Molly Weaver

February 21, 2021

Foundations of Programming: Python

Assignment 06

<https://github.com/molweave532/IntroToProg-Python-Mod06>

Return of the ToDo List Program

# Introduction

For assignment 06 we introduced functions. We essentially re-wrote homework 05 but moved all the processing and input/output steps into separate functions. We again started with a fragment of program written by the professor and then filled in the code for each menu option. We took user input to create a to do list. The user could add or delete tasks, save the data to create a running list that could be returned to and edited again and again, or re-load the original list. I also used some tips I learned from reviewing some of my classmates’ programs to improve the readability of my code.

## The Program

The first thing I did was to create my text file and add in some data. Then I started with the main body of the script, menu option 1 – asking the user for a new task to add (**figure 1**).

Graphical user interface, text, application

Description automatically generated

***Figure 1.*** Adding a new task to the list.

I started with a simple pseudocode outline, then filled in with a call to the function “input\_new\_task\_and\_priority”. I saved my expected output of the function in the two variables strTask and strPriority, and then printed those to be sure I was getting the output I expected. It took me a few test runs to realize I needed to add “IO.” in front of the function call. I assume that’s because all the presentation functions are defined in the class “IO”. I then filled out the code for the function (**figure 2**).

Graphical user interface, text, application

Description automatically generated

***Figure 2.*** Asking the user for the new task and priority.

Again, it took a few tries to get the format right. I modeled my docstring to look like the others from the starter code. I used what the professor called “snake casing” to name my functions and local variables – the function names utilize underscores between words and the local variable names don’t include types. Eventually I was able to get the expected user input, pass it back to the main body of the script, and print it to the screen (**figure 3**).

Text

Description automatically generated

***Figure 3.*** Asking the user for a task and priority to add.

Next, I wanted to add the task to the to do list by calling another function “add\_data\_to\_list” (**figure 4**).

Graphical user interface, text, application, email

Description automatically generated

***Figure 4.*** Calling the function to add the new task to the list.

This time the function call had the prefix “Processor” because all the processing functions are defined in the class “Processor”. The I wrote the function, using my code from HW 05 (**figure 5**).

Text, letter

Description automatically generated

***Figure 5.*** Adding a task to the list.

Then I did a test to run to be sure things were working as expected (**figure 6**).

Text

Description automatically generated

***Figure 6.*** Adding a task to the list, or not?

It looked like I was able to add the new task to the list, but somehow, I broke another function. I looked back through the starter code. In the first function that the professor had written for us, he created the dictionary rows in a slightly different way than what I was doing –

Where I wrote – row[task] = priority

The starter code wrote – row = {“Task”: task, “Priority”: priority}

One purpose of this homework assignment was to work with code someone else had started. Instead of changing the starter code to match my method, I changed my code to match the starter code. In my code the key is the task, and the value is the priority. In the starter code, the strings “task” and “priority” are both keys and the actual task and priority are the values. In essence I had each row as a dictionary with one item, but in the starter code each row is a dictionary with two items. That would complicate things! I couldn’t just copy my code from the previous homework. I would have to modify it to match this different approach. When I ran option 1 again with this new format, it worked (**figure 7**).

Graphical user interface, text, application

Description automatically generated

***Figure 7.*** Adding a task to the list – second draft.

I was able to add a new task to the list in the right format and print the new list to the screen.

Next, I wanted to run a check to see if the task was already in the to do list before adding it. I started with my code from HW 05 but adapted it to reflect the new format of my list (**figure 8**).

Graphical user interface, text, application, email

Description automatically generated

***Figure 8.*** Calling the “is\_task\_in\_list\_function”.

Writing the new function took multiple test runs. This was maybe not the simplest or most elegant way to write the function, but it worked (***figure 9***).

Graphical user interface, text, application

Description automatically generated

***Figure 9.*** Searching the list for a task before adding it.

I tested it with a task that already existed in the list (**figure 10**).

Text

Description automatically generated

***Figure 10.*** Task is already in the list.

And I tested it with a task that wasn’t already in the list (**figure 11**).

Text

Description automatically generated

***Figure 11.*** Task is not already in the list.

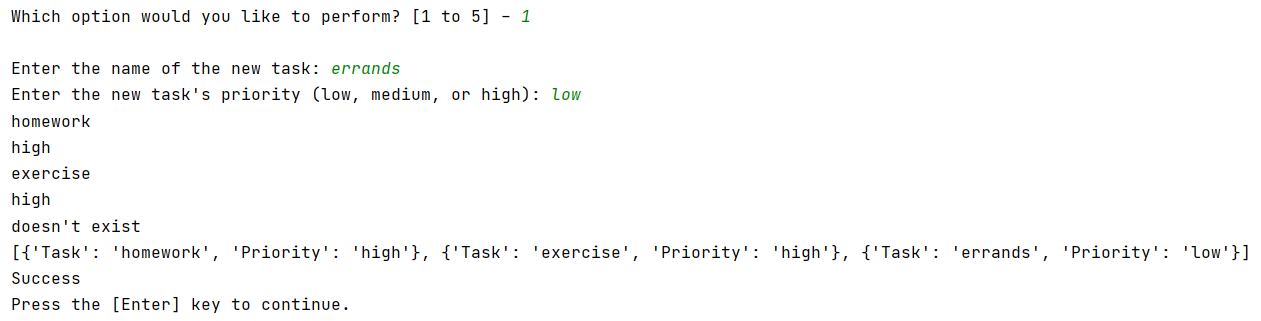
Then I added an if statement to add the item to the list if it didn’t already exist (**figure 12**).

Graphical user interface, text, application, email

Description automatically generated

***Figure 12.*** If task doesn’t exist, add it.

I also put in an else statement so that if I had time, I could give the user the option to change the priority if the task was already in the list. But for now, I didn’t write any code. I used pass to tell the program to skip ahead in the code. Then I ran the code to test it (**figure 13**).



***Figure 13.*** The task doesn’t exist, so add it to the list.

Menu option 2 allowed the user to remove an item from the list. In the main code I added two function calls, one to get the user input and one to remove the item from the list (**figure 14**).

Graphical user interface, text, application

Description automatically generated

***Figure 14.*** Remove a task from the list.

The input function was very simple (**figure 15**).

Graphical user interface, text, application

Description automatically generated

***Figure 15.*** Asking the user which task to delete.

The delete function was more complicated. I couldn’t just use my code from HW 05. I had to take into account the different way the list was created (**figure 16**).

Graphical user interface, text, application

Description automatically generated

***Figure 16.*** Delete a task.

After a bit of a struggle to get my code to work, I realized I could have reused my function “is\_task\_in\_list” that I wrote to check before adding an item. Then written a second function to delete the item. If I have time, I will go back and do that. But for now, this function worked as I expected (**figure 17**).

Text

Description automatically generated

***Figure 17.*** Item deleted.

The I tested it with an item that wasn’t in the list (**figure 18**).

Text, letter

Description automatically generated

***Figure 18.*** The task can’t be removed because it isn’t in the list.

Menu option 3 allowed the user to save the list to the text file. Again, in the main body of the script I added one line to call the function “write\_data\_to\_file” (**figure 19**).

Text

Description automatically generated

***Figure 19.*** Save the list to the text file.

I adapted my HW 05 code to pull out the values of the two keys in my dictionary and wrote them to the text file (**figure 20**).

Graphical user interface, text

Description automatically generated

***Figure 20.*** Adding only the dictionary values to the text file.

To test, I first added a new item to the list and then I saved it to the file (**figure 21**).

Text, letter

Description automatically generated

***Figure 21.*** The list is saved!

And checked the text file (**figure 22**).

Graphical user interface, text

Description automatically generated

***Figure 22.*** The saved list.

The list looked as I expected with a new item added to it.

The last menu option gives the user the opportunity to re-load the data from the file. I should be able to just repeat the first step of the main script body that loaded the data from the test file initially (**figure 23**).

Graphical user interface, text, application

Description automatically generated

***Figure 23.*** Re-loading the saved data in the text file.

I deleted a few items from my to do list and then chose option 4 (**figure 24**).

Text

Description automatically generated

Figure 24. The re-loaded to do list.

The original list was successfully re-loaded! All the tasks I had added were overwritten. I went back and removed all my extra print lines and tested the code using the command line.

I was able to run everything in command line successfully expect menu option 2 – removing an item from the list. If I removed the last item, it worked every time. But if I tried to remove any other item, I would get an error (**figure 25**).

Text

Description automatically generated

***Figure 25.*** Error removing an item in command line.

I went back and re-ran my program in PyCharm, and the same thing happened. I guess I hadn’t been very thorough in my initial testing.

After some efforts at debugging, I realized that when the function found the task to delete in the list it was exiting out of “if myValue == task:” but it wasn’t exiting out of the for loop just above it. As a result, i was always equal to its maximum value and so the last item was always being deleted from the list. I added another variable j, set it equal to the value of i for the list item I wanted to be deleted, then moved my delete statement to the next if statement. There is likely an easier way to fix my issue, but that’s what I came up with (**figure 26**).

Text

Description automatically generated with medium confidence

***Figure 26.*** Final version of the delete function.

### Conclusion

For assignment 06 we introduced functions. It should have been a simple matter of converting all the processing tasks and input/output tasks from homework 05 into functions. But we were given a starter script and the professor wrote things a bit differently than I did, so I had to re-work how my list was created and how to read to and write from the file. This created a lot of extra work but gave me a better sense of what I was doing. I was able to get the new code working in my test program, then transfer it to my homework program with a few tweaks to variable names. I still feel like my code is maybe a bit clumsy, but it’s getting easier to write it.