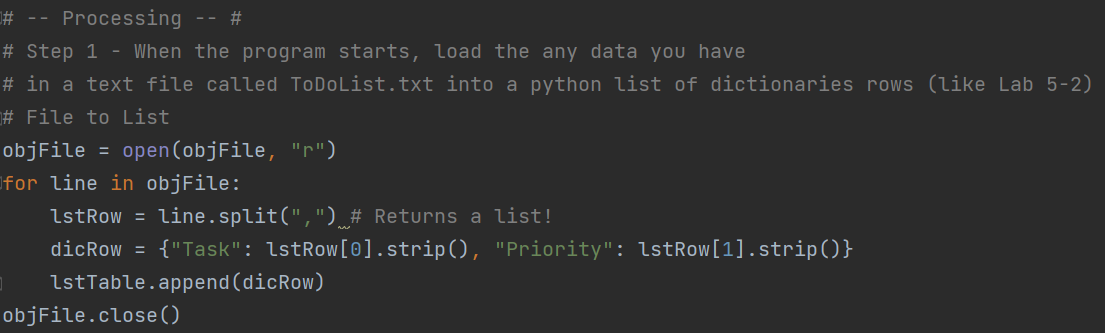


**Figure 1. Script Header**

The first concern addressed was the data (Figure 2) where the variable and constants for the dictionary and table were declared.

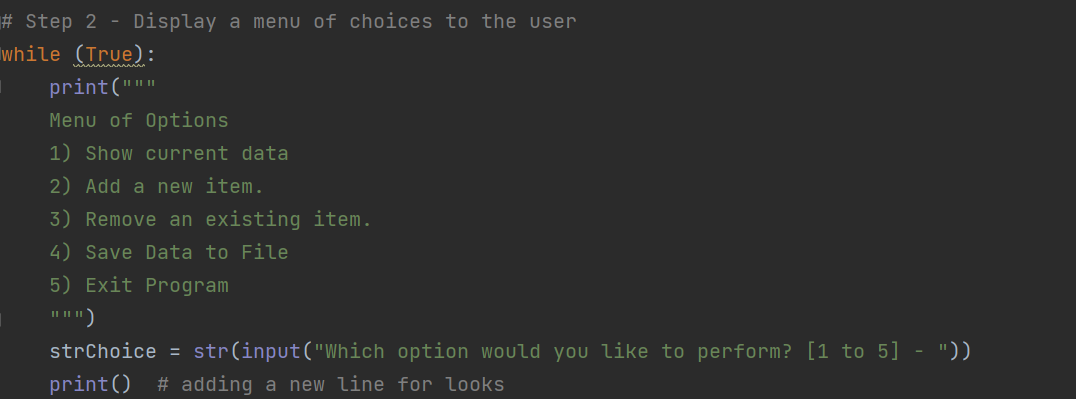
**Figure 2. Data Section**

The processing section (Figure 3) began with reading the .txt file that was declared above as objFile. It splits each line based on the separator “,” and then establishes the dictionary keys of Task and Priority for the appropriate string columns.



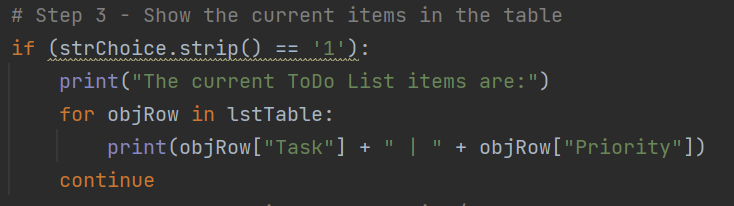
**Figure 3. Processing .txt File**

A while statement (Figure 4) was next to start the condition where the program will keep accepting data until the break statement is enacted with the input of 5 for the option input. After being displayed the menu options immediately, the user is prompted to go to the submenu they prefer.



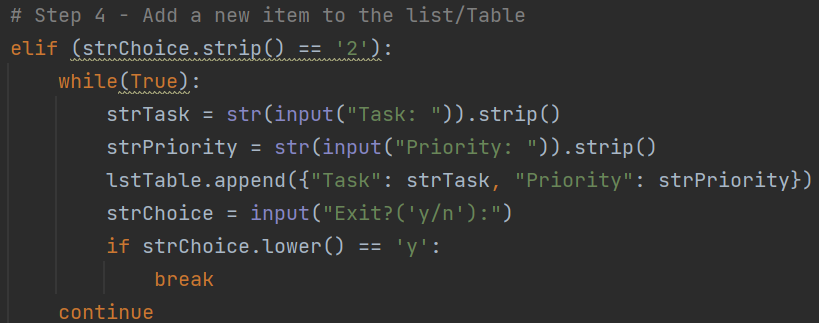
**Figure 4. Display User Option Section**

If “1” is entered, the user is displayed the current ToDo list items. This section serves as the Display Data section within the processing data section. A “|” acts as a separator.



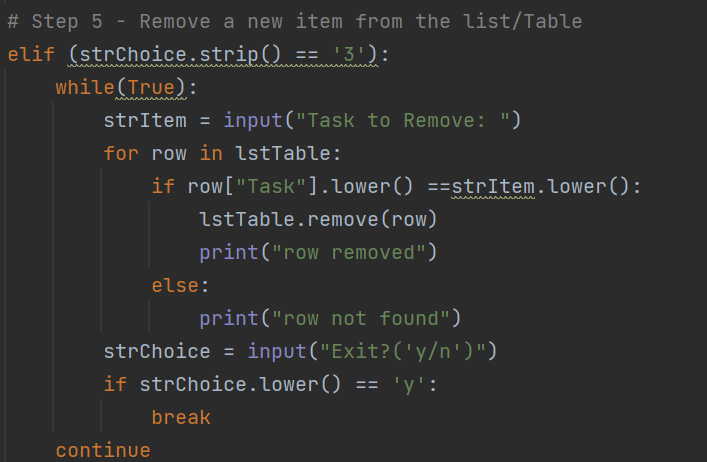
**Figure 5. Option 1 for Displaying the Data**

If “2” is entered, the user is asked to input data for the item and value (Figure 6). They are both recorded as strings which are then placed in a dictionary that is directly appended to the lstTable. This continues until the user indicates that want out of this inner while loop with a “n” input.



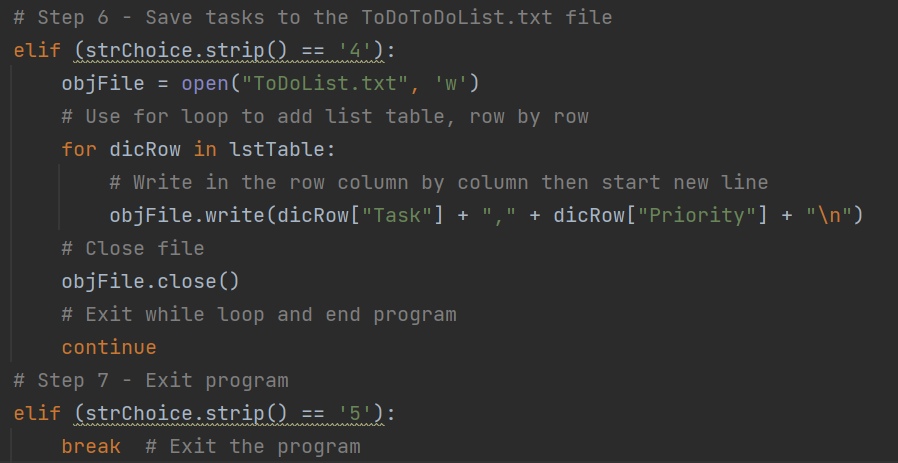
**Figure 6. Option 2 for adding new data to list table**

Option 3 removes an item from the list table from searching through the table (Figure 7), row by row or rather dictionary by dictionary until the identifying data for the Task key is found and then removed through the remove function for dictionaries. This continues until either the row is found in the table or all rows are examined and the task is found not to exist. The user is then prompted if they want to continue their deletion task or be taken back to the main menu.



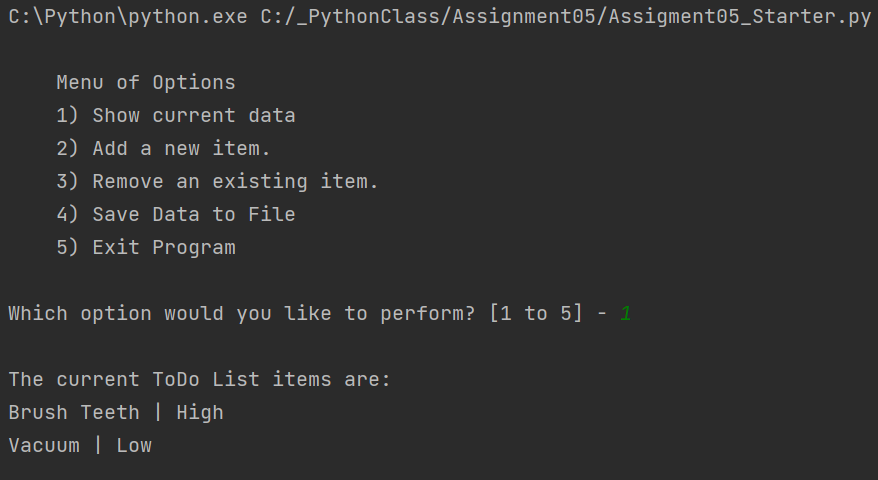
**Figure 7. Option 3 for removing data from list table**

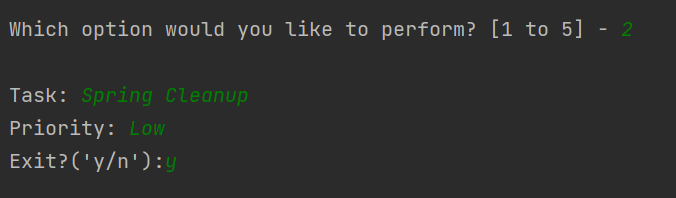
Option 4 leads to the .txt file being open and then overwritten (Figure 8). Once it is overwritten, the program restarts to the top of the script. Option 5 leads to a break that ends the script.

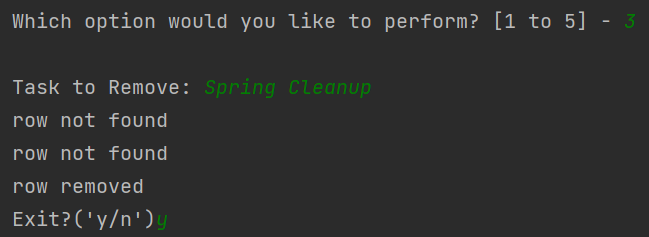


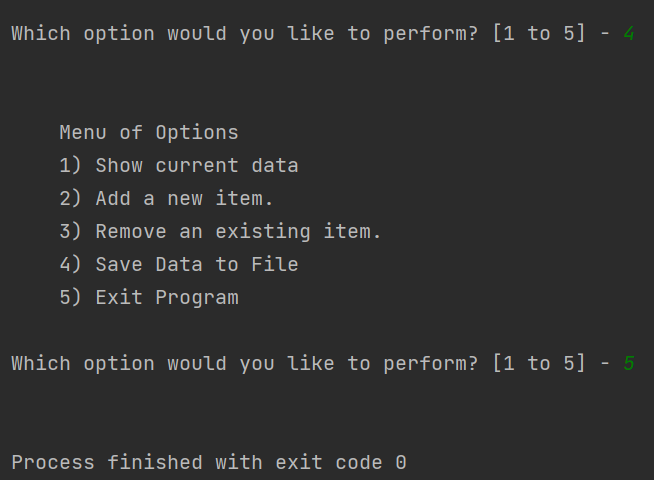
**Figure 7. Options 4 and 5 saving tasks to .txt file and exiting script**

# Program Test

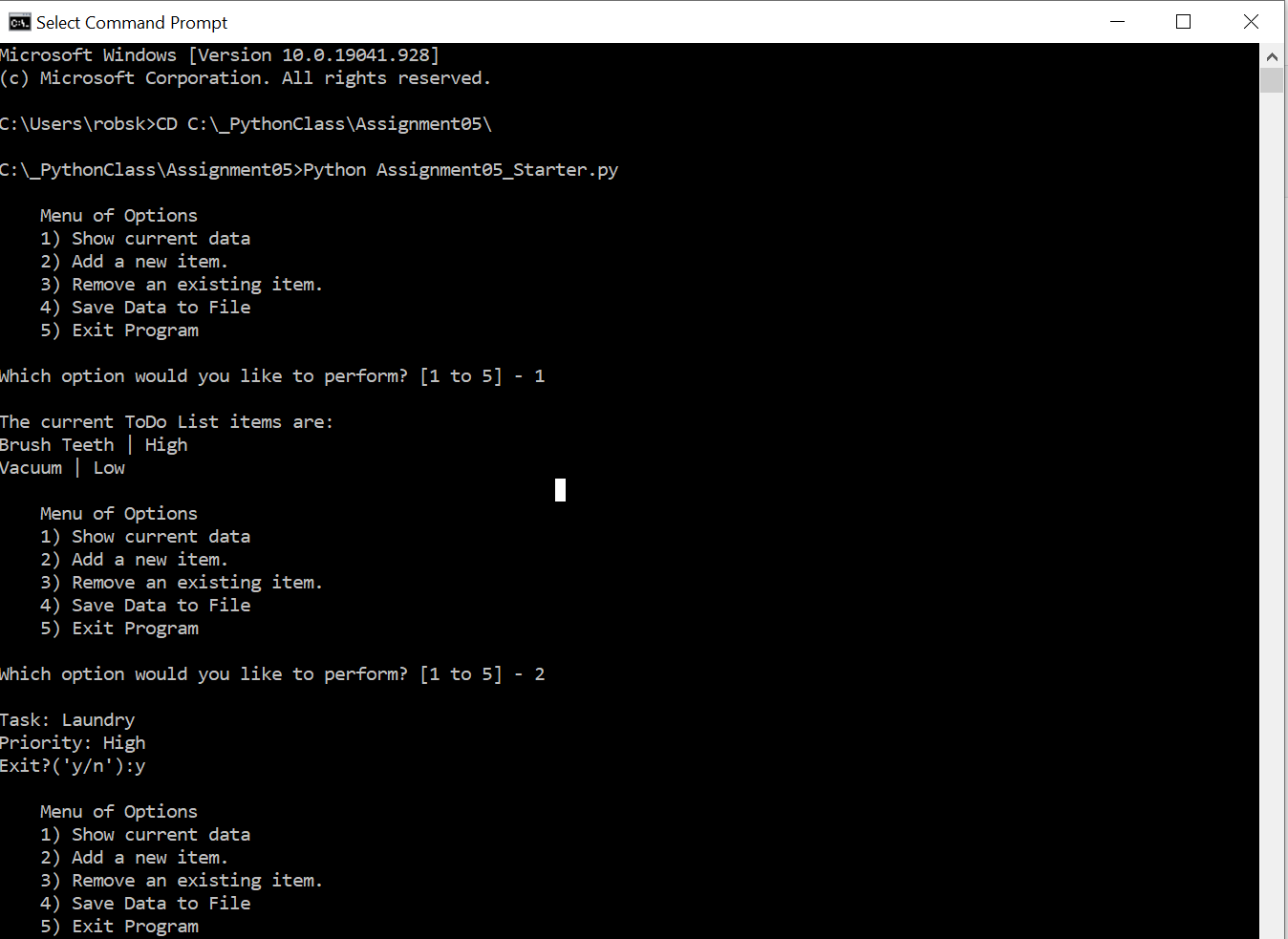


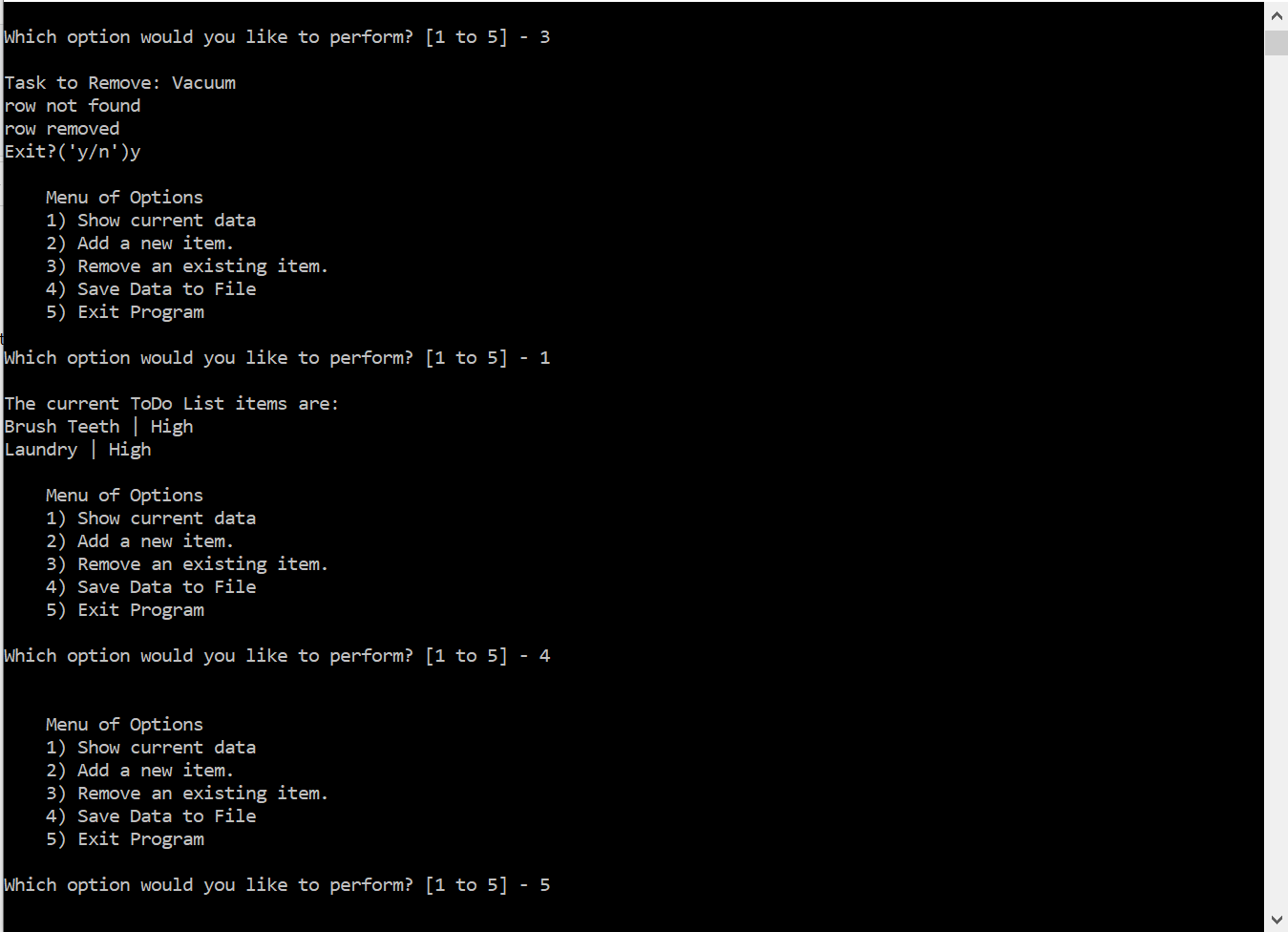




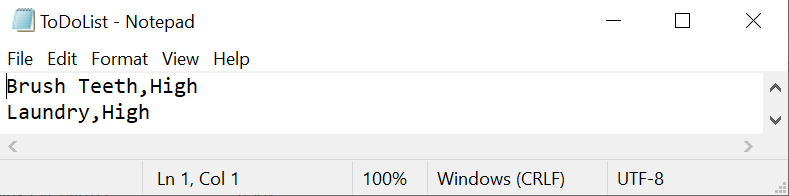


**Figure 8. Output from running HomeInventory.py in pyCharm**





**Figure 9. Output from running HomeInventory.py in Command Prompt**



**Figure 10. Evidence of the text file’s creation with a pair of inputs**

C:\\_PythonClass\Assignment05\ The program test was run through Pycharm (Figure 8) and through the command prompt (Figure 9). They both proved to be successful as the file was read and the program could display data, add to the list table, delete from the list table, save to the file and exit the program.

# Summary

This assignment was a timely exercise in how to better manage data from a user interface. It also honed our skills in reviewing code that is necessary and how to sort through existing code for what needs to be added to and subtracted. I plan to organize my scripts better in terms of separation of concerns so that a reviewer can better understand my code from their first review.