Jessica Carnes

November 24, 2020

Foundations of Programming: Python

Assignment06

<https://github.com/Jessicawyn/IntroToProg-Python>-Mod06

Using Functions to Organize Code

# Introduction

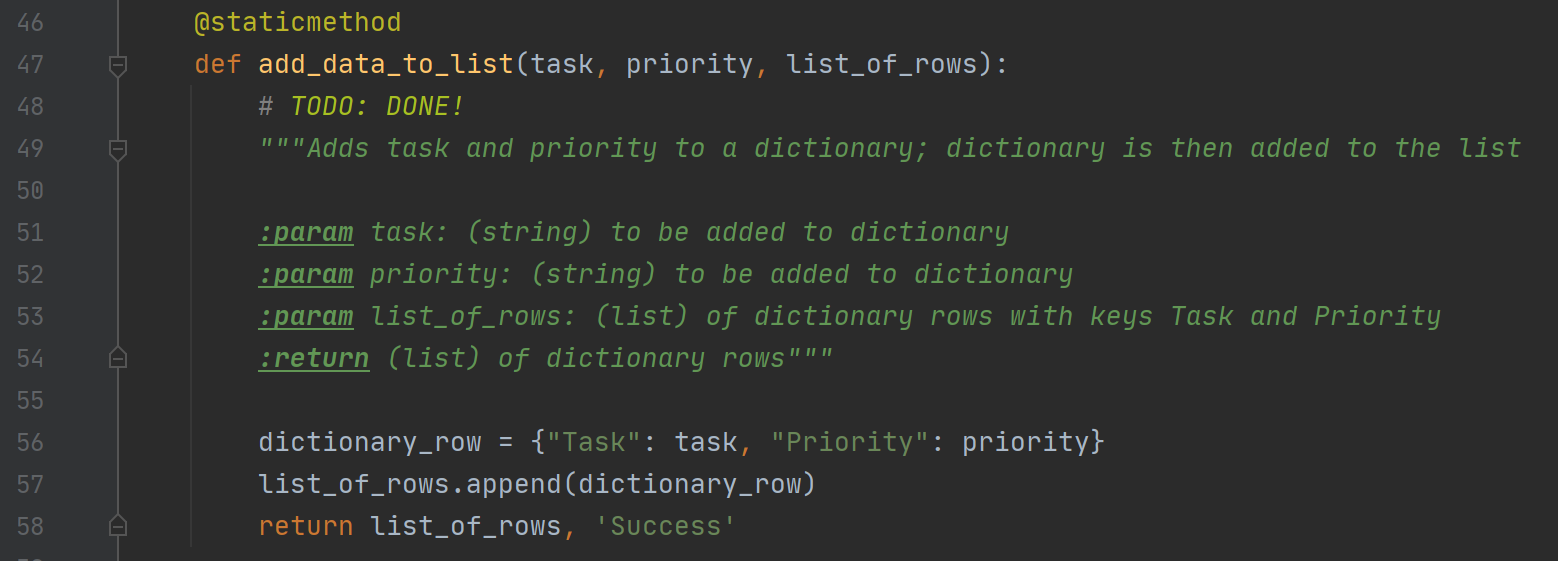
The assignment this week was to use functions to help organize code. In general, a script should be laid out with a data section, a processing section, and a presentation section. Functions, along with the use of classes, help maintain this layout and keep the code separated. Functions also allow for users to reuse code in multiple sections of the main body of the script. The following discusses the updates to the starter code made to complete the To Do List assignment.

# Updating the Processor Class Functions

In the Processor class, the function read\_data\_from\_file() was already completed in the assignment starter document.

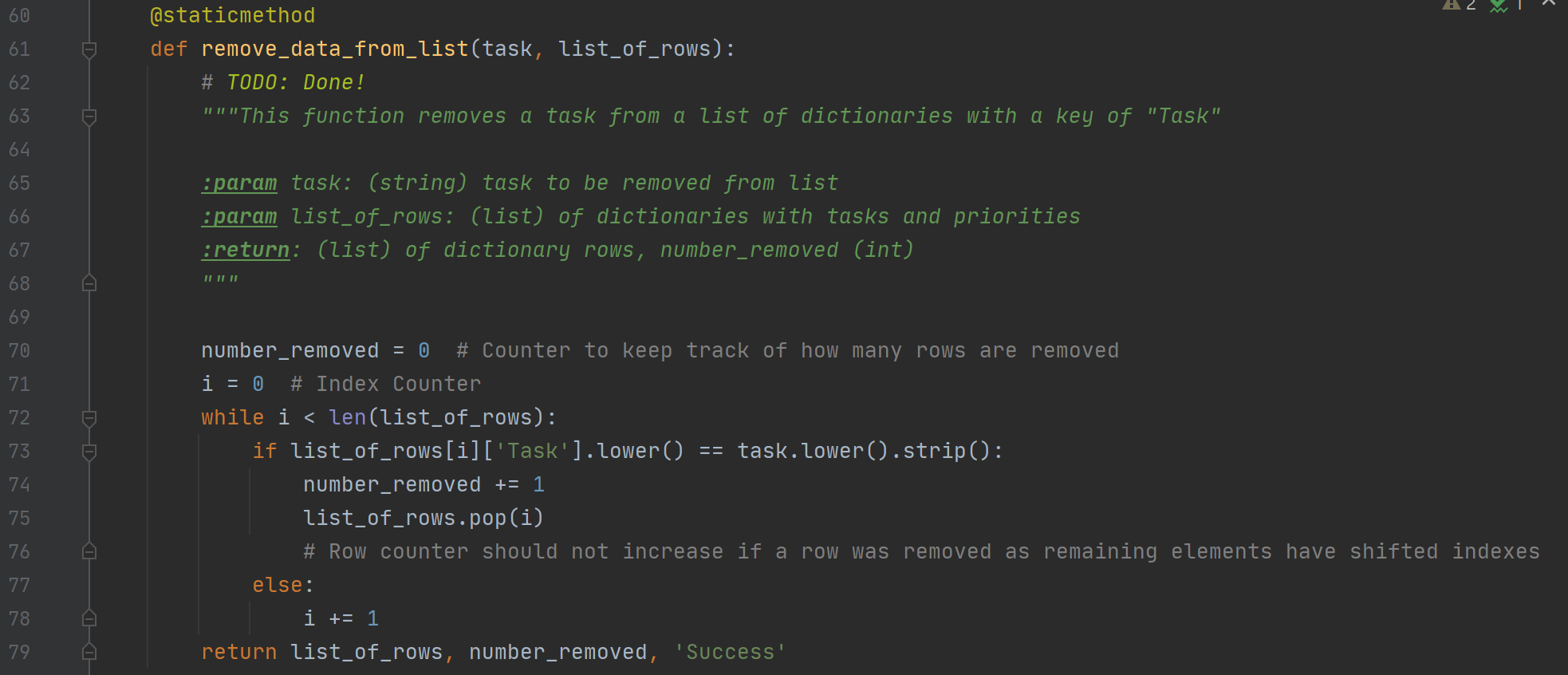
## Add Data to List

The first function that needed to be completed was add\_data\_to\_list. This function requires the string parameters task and priority, as well as list\_of\_rows which needs to be a list. The function takes the task and priority, adds them to a dictionary with the keys “Task” and “Priority”, then appends the list passed to the function with the newly created dictionary.

  
Figure 1: The completed function for add\_data\_to\_list()

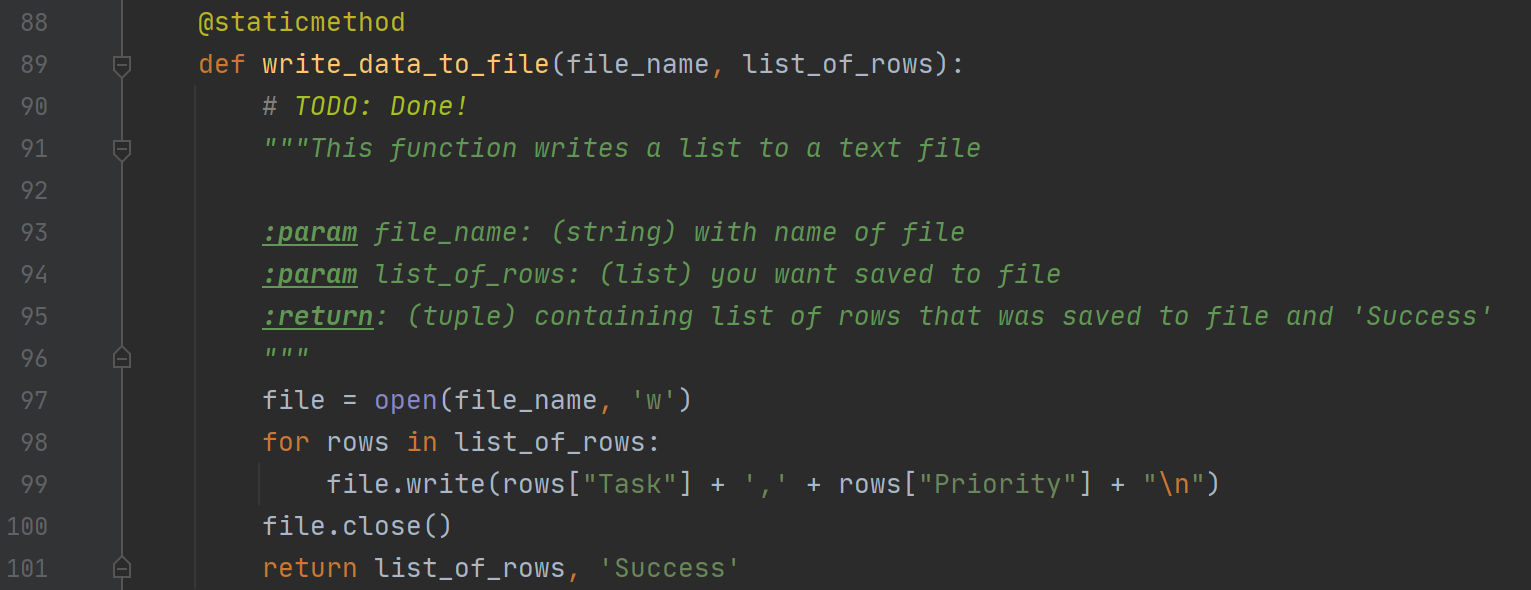
## Remove Data from List

The second function that needed to be completed was remove\_data\_from\_list. This function requires the parameter task which is a string that should be removed from the list as well as the parameter list\_of\_rows which is the list to search for and remove the task. The function searches through the list\_of\_rows, keeping track of how many times the task is found and removed. It then tells the user the number of tasks removed from the list and returns the updated list\_of\_rows as shown in figure 2.

  
Figure 2: The completed function remove\_data\_from\_list

## Write Data to File

The write\_data\_to\_file function requires a file\_name (string) and a list\_of\_rows (list). The list of rows must be a list of dictionaries with the keys “Task” and “Priority”. The function opens the file in write mode, then writes the task and priority separated by a comma of each of the dictionaries in the list to a row in the text file. The file is then closed, and the function returns the list of rows as well as a ‘Success’ status as shown in figure 3:

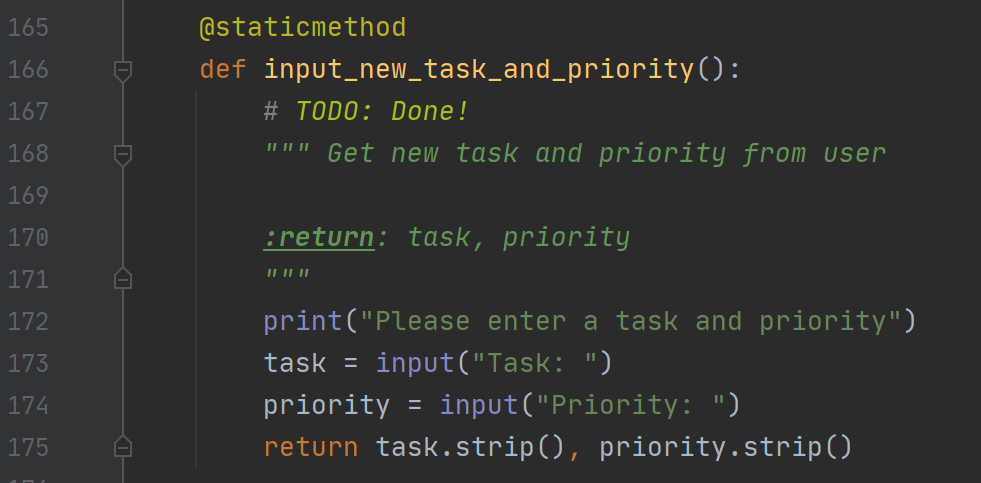
  
Figure 3: The completed function write\_data\_to\_file

# Updating the IO Class Functions

The input/output section of the script contains the code that displays and collects data from a user. In this class the functions print\_menu\_tasks, input\_menu\_choice, print\_current\_tasks\_in\_list, input\_yes\_no\_choice, and input\_press\_to\_continue were already completed in the starter document. The following functions needed to be updated.

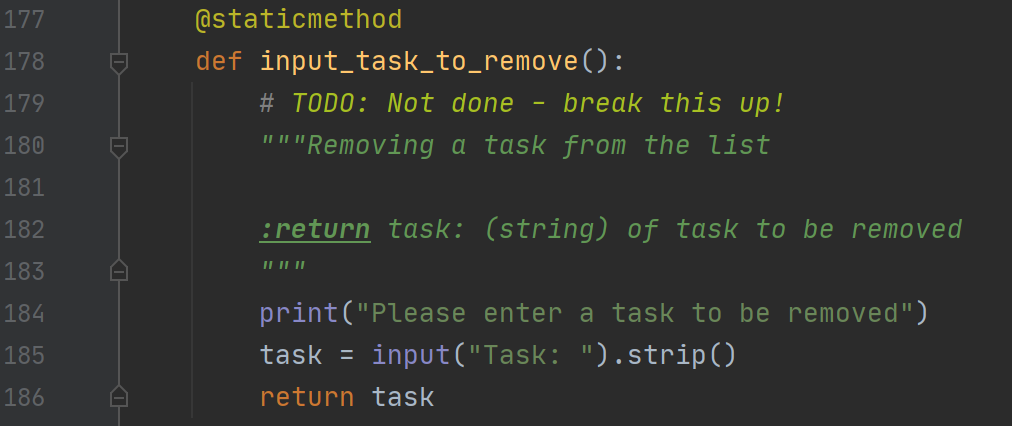
## Input New Task and Priority

The function input\_new\_task\_and\_priority does not require any parameters. The function prints a message to the user asking them to enter a new task and priority, then uses the input function to collect a task and priority from the user. The function then returns the task and priority as shown in figure 4.

  
Figure 4: The completed function input\_new\_task\_and\_priority

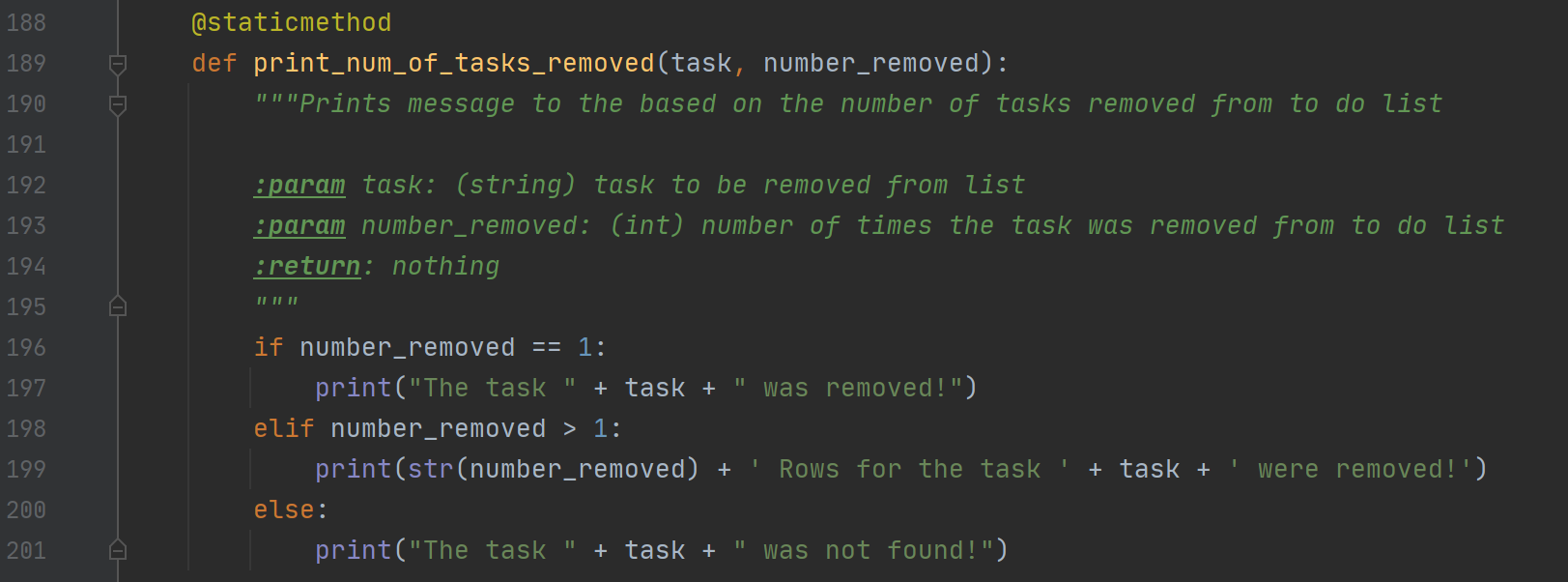
## Input Task to Remove

The function input\_task\_to\_remove also requires no parameters. The function asks the user to input a task to be removed then collects this task by using the input function. The function then returns the task the user entered after it has been stripped of blank spaces or other hidden characters as shown in figure 5.

  
Figure 5: The completed function input\_task\_to\_remove

## **Print Message to User based on number of Rows removed**

The function print\_num\_of\_tasks\_removed was created to display a custom message to the user based on the number of times a task was found in the list and removed. The function requires the parameters task, which is the string the user searched to have removed from the list, and number\_removed which is an integer that counts the number of times the task was removed. The function prints out a message to the user based on these two parameters and nothing is returned as shown in Figure 6.

  
Figure 6: The additional function print\_num\_of\_tasks\_removed

# Main Body of the Script

All of the functions are initially defined in the script and loaded to memory when it is launched, but nothing is truly ran until the code has reached the main body. In the main body of the To Do List script a while loop runs giving the user the following five menu options to choose from:

1) Add a new Task

2) Remove an existing Task

3) Save Data to File

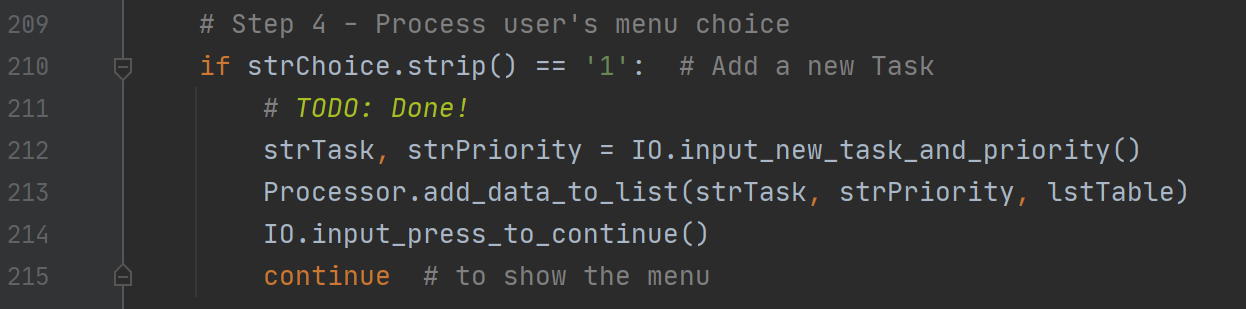
4) Reload Data from File

5) Exit Program

The code within menu options 1 through 4 is completed using the functions defined previously in the script. If the user chooses option 5 the menu prompt while loop is broken, and the program exits. The following describes how the functions are used to achieve the desired results in menu options 1 through 4.

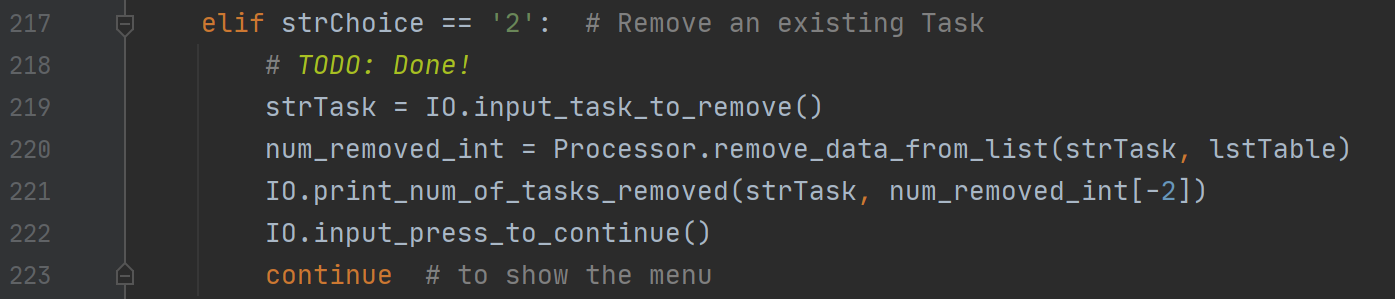
## Option 1: Add a new Task

When the user enters menu option 1, they want to add a new task to the list. This branch of code initially captures the task and priority to be added by using the input\_new\_task\_and\_priority() function. This function returns a task and priority which are captured in the variables strTask, strPriority. These two variables are then passed as arguments for the add\_data\_to\_list function along with the list, lstTable. The function adds the strTask and strPriority to a dictionary which is then appended to the lstTable list as shown in figure 7.

  
Figure 7: Using the previously defined functions to add a new task

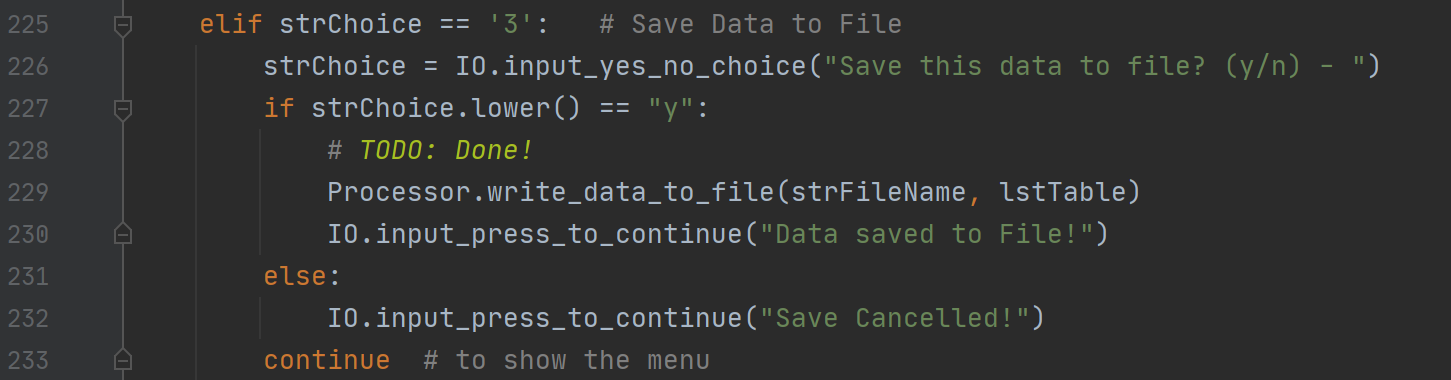
## Option 2: Remove an Existing Task

The code block for when a user enters menu option 2 to remove a task begins by capturing the task the user wants to remove in the variable strTask by calling the input\_task\_to\_remove() function. The variable num\_removed\_int is set to capture the return of the remove\_data\_from\_list function. This function searches the lstTable list table for the strTask variable and removes all rows where this is found. The function then returns the updated list and a counter for the number of times this task was found. The print\_num)of\_tasks\_removed function takes in the arguments of the task that was searched for (strTask) and the number of time it was removed. The number of times removed is found in the second to last index of the return from the remove\_data\_from\_list function therefore the num\_removed\_int with an index of -2 returns this integer needed as shown in figure 8.

  
Figure 8: Calling the previously defined functions to remove a task

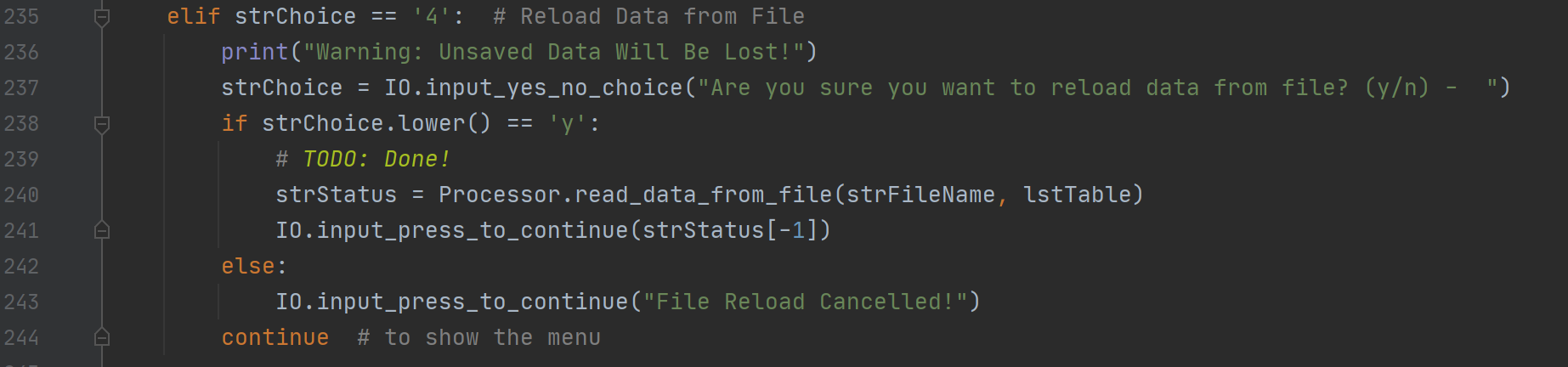
## Option 3: Save data to File

To save the data back to the file the user enters menu option 3. This initially prompts the user to enter y/n is they want to save the data. If they enter y, then the function write\_data\_to\_file is called and variables strFileName and lstTable are passed as arguments. This function writes the data in lstTable to the file strFileName. If the user enters n then a message appears saying the save is cancelled as shown in figure 9.

  
Figure 9: Saving data to the file through previously defined functions

## Option 4: Reload data from File

If the user has made changes to their list that they would like to discard, then they select menu option 4. This option initially asks the user to confirm if they would like to reload the file by calling the input\_yes\_no\_choice function. If they enter y then the script calls the read\_data\_from\_file function which initially clears the data in the lstTable variable then opens the file stored in the strFileName variable in read mode, then writes the data from the file into a list of dictionaries into lstTable. The calling of these functions is shown if figure 10.

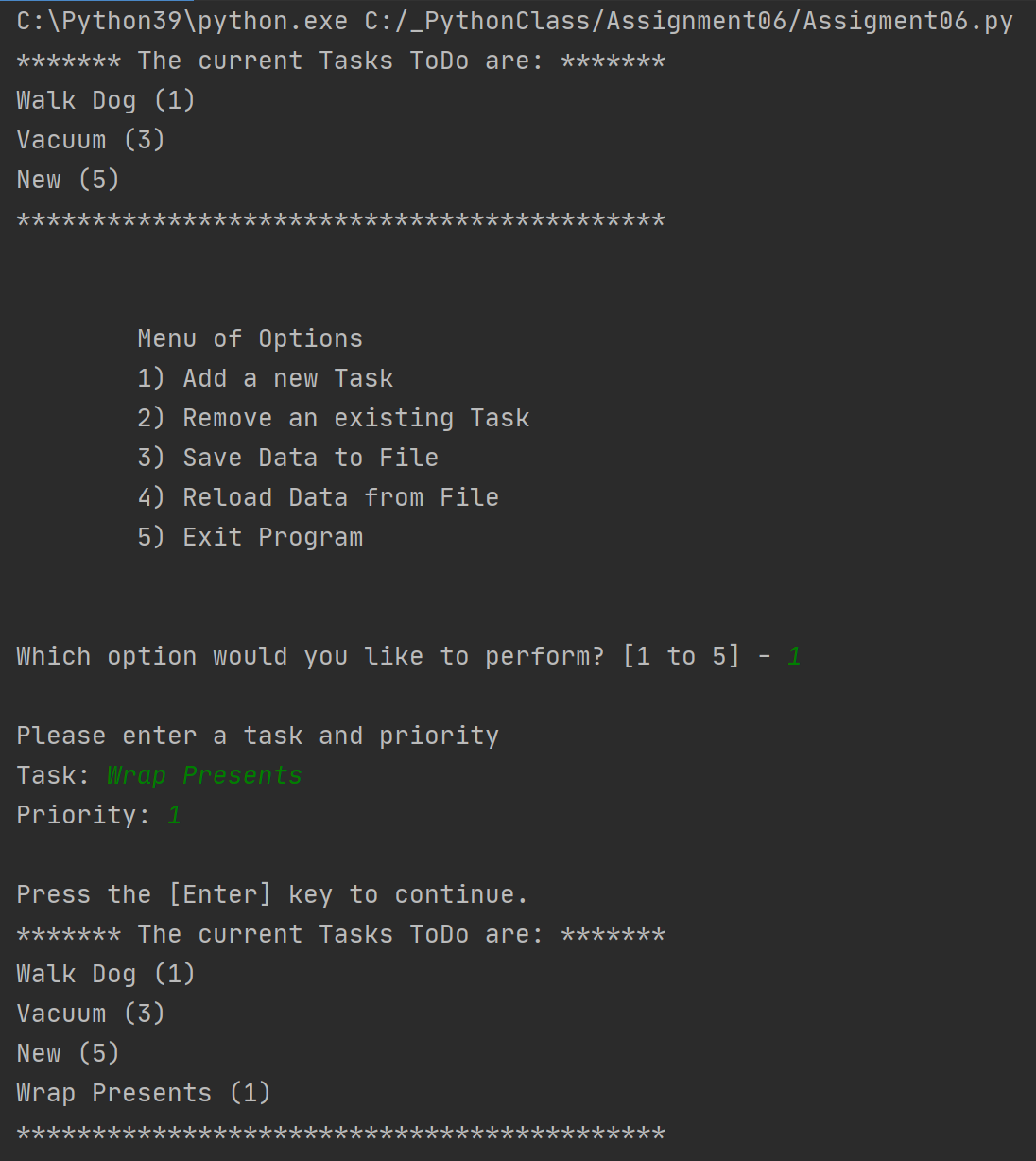
  
Figure 10: Reloading data from the file through calling functions

# Running the To Do List Script

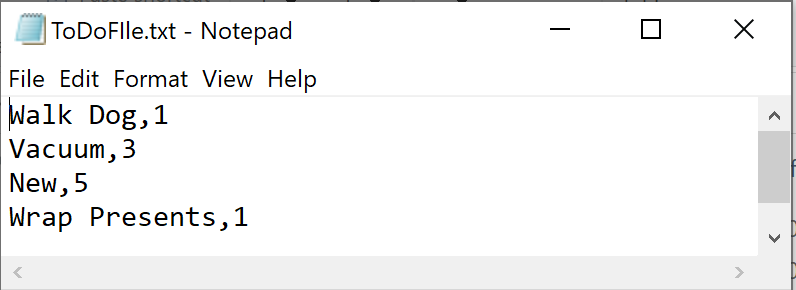
The To Do List script can be ran using PyCharm or from a command window. The following provide examples of successful completions of the script using both methods.

## Output using PyCharm

Figure 10 contains a successful run of the script using PyCharm. In this example the initial data in the ToDoList.txt file was displayed and there were initially three tasks in the list. Then by selecting menu option 1, a new task, wrap presents, was added to the list as shown in Figure 11.

  
Figure 11: A successful run using PyCharm

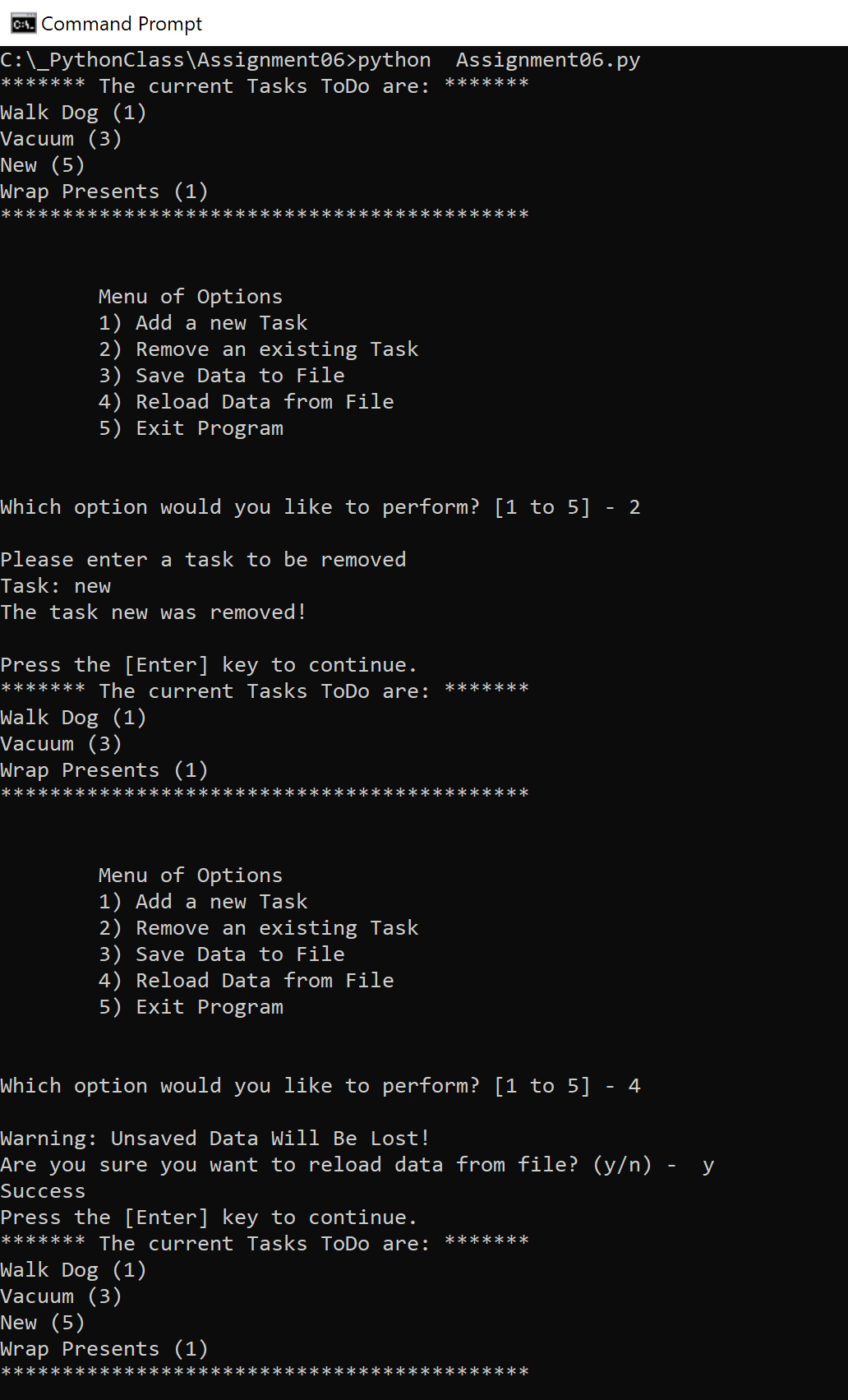
The user then selected option 3 to save the file and then option 5 to exit. After saving and exiting the ToDoList.txt file contained the four tasks as shown in Figure 12.

  
Figure 12: ToDoList.txt file after the PyCharm run

Assignment Asignment

## **Output using a command prompt**

The following output as displayed in Figure 13 is an example of a successful run using a command prompt to launch the script. Upon initial load, the user selects menu option 2 to remove an item from the list. The user enters New which was found in the list and removed. The next menu item selected is to reload data from file, the user confirms with a y that they want the data reloaded. Upon the next printing of the current items the user can see that their unsaved changes have been cleared and the list matches the data currently in the text file from Figure 12.

  
Figure 13: A successful run in a command prompt

# Summary

When writing scripts the best practice is to organize the code with a data section, processing section, and presentation section. Functions are useful to help maintain this separation. Additionally, functions allow the script to reuse code so that tasks that are repeated can be called through a function instead of having to repeat the same function code. In this assignment the functions for the processes and presentation were initially completed, then the main body of the code ran utilizing these functions to run the tasks required to create the To Do List program.