Steven Huang

May 24, 2020

IT FDN 100 A Sp 20: Foundations Of Programming: Python

Assignment06

<https://github.com/shpy086/IntroToProg-Python-Mod06>

Assignment Recap

Introduction

In this assignment, we are asked to modify a python script that manages a To-Do list. The existing script loads data from a file into a Python list of dictionary objects. The code in the script only uses a few functions so we are asked to add more functions to organize the code, perform the operations listed in the menu to manipulate the data, and write the data to a text file.

Review the Starter Script

To start, I read through the existing script several times to familiarize myself with the layout. Similar to the starter script in our previous assignment, the code was segmented into three layers: a data layer, a processing layer, and a presentation layer. However, this starter script required significant updates to both the processing and presentation layers. Since there were not any similar examples I could reference from this week’s labs, I had to thoroughly review the notes from this week’s lecture and do additional research.

Create and Test

The starter script starts with data loaded from a text file and a menu of operations that we can use to perform different operations on the data in the To-Do list. We were instructed to use each of these operations to manipulate the data. See Figure 1 below:

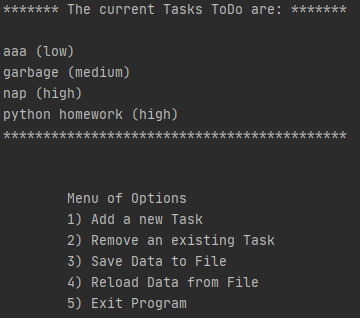


Figure 1

As you can see, it starts by listing out the current To-Do items which were loaded from the text file. This is followed by a menu of options. The first option is “add a new task”, the user needs to input the name of a task and specify its priority level. Since the program specifically asks us to input these two columns of data as part of a dictionary object, we will need to define them as two separate keys: “Task” and “Priority”. The corresponding values for these keys are inputted by the user when a new item is added. See Figure 2 below:

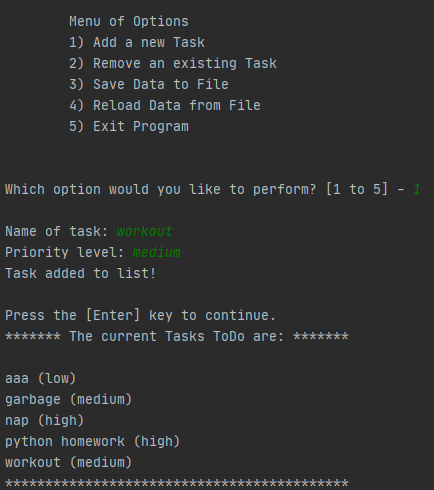


Figure 2

When the user selects option 1, they are prompted to input the name of the task and its priority level. Once both pieces of data have been entered, the program informs the user that the task has been added successfully (saved to memory), and asks the user to hit ‘enter’ to continue. Upon hitting enter, the user sees the updated To-Do list with the new task added. The second option “remove an existing task”, asks the user to remove an item on the list. When the user selects this option, they are prompted to enter the name of the task.

If they enter a task that matches one of the existing items on the To-Do list, the program confirms this with a print statement saying “task removed successfully.” Figure 3 shows this option below:

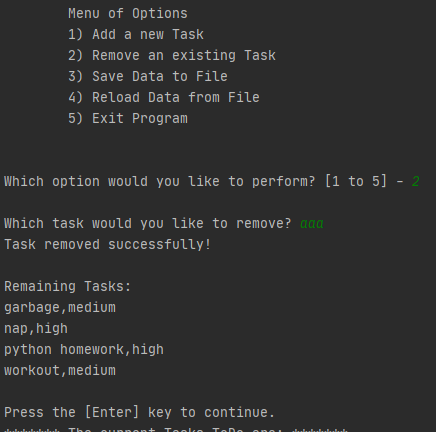


Figure 3

In the event that the user selects option 2 and enters a task that does not match any of the existing items on the To-Do list, the program will respond with a print statement saying “task does not exist.” The third option “save data to file” saves the updated data to the text file. When the user selects option 3, they are asked if they would like to save data to the file and prompted to enter “yes” or “no”. See Figure 4 below:

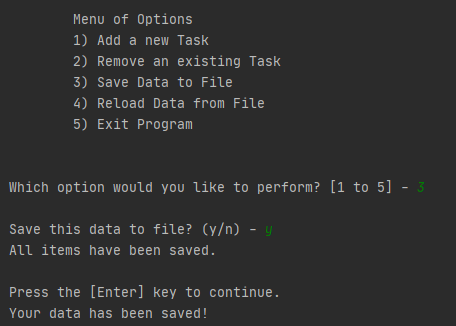


Figure 4

Once the user enters “yes” to confirm the command, the program replies with confirmation saying “all items have been saved.” The fourth option, “reload data from file” asks the user if they would like to reload data from the text file. When the user selects this option, the program replies with a warning stating any unsaved data will be lost. See Figure 5 below:

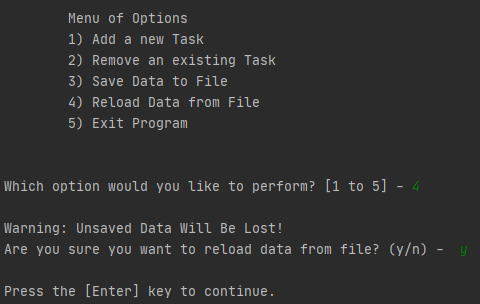


Figure 5

The final option exits the program. When the user selects this option, the program replies saying “Goodbye!” Figure 6 below shows the outcome of the final option:

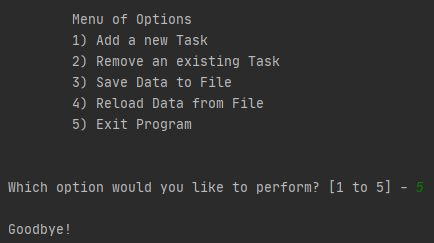


Figure 6

One important thing to take away from this program is that it performs these various operations while storing data in memory. When the user selects options 1, 2, and 3, it confirms the user’s selection by displaying the updated data in the To-Do list. In addition to saving this data to memory, it also saves the data externally by writing it to a text file. Figure 7 below is a screenshot of the To-Do list text file:

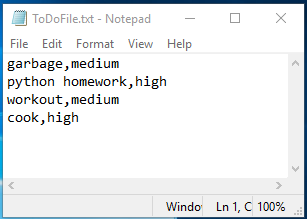


Figure 7

Processing Code

Now that we’re reviewed each of the menu operations, let’s take a deep dive into the code written in the processing layer. In the starter script, we’re provided with a data layer containing all the global variables and a processing layer containing a class called “Processor”. The first function in Processor, “read\_data\_from\_file”, is already defined for us. We’re asked to create three different functions: one that adds a row of data to the list, one that removes a row of data from the list, and one that writes data to an external text file. Figure 8 shows the first of these functions that adds data to the To-Do list:

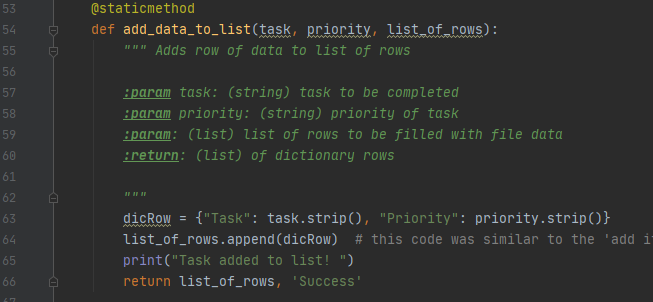


Figure 8

This function uses three main parameters (task, priority, and list of rows). “Task” and “Priority” are the two components of the dictionary object that constitute a row of data, “List of Rows” is the list of To-Do items that we will be adding to. In this function, we declare the dictionary row variable and use the append method to add this row to the existing list of rows and return the updated list. The next function removes data from the To-Do list. See Figure 9 below:

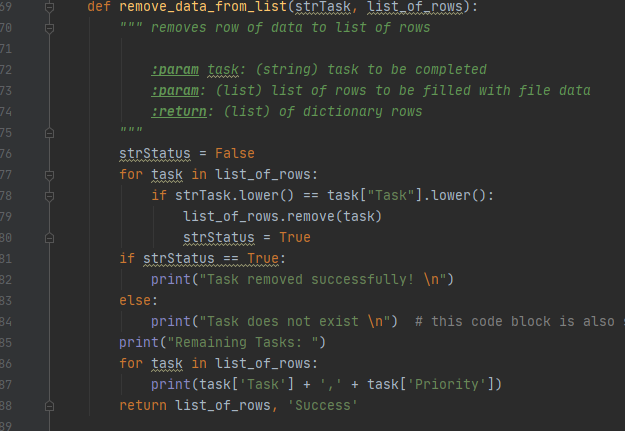


Figure 9

This code was very challenging to write, it contains two parameters: “strTask” and “List of Rows”. The strTask is the string of data entered by the user when they are prompted to enter the task they want to remove from the list. We start by declaring the “strStatus” variable and setting it equal to false, we then create a conditional statement that says if the user string data matches a row of data in the list of rows, remove that list, and set strStatus equal to “true”. If strStatus is true, the program confirms with a print statement saying “task removed successfully!” If the user string data does not match any of the rows in the list, the program replies saying “task does not exist.”

The last function in Processor writes data to an external text file. Figure 10 below shows the code for this function:

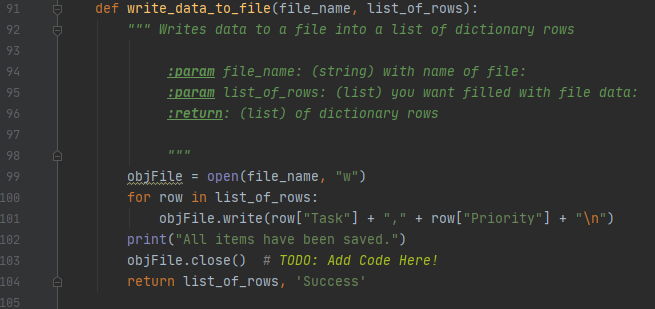


Figure 10

Here, we use the file name and list of rows as the two parameters. We start by declaring the objFile, opening the file, and using a write command to write the data to the text file. Then, we use a conditional statement that says for any row in the list of rows, write a row of data to the text file, and close that text file.

Presentation Code

Now that we covered the code written for the processing layer, let’s discuss the code written for the presentation layer. The presentation layer required some heavy lifting, it starts with a class of functions called “IO”. The first half of the IO class is provided for us in the starter script but we’re asked to add additional code to two of the functions: “input new task and priority” and “input task to remove”. Figure 11 shows these two functions:

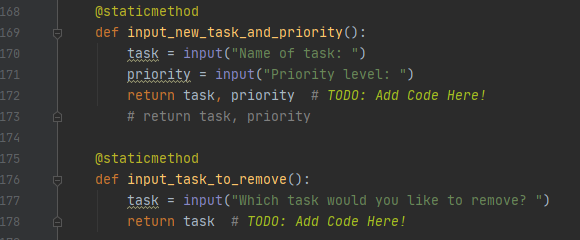


Figure 11

Both of these functions are pretty straightforward. The “input new task and priority” function requires us to declare two local variables (task and priority), both of these variables will contain data inputted by the user and return the values for both. After creating these two functions, we have finalized the IO class and can move on to the main body of the script. This last code block contains a “while” loop that cycles through each of the user options. For each of the conditional statements in this loop, we used a combination of functions from the processor class and IO class. Figure 12 below shows this entire code block:

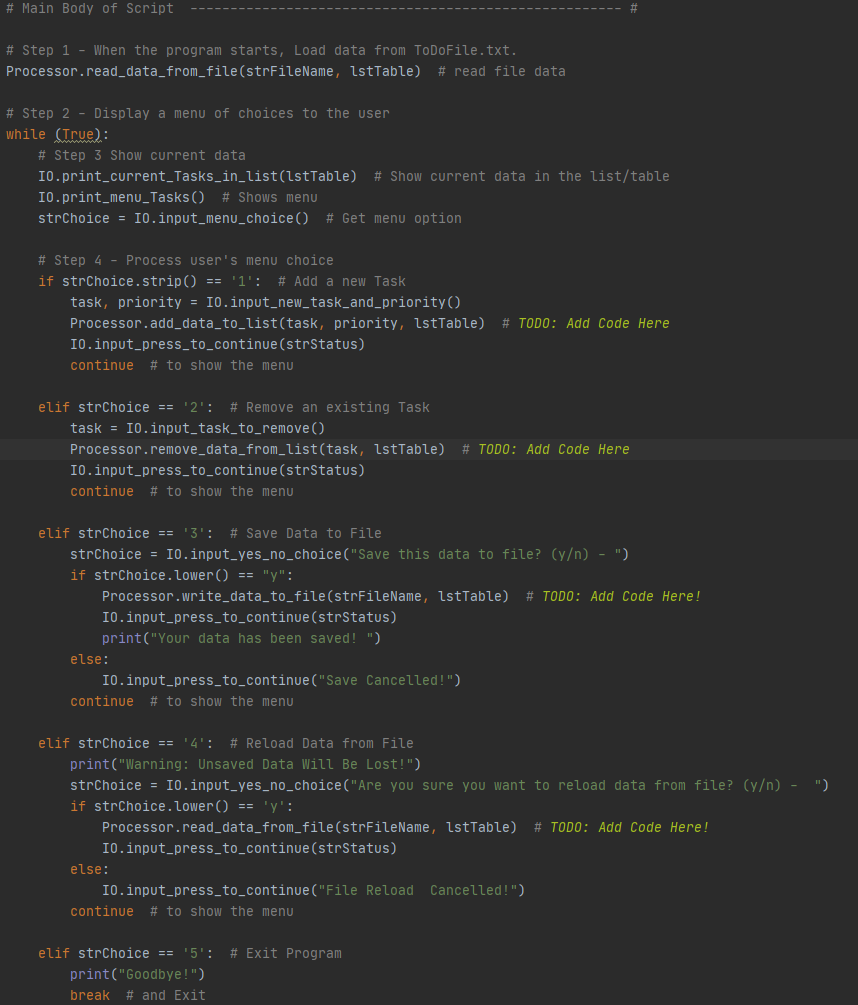


Figure 12

Run the Script

We have reviewed each option of the program menu and evaluated the code that was required to accomplish each step. While reviewing each step, we verified that the complete script ran successfully in PyCharm. Figure 13 below shows the program running successfully as a console application in the command shell:

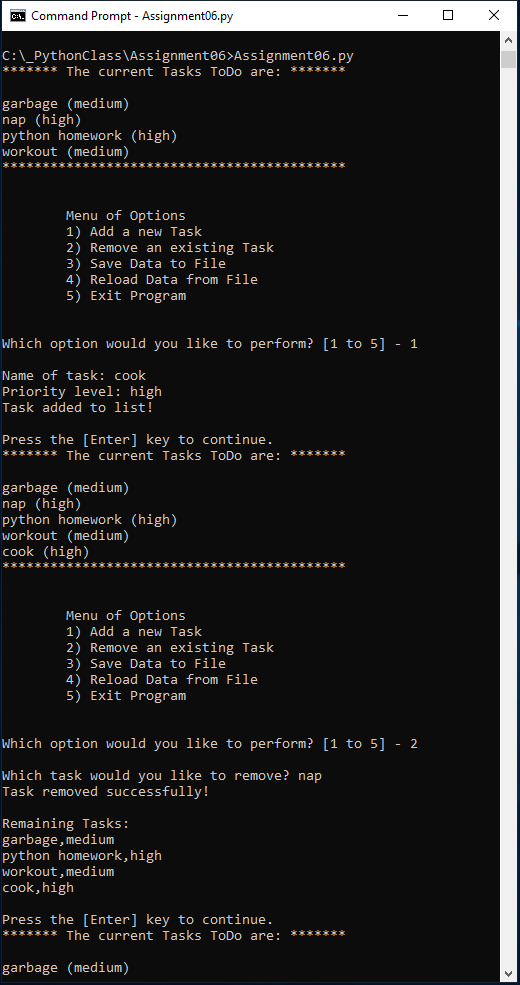


Figure 13

Conclusion

This assignment required a number of challenges. While reviewing the starter script, we had to pay very close attention to the formatting of the code at each layer. After familiarizing ourselves with the format of the processing layer and the presentation layer, we had to create custom functions for the processor class and the IO class. Then, we had to combine both classes into a while loop in the main body of the script. Using a combination of both of these classes, we were able to call the functions and successfully modify the program.