Naomi Martin

Assignment 05

Aug 10, 2022

**The To-Do List Program**

**Introduction**

This assignment was intended to familiarize ourselves with creating data in lists and dictionaries, and converting between the two data types by creating a program that keeps track of a user’s to-do list. It was also intended to give us more practice in using while loops and if statements through the use of a menu of options, as well as for loops through the requirement of iterating through multiple rows of a list. I was able to apply my knowledge from the previous modules to edit the starter code given to us, effectively completing the program to my satisfaction. In this paper, I will review my code and discuss mishaps I encountered or concerns that I have.

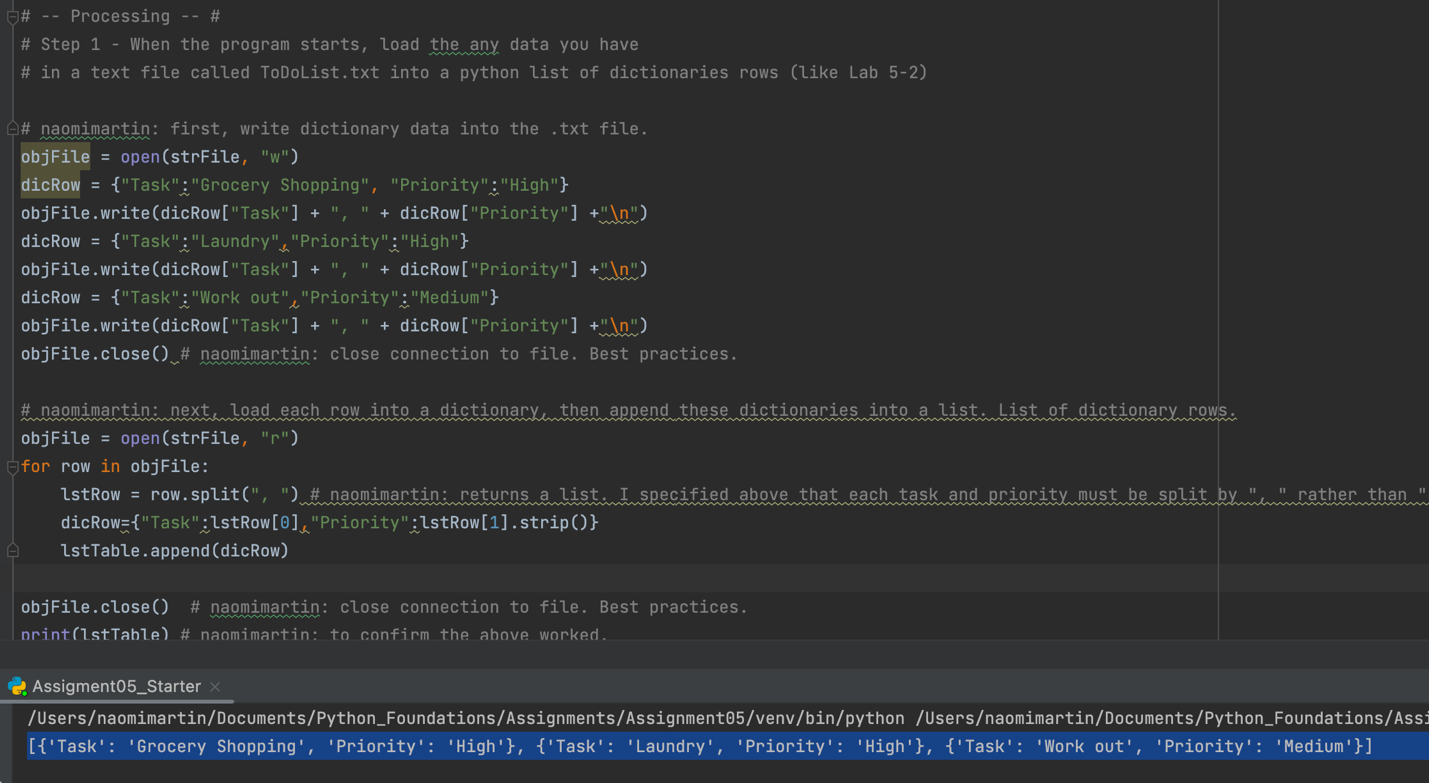
**Step 1**

The first thing I did even before I tackled step 1 of the data processing section was update the change log to reflect that I made changes to the code on this date. Additionally, I kept in mind that any future comments that I make in my code should be specified as my comments, to differentiate the starter code from my code. I do this by specifying my username “naomimartin” before any text that I comment out (figure 1).



**Figure 1.** An example of code that I commented out, specifying my name.

I then proceeded to step 1. In step 1, we are instructed to load in any data that we have. I decided to load in the items that are on my to-do list for today. I did this with the following code (figure 2).

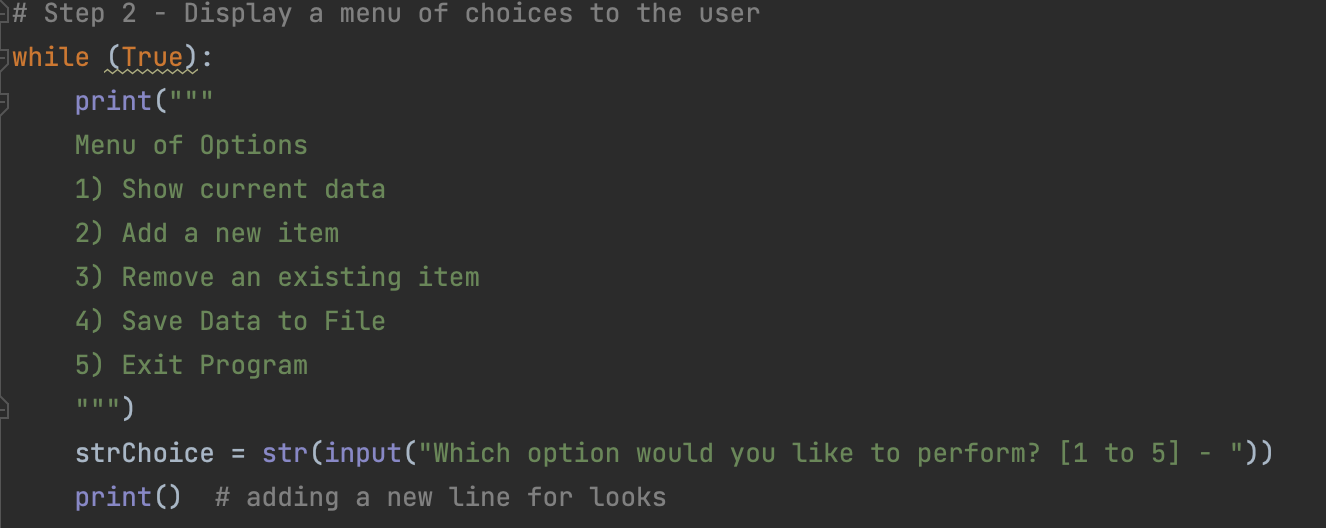


**Figure 2.** My code data processing step 1, and expected output for lstTable.

I started by assigning dictionaries that contained my data to the pre-defined dicRow variable, and wrote it to the file by opening the file with the “w” argument. This simply wrote the data to the file, but next I wanted to load the data into the program. I did this by opening the file with the “r” argument. First, I created a new variable lstRow which split the elements separated by “,” in each row of data (this variable is defined at the top; more on this in the “Step 4” section). For one row, this returned a list containing the task and priority. For all rows, this returns a list of lists, each containing a task and priority. I then extracted each row’s task and priority using an index, and assigned it to the dicRow variable. I then appended dicRow to lstTable, to store the data from the .txt file. I printed the updated lstTable that I loaded my data into to ensure that they were properly loaded, and to view the different data types that were stored. I saw what I expected: a list of dictionaries.

**Step 2**

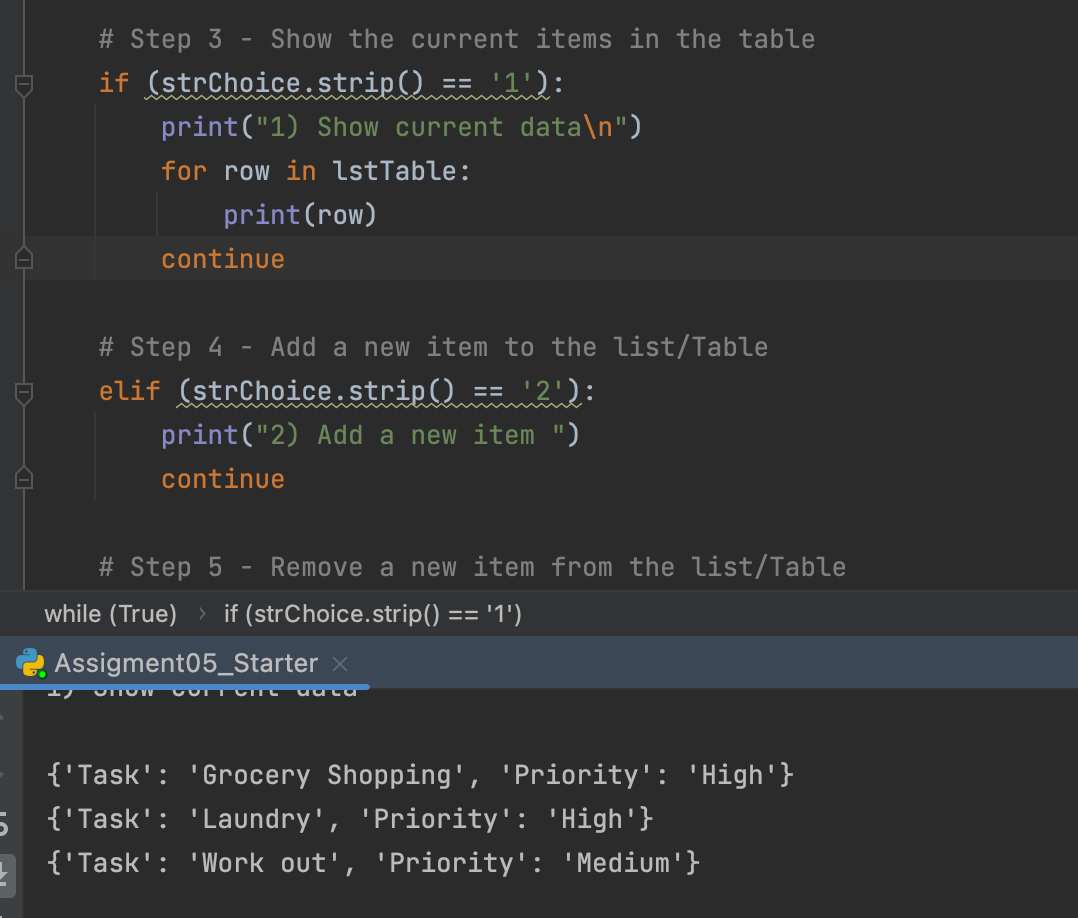
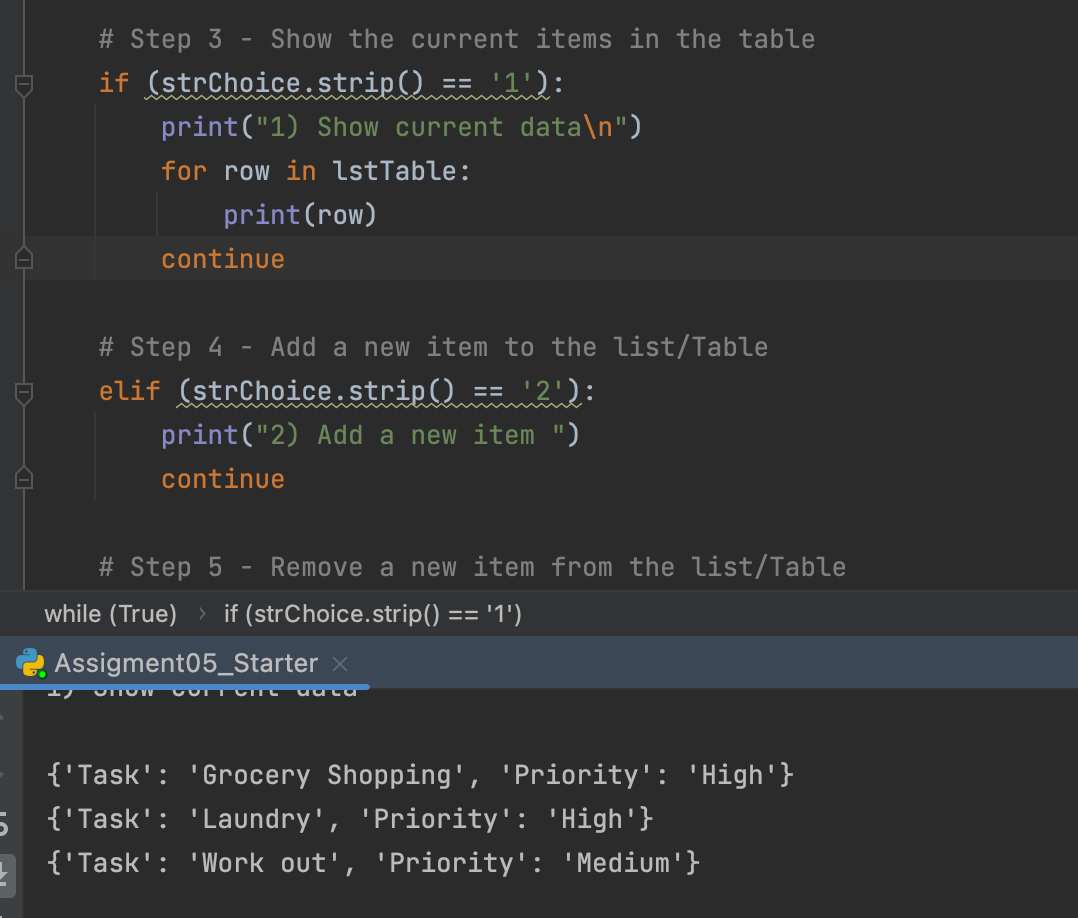
Step 2 of the processing data section had already been written by Professor Root, so I did not have to add any additional code here. Basically, step 2 prints the menu of options, and allows the user to input the menu option they would like to select (figure 3).



**Figure 3.** Professor Root’s code for Step 2

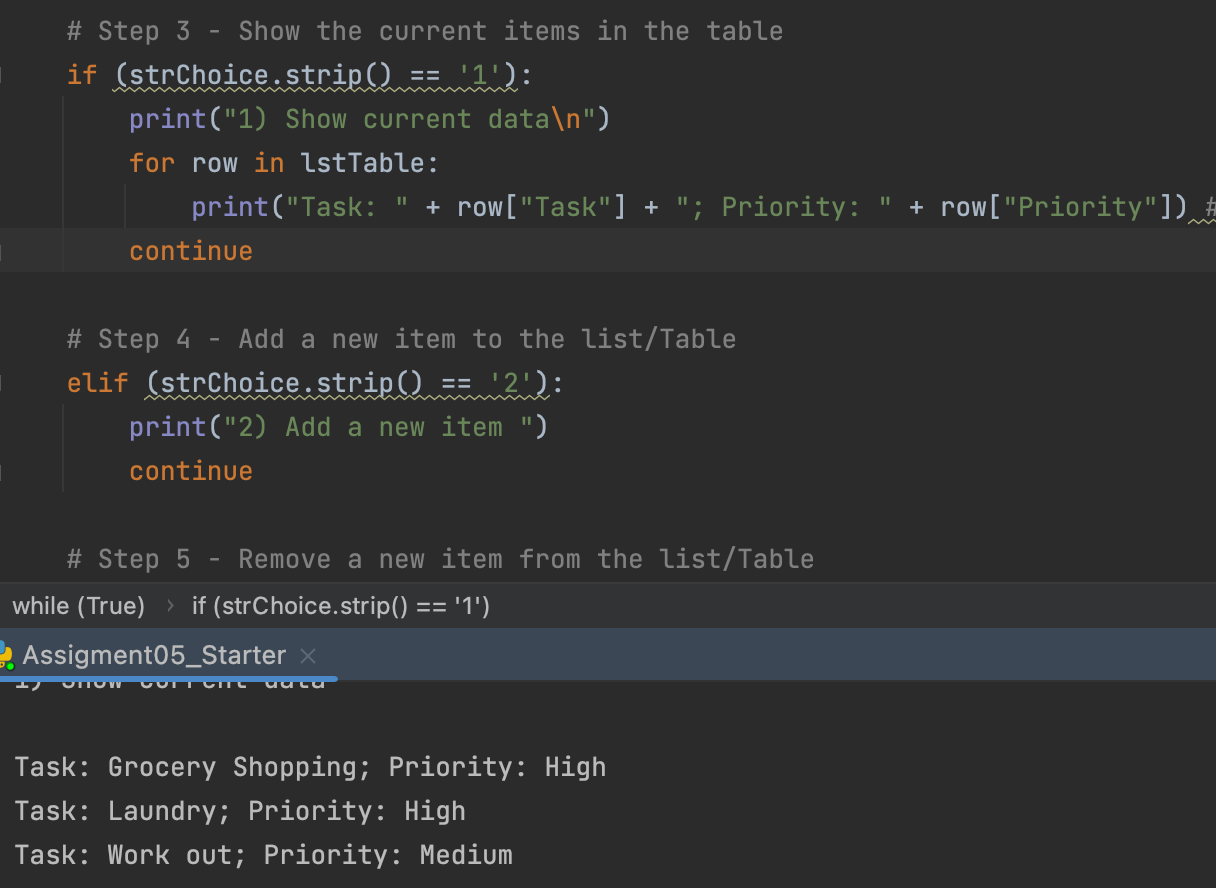
**Step 3**

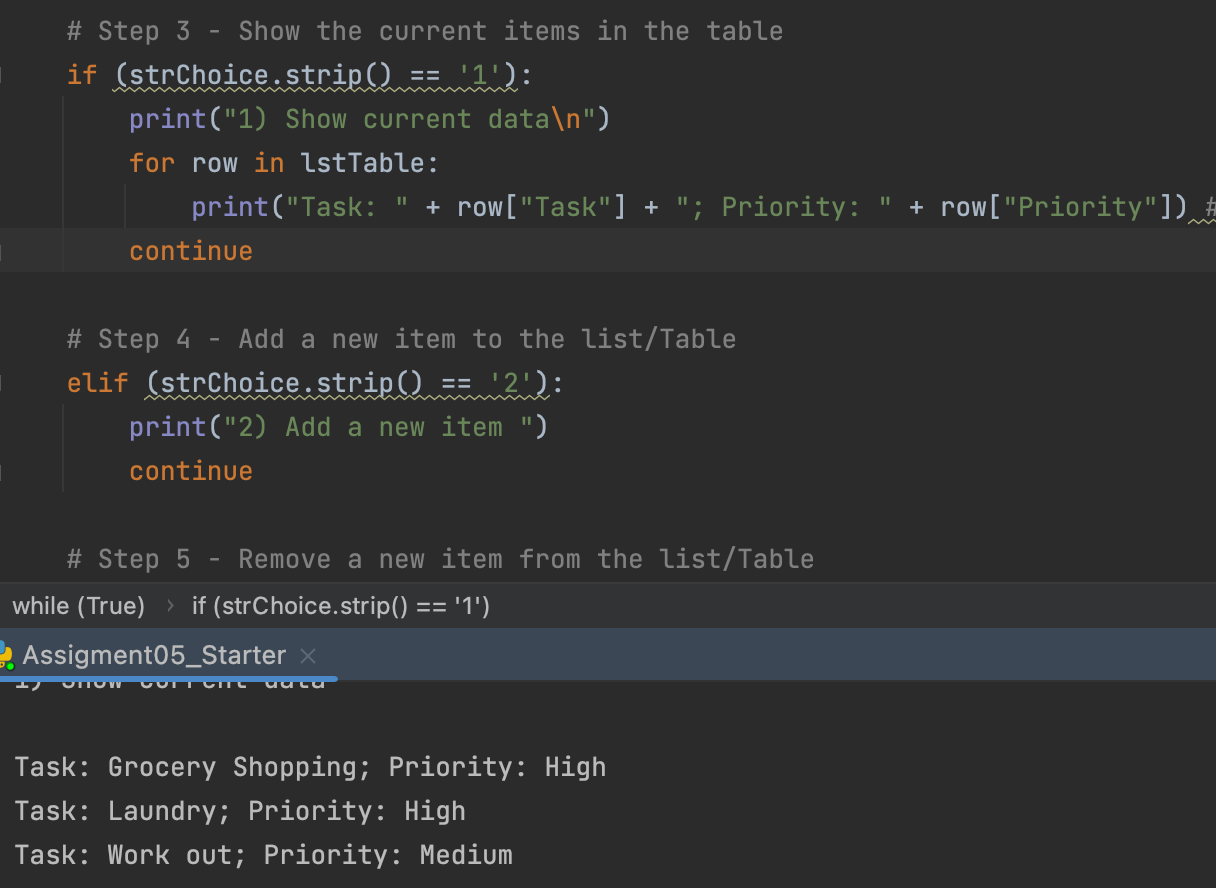
Step 3 was simple enough, as I just wanted to print the data to show it to the user. I initially tried to just print each row of lstTable, but I did not like how the information was presented to the user (figure 4).

****

**Figure 4.** My code which printed each row in lstTable, and the output of this code, the look of which I did not like.

Therefore, I decided to change my code up a little bit to improve presentation to the user. I instead printed each task in the dictionary after the string “task: ”, and each priority level in the dictionary after the string “priority: ”. Here, I do not specify a numerical index to get an element in the dictionary, I specify a key (figure 5).



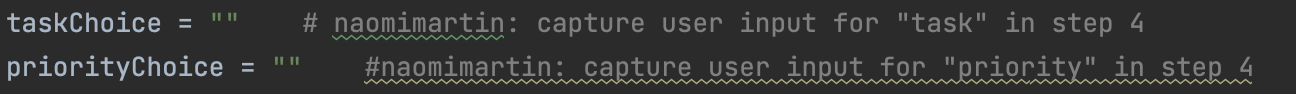


**Figure 5.** Updated code that streamlines the presentation of the to-do list.

Once I was happy with the presentation of the data to the user, I proceeded to the next section.

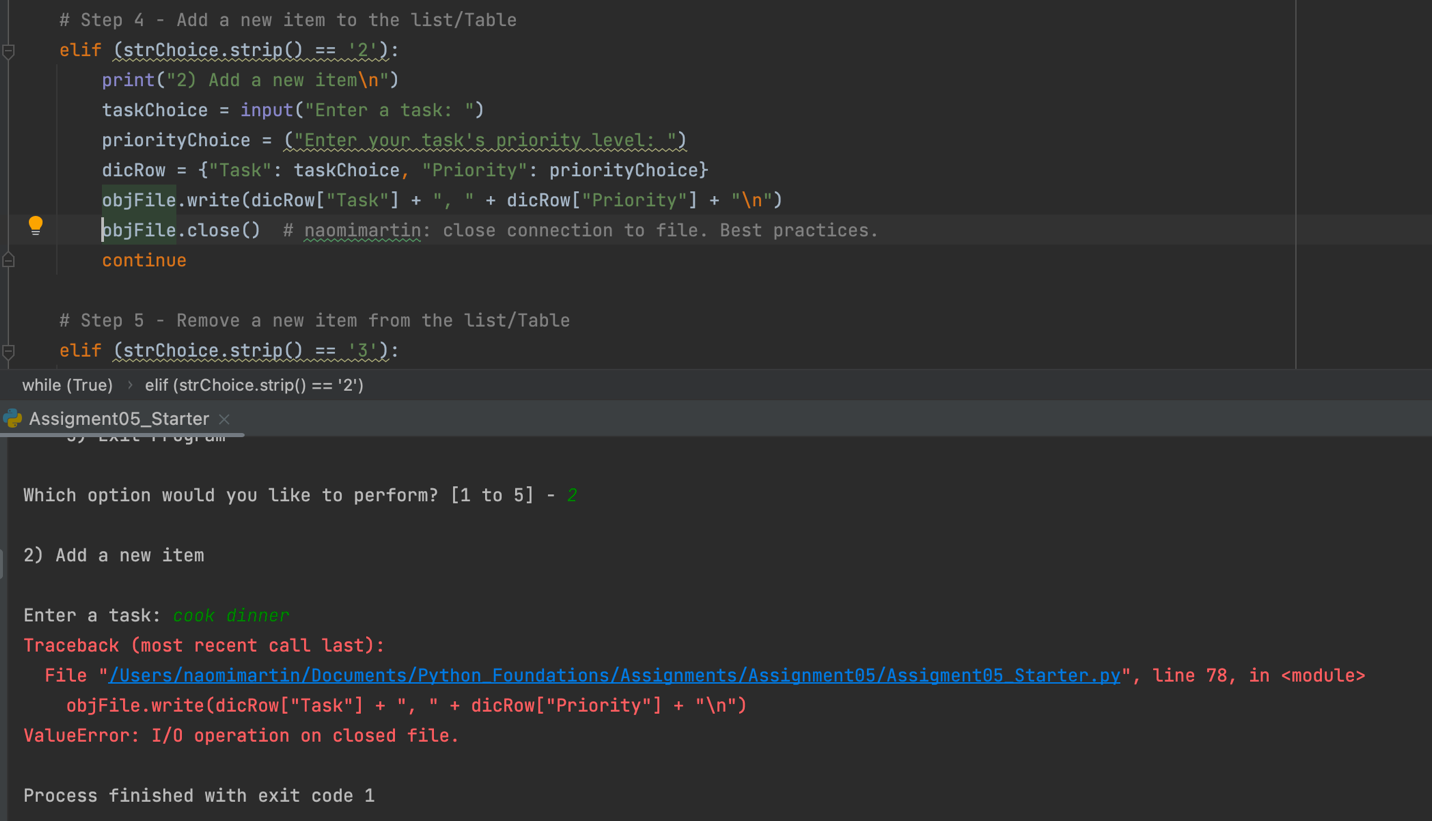
**Step 4**

In step 4, I realized that I am asking the user for input, so for programming best practices, I made sure to define my variables at the beginning of my program’s code prior to writing the code for this step. I do the same thing in step 2 with the variable lstRow. Here in step 4, I introduce two new variables: taskChoice and priorityChoice to capture the user’s input for adding a new task and assigning its priority level (figure 6).



**Figure 6.** The new variables defined at the beginning of the program’s code.

Once these variables were defined, I proceeded to ask the user for input. Here is a first run-through of my code for Step 4 (figure 7).

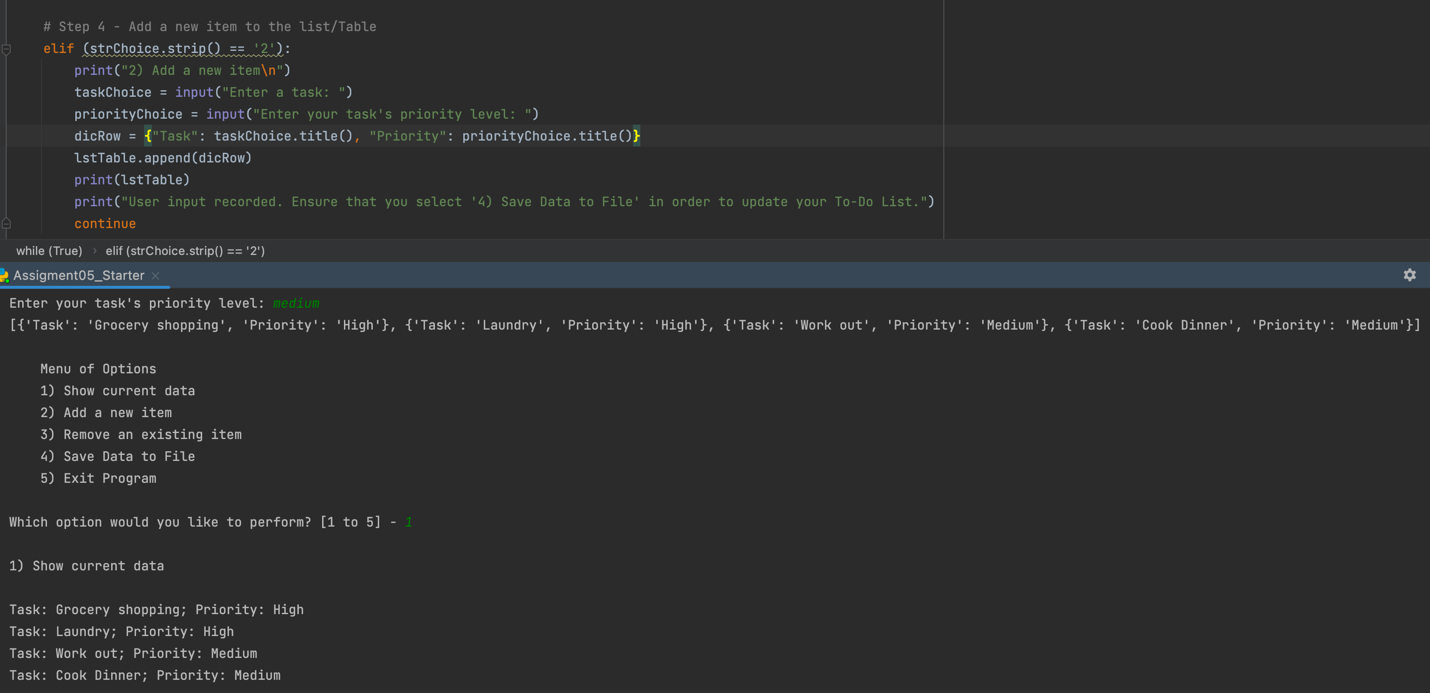


**Figure 7.** The first run-through of my code produced an error.

There were two things I had noticed after this initial run. The first, less obvious thing that I noticed was that I forgot to use the input() operator in the priorityChoice variable. This simply assigned the text specified in between the quotations to the variable, rather than asking for user input. I rectified this by writing input before the string.

The second, more obvious thing I noticed was the error message that said an operation was performed on a closed file. For best programming practices, it is always a good idea to close the connection to an open file. I had done that previously, and reviewing my code, I realized that I had forgotten to re-establish the connection!

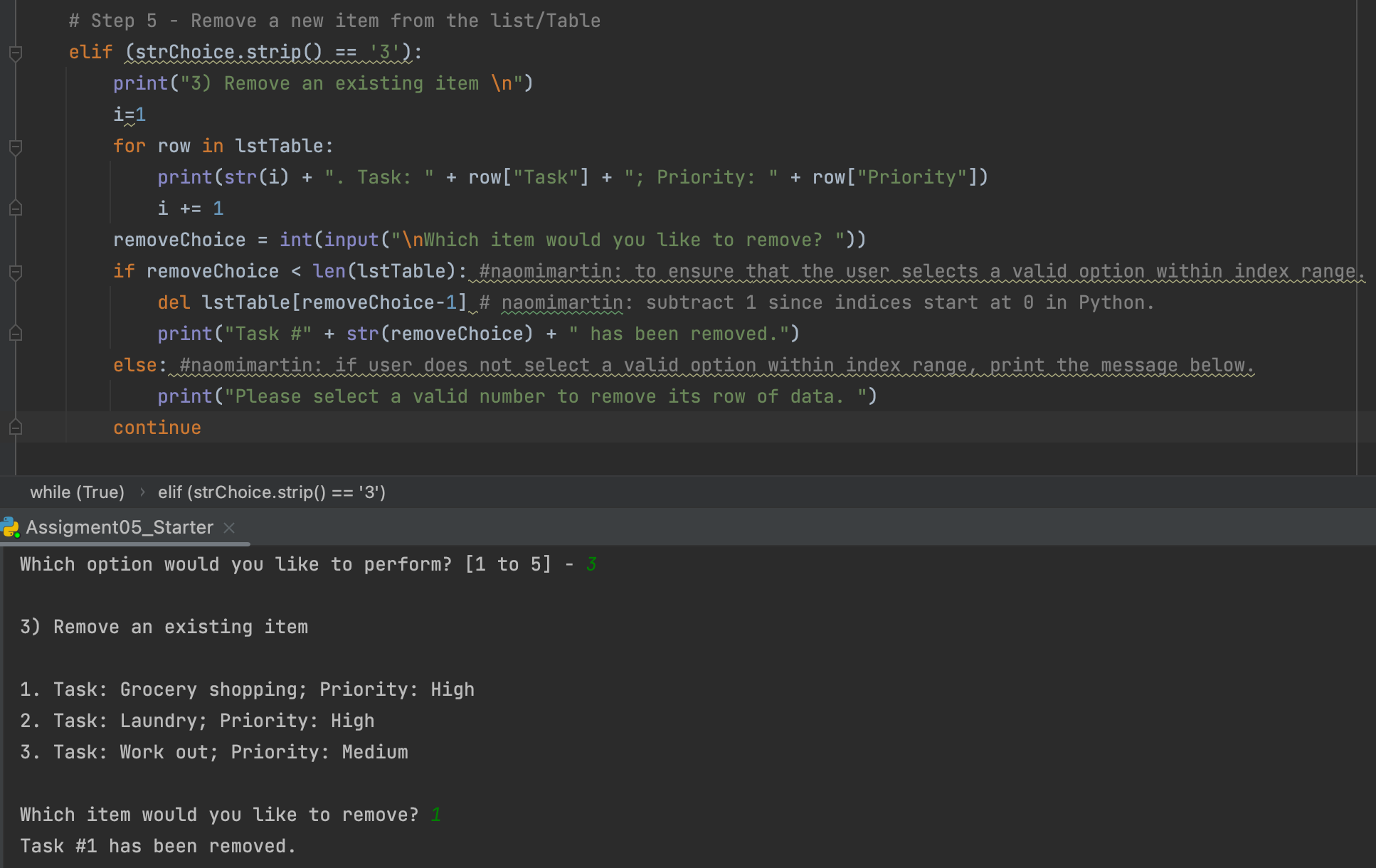
However, as I thought about this more, I remembered that the menu of options contains an option to save user input data to the .txt file. Therefore, step 4, or menu option #2, was intended to gather user input and store it in the program, rather than save it to the .txt file. When I realized this, I rectified my code so that the user input was saved back into the dicRow variable, and appended dicRow to lstTable. In order to test the functionality of this section, I ran my program with my input of “cook dinner” for task and “medium” for priority, and selected menu option “1” to show current data. It worked as expected (figure 8).



**Figure 8.** Code for step 4, gathering user input and appending to lstTable.

**Step 5**

I was stuck for a short while determining what was the best way to code for the removal of user-specified data. I decided that creating a number system would be the easiest way, as I could simply index lstTable to remove a row based on the user’s input. Figure 9 shows my code for this section (figure 9).

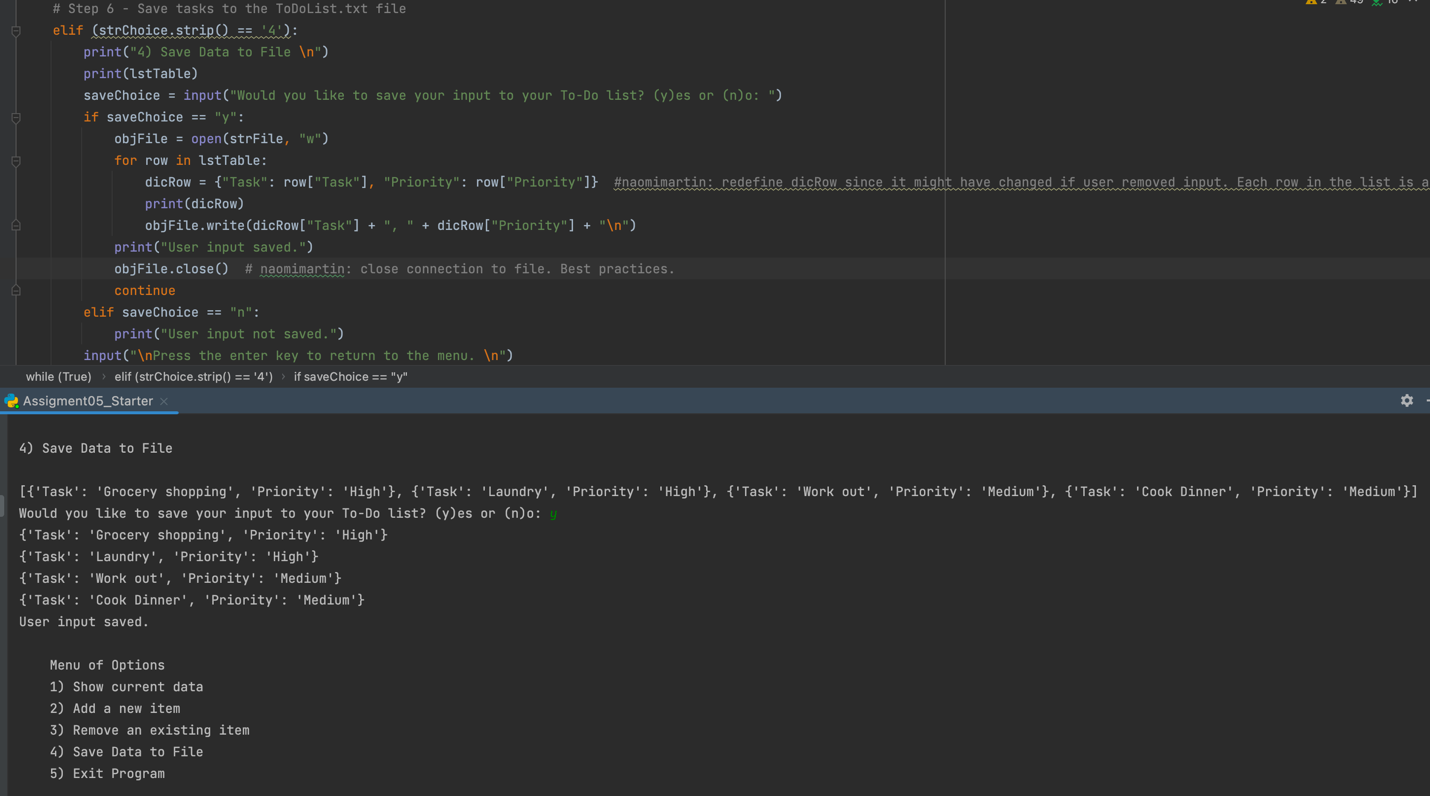
**Figure 9.** Code for step 5 with its expected output.

I first assigned the value of 1 to a random variable *i*. I then wanted to assign each row of lstTable to a number, starting with 1. For each row in lstTable, I converted the value in *i* to a string, and printed that row’s information with respect to task and priority. For each row in lstTable, I also added 1 to the variable *i,* so that each time the *for* loop was run for each row, the variable *i* increased in value, to show an ascending numerical list containing each row of lstTable. This effectively created a numbered list showing all of the data currently saved in the program.

I then wrote an *if* statement, with the condition being that the user-inputted number was smaller in value than the length of lstTable. Therefore, if the user attempts to enter a value that is not displayed in the numbered list, the program will display the message “Please select a valid number to remove its row of data”. If the user enters a valid number, then the program deletes that row of data, by indexing lstTable with the user-inputted number subtracted by 1, as Python indices start at 0. The program them displays a message informing the user that their specified row of data has been deleted.

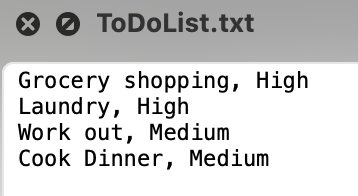
**Step 6**

In step 6, all I wanted to do was write the data contained in lstTable to the .txt file. This was done simply enough, as shown in the code below (figure 10).



**Figure 10.** Code that saves data contained in lstTable to the .txt file.

Here, I wanted to view lstTable by printing it to get a refresher on what it looked like, and the data types it contained. In this way, I could see how exactly I wanted to define my variables to write them to the file. If the user enters “y” for “yes,” the .txt file is opened with the “w” argument specified. For each row in lstTable, where each row is a dictionary, I created a new dictionary containing each row’s task and priority level, and assigned it to the variable dicRow. I specified the dictionary key for each row, as dictionaries require key inputs rather than numerical index inputs. I then wrote this information in a visually clear way to the .txt file, by concatenating each row’s task and priority information, separated by a comma. I also added a carriage return at the end of the string concatenation, so that each row can have its own row in the .txt file. A message is then printed by the program to inform the user that their input was saved. After running through this, I checked the .txt file, and saw what I expected to see (figure 11).

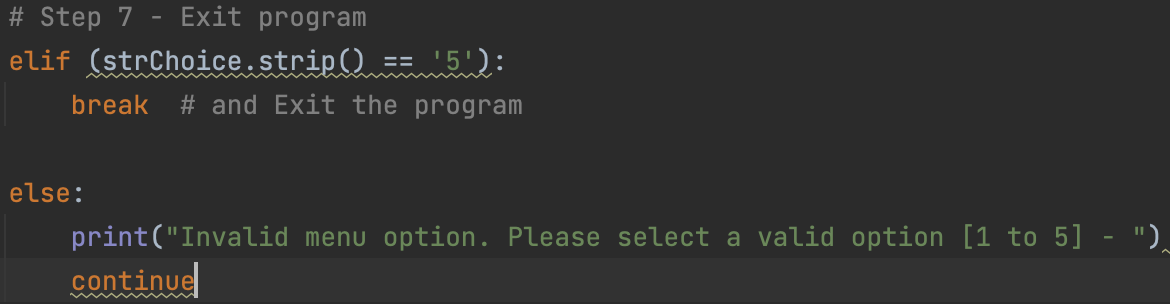


**Figure 11.** The ToDoList.txt file that saved user input from the To-Do List program.

On the other hand, if the user enters “n” for “no,” the program informs the user that their input was not saved, and allows the user to return to the menu.

**Step 7 and Finishing Touches**

The code in step 7 was already written by Professor Root in the assignment 5 starter file, so there was nothing more I needed to add. I did, however, want to specify in the program that if anything other than the specified menu options were inputted by the user, the program would inform them that they inputted an invalid option. I did this with a final else clause. Here, if anything other than the provided menu options are typed in, the while loop jumps back to the top, and the program again asks the user to select a menu option (figure 12).

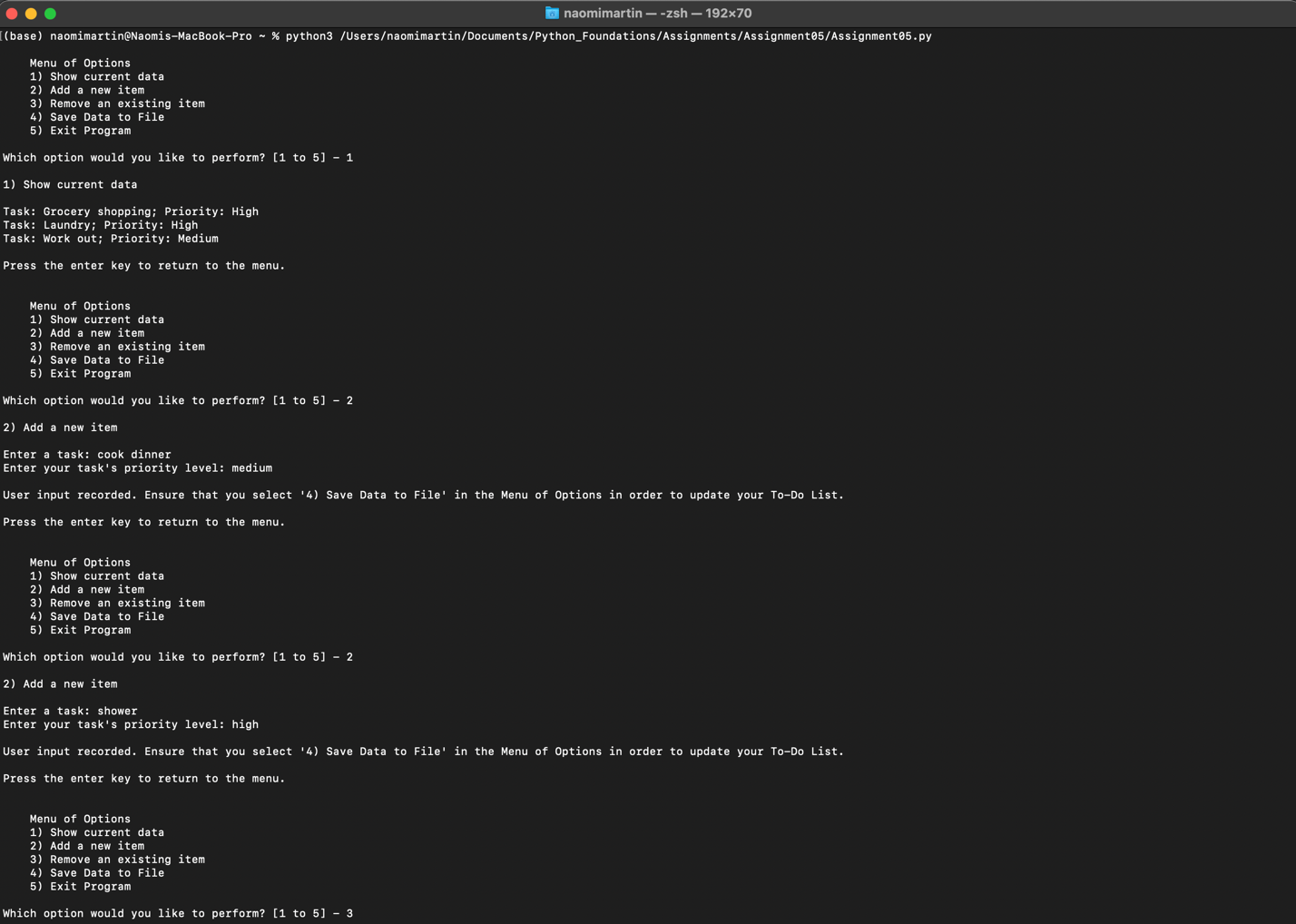


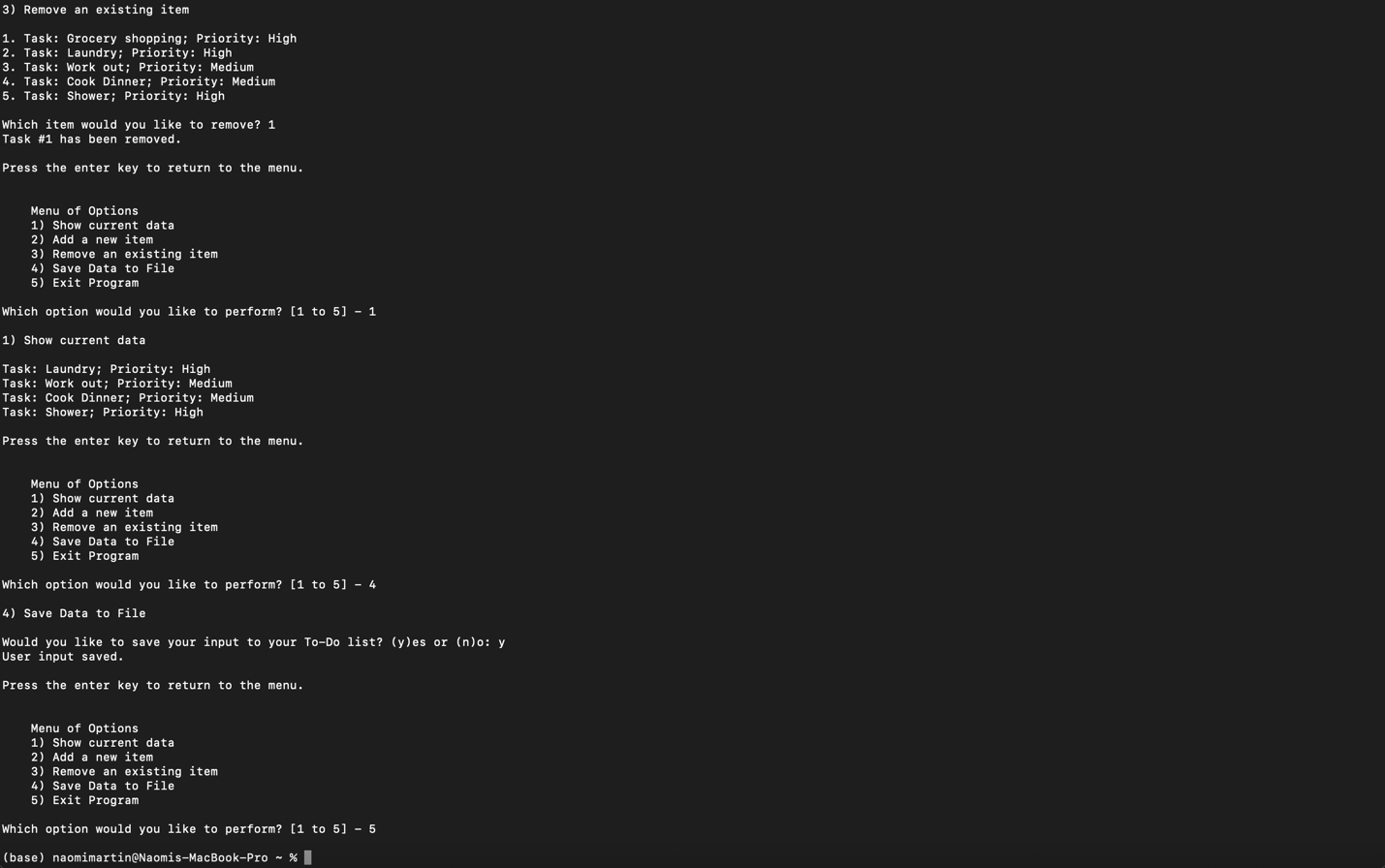
**Figure 12.** The final options that either end the program, or jump back to the top if an invalid menu option is entered.

I ran through my code one last time to identify any errors and/or aesthetic preferences, and went back and added carriage returns to the beginnings and/or the endings of some of my printed character strings. I also did not like how the program automatically went back to the main menu each time without warning, so I went back and added input statements “Press the enter key to return to the main menu” so the user can have more control over the flow of the program.

**Final Test in the MacOS Terminal Command Line**

I performed one last full test run in the MacOS Terminal Command Line, to ensure that everything was working as it should. Figure 13 shows a full test run through each menu option of this program (figure 13).





**Figure 13.** Full run through my final code.

I checked the ToDoList.txt file and found that all my data had successfully been written to the file (figure 14). My program was complete.



**Figure 14.** The expected information in the ToDoList.txt file.

**Conclusion**

This assignment was slightly easier for me than the two previous assignments, as through practice I had become more familiar with writing *for* loops, while loops, and *if* statements. I also think that the interactive labs Professor Root had us do in module 05 was very helpful; I liked the more interactive engaging nature of these labs, which got us to think pre-emptively on a problem we would encounter in this assignment.

Working with someone else’s code was a challenge at first, as Professor Root’s preferred formatting, variable name designations, and syntax were not my preferred. However, it did teach me about new things that I should include in my code in the future. For example, I had not been writing “continue” or “break” after certain *if* statements, which is something I should do in the future to be clearer about the intention of my code, and to avoid potential debugging down the line.

Although this assignment was easier than the previous two assignments, I still encountered some minor syntax problems that were simple yet troublesome to fix. One of such problems involved proper indentations for loops and statements, not only for accidentally mistyped indentations, but also for intentionally mistyped indentations that did not match with the expected block of code. I think this is something I will grapple with for a while, but will get comfortable with in time.

Another problem I struggled with was with my lack of, albeit growing intuition for programming. For the “add an item” and “remove an item” steps, I was attempting to write additional loops and *if* statements to allow the user to add or remove multiple items at once, but I could not figure out a way to do so. I want to continue to modify my code to try to add these features, but at the moment I do not feel completely comfortable with loops, or feel I have the proper intuition for incorporating such features.

On another note related to the intuition for programming, I am curious to see other ways in which the “remove an item” step could be approached. I chose to display a numbered list in which the user specifies a number to be removed, and I am curious to see how other students, and how Professor Root approached this step. I believe my intuition was close to correct for this problem, but could have been written in a more streamlined manner.

Working with someone else’s code

* I wouldn’t have added “continue” after if statements
* Naming convention is different from mine
* Separation of concerns

Problems + concerns

* Indents for loops and statements
* Curious: how else could step 5 be approached?