Linda Shmait

8/16/2022

Foundations Of Programming: Python

Assignment 06  
https://github.com/lzs425/IntroToProg-Python

# To Do File

## Introduction

This article will build on the script functionality in Assignment 05, essentially centering on building a to do list with several action items the user can take. This script builds on last week’s general functionality, which involves displaying a menu, taking in user input, executing a process based on that selection, and returning some information or notification.

The difference this week is that instead of writing functions under a while loop, the script is organized in three main sections, the Processing section with the Processor class with methods operating on the data, the Presentation section with the I/O class that deals with the user input/output needs, and then the Main section that calls on those functions and methods with the relevant parameters to execute the required steps.

The code will appear like Figure 1 below when run, though there may be differences based on user selection.

Text

Description automatically generated

***Figure 1. Portion of Completed Script Display in Command Prompt***

## Script

The script consists of Processing data, Presentation, and Main sections as mentioned above. Much of the code and structure was already provided, so the primary task in Assignment 06 was to complete the methods within the Processing and Presentation sections. This organization separates the script for completing certain actions from the section that executes them (Main) in order to lend the code structure and clear to anyone reviewing the code. Once the methods within the Processor and I/O classes were completed and key outputs determined in the return portion, the Main section called them to perform with the relevant arguments input within the parameters.

### Processing Section

The processing section consists of the class Processor, within which several methods are housed. This section is purely to perform actions on the data, like back office computational work rather than interaction with the user. The methods in Processor are listed below in Figure 2.

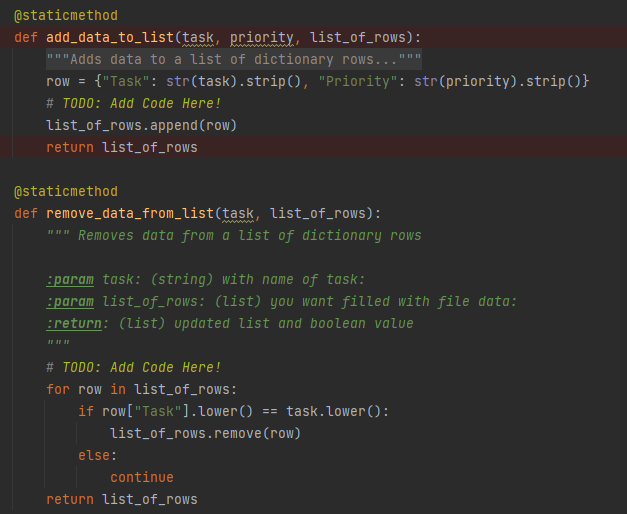


***Figure 2. Class Processor Overview***

The first method, read\_data\_from\_file was fully coded already and designed to pull in data from a file, set as a parameter, and output the data into a table of dictionaries in a table set as the parameter for list\_of\_rows.

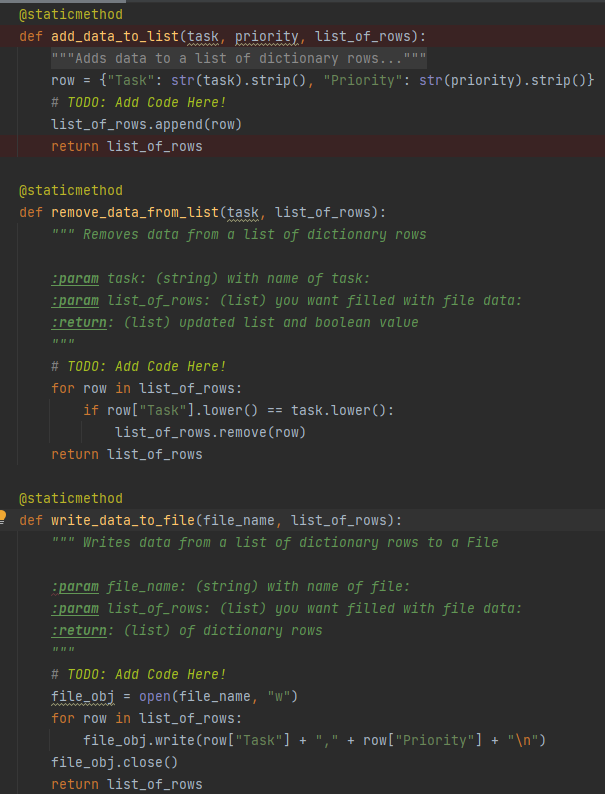
There was code that needed to be added to the other three methods: add\_data\_to\_list, remove\_data\_from\_list, and write\_data\_to\_file. Figure 3 below shows the code for add\_data\_to\_list and \_remove\_data\_from\_list. In add\_data\_to\_list, the parameters to be input to run the method are task, priority, and list\_of\_rows. List\_of\_rows just represents the table we want to add the data. We first put the task and priority into a dictionary row, and then append it using list\_of\_rows.append(row).

The next method I needed to fill in was under remove\_data\_from\_list, which takes in a task and table as parameters, and then loops through the data to remove any row in which the task input by the user and then used as a parameter when it matches the tasks in the table input as a parameter.



***Figure 3. Class Processor Methods to Add and Remove Data***

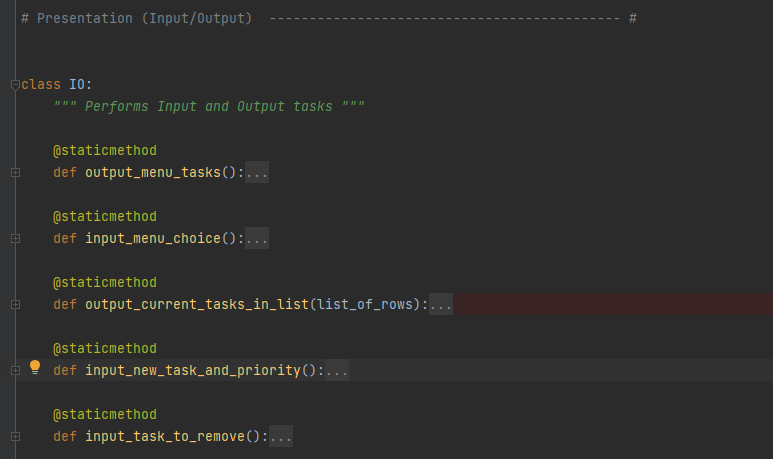
The last method that required some additional code was write\_data\_to\_file with file\_name and list\_of\_rows as parameters. The code, seen in Figure 4, will open the file input as the parameter file\_name, and loop through the rows using a for loop and write the Task and Priority entries of each dictionary row. Once this method is called with appropriate parameters input, it will output the table data into the file set as parameter file\_name.



***Figure 4. Class Processor Methods to Write Data to a File***

### Presentation

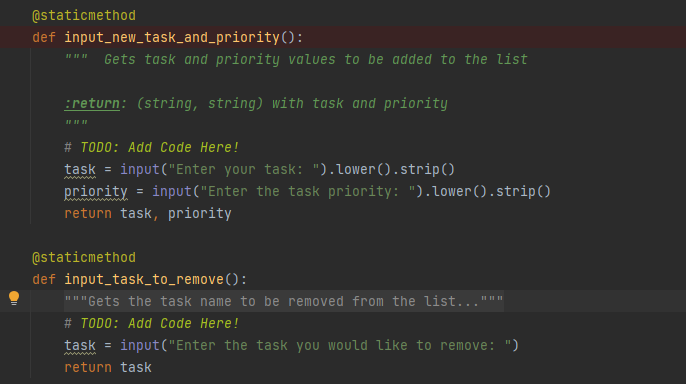
The presentation section is where class IO (input/output) is housed with the corresponding methods. This is where the methods that either require input from the user, or display information to the user, are located. The methods include output\_menu\_tasks, input\_menu\_choice, input\_new\_task\_and\_priority, and input\_task\_to\_remove, as seen in Figure 5. Again, much of these methods were completely or partially complete prior to my input.



***Figure 5. Class IO and Method Overview***

Output\_menu\_tasks, input\_menu\_choice, and output\_current\_tasks\_in\_list were all pre-coded. They use the print and input functions, and in the case of input\_menu\_choice, the user input response to the menu is stored in variable choice, which is what is returned when the method is called. The other two methods do not have a return as they are called only to display data for the user. Output\_menu\_tasks will print the menu options when called. Input\_menu\_choice will prompt the user to enter a menu item number and store it in variable choice. Method output\_current\_tasks\_in\_list will print the current task and priority list when called.

The last two methods under class IO are input\_new\_task\_and\_priority and input\_task\_to\_remove, both of which required some contribution on my part. For input\_new\_task\_and\_priority, I used the input function and stored the responses in the task and priority variables, which are then returned when the method is called. For input\_task\_to\_remove, I just had to store the input from the user for when they want to remove a task, and then have that variable task as a return value. These portions of class IO are in Figure 6 below.



***Figure 6. Class IO Input Task and Priority and Remove Task Methods***

### Main Section

The Main portion of code is where the classes and methods are actually called upon to perform actions and arguments sent to the functions. This section of code was complete prior to my coding anything, and is structured similarly to Assignment 05. It is based on a while loop structure, with a menu display that the user can view to determine and input their next step. The if statement is based on that user entry and will direct the script accordingly. Within the if/elif/else section, the methods are called with appropriate arguments set for each of the parameters.

Prior to any menu choice though, there are methods called to perform specific tasks immediately. We can see two methods from the IO class called in immediately to show the current tasks and to print out the menu for the user to view. That means, upon initiation and completion of entries 1-3, because the while loop is true until the break in option 4, the user will see that list of current tasks that is stored in table\_list- the argument input, and the menu. With these methods, the user will also always be prompted to put in a menu choice, again using the IO method input\_menu\_choice as it is a user-facing method. Based on that entry, stored in choice\_str, the script will take the appropriate steps. This portion of script is below in Figure 7.

Text

Description automatically generated

***Figure 7. Initial Methods in While Loop***

As with Assignment 05, the script will then route to the appropriate code based on the user input into choice\_str. The difference is instead of housing functions within the if statements, the class methods are called in with the arguments set as the appropriate parameters. The first option is to add a new task and priority, so the IO method input\_new\_task\_and\_priority is called in, which returns task and priority. Those returns are then used as arguments in Processor.add\_data\_to\_list to store the data in table\_lst. This can be seen in the first portion of Figure 8 below.

The second section, where choice\_str == ‘2’ will remove an existing task by calling an IO method to get the user input for the task to remove, and then call the processor method to input that task into the remove\_data\_from\_list method that scrubs the task from the list.

The final two sections, based on the user inputs of ‘3’ or ‘4’, do not require user input beyond the menu choice, so no IO functions are called. The selection ‘3’ portion calls on the method of Processor class that writes the data to the external file, with the appropriate variables put in as arguments. Selection ‘4’ uses a break to exit the loop and the program.

Text

Description automatically generated

***Figure 8. If Statements within Main Section***

### Testing

I tested the script in both the Command Prompt and PyCharm. I did this iteratively while adding code and as a final double check. Figure 9 shows a more complete Command Prompt test. Similar to my testing in PyCharm, I input a couple items, verified they showed up on the ToDo list of tasks, tried removing one with Option 2, then saved the data with Option 3 and verified my text file had the appropriate entry. Lastly, I input Option 4 to exit the program. In my final run of testing, all functionality behaved as expected.

Text

Description automatically generated

***Figure 9. Command Prompt Testing***

## Summary

This script uses classes and methods to organize action steps more cleanly, and streamlines the main section in which those actions need to be performed. The script successfully calls both class Processor and IO methods to perform actions on data and interact with the user respectively. Overall, this script can help a user create, edit, and export a to do list.