Chloe Wei

May 14, 2020

IT FDN 100 A

Module 06 Assignment 06

<https://cvarw.github.io/IntroToProg-Python-Mod06/>

Adding Functions to Legacy Code

Introduction

This week we will practice an important skill for programmers: working with legacy code, creating classes of functions from the main code body and how to call the functions within the main code.

Using Randal’s script for the ‘ToDoList’ from Module 05’s assignment, I will create functions in their respective classes then have them called based on the user’s selection from the User Interface, UI, as called within the main program.

A close up of a map

Description automatically generated

Figure 1: Structure of ToDoList

The menu options will be reorganized as functions in either the input-output class, “IO”, or the processing class, “Processor”, depending on the statements within the individual block of code.

Formulating a Plan and Solution

Once the run in the Python interpreter, the ‘ToDoList.txt’ will be read and the data stored into the list, lstTable. That same data will be transposed into dictionaries acting as rows of data containing the keys, “Task” and “Priority”. This and other functions that will pass arguments for processing are classified into the class, “Processor”. The functions that contain the respective arguments for the Processor functions are written under the class, “IO”.

First, I ran Randal’s script and test all the options making note of the systems output based on the inputs. After making a run of the script, I read through the code and wrote down a list of the functions needed, how to section off different blocks of code, where they were to be stored, and how to call them.

Adding New Data

Using the new function, input\_new\_task\_and\_priority(), to assign data to global variables, strTask and strPriority, that will then pass into the parameters for the function “add\_data\_to\_list()”, which will perform the task of adding the new dictionary into the list existing list of dictionaries, lstTable.

A close up of a sign

Description automatically generated

Figure 2: Replace code with call to IO.InputNewItem()

In the IO class, I relocated the arguments for the new task & priorities to local variables, task & priority, respectively.

A picture containing clock

Description automatically generated

Figure 3: New function in Class IO

I relocated the block of code (see figure 4) that appends the lstTable into the Processor class only after testing the input function of figure 2.

A close up of a sign

Description automatically generated

Figure 4: 'add\_data\_to\_list' function

Deleting an Item

Repeating the method for adding a new item to lstTable, the function for arguments were relocated to the IO as the “input\_task\_to\_remove()” function. The value parsed to the function remove\_data\_from\_list() withing the Processor class was checked for a match in the list table. If it made a match, then it would be removed, and UI would display it was found and deleted. If it was not found in the list, then the UI would display that a match was not found.

A screen shot of a smart phone

Description automatically generated

Figure 5: Removing task from lstTable

This script will stop searching through the remaining values upon its initial match and remove that first matching value. In other words, if there are more than one entry of the same name, only the first matching value will be removed.

Saving Data to a test file

Writing the data back into the file will only occur after the ‘Double-Check’ was made and user confirms saving in the main source code.

A picture containing black, table, large, holding

Description automatically generated

Figure 6: when ready to save data

The write\_data\_to\_file() function will then write the data into the text file, ‘ToDoList.txt’.

A picture containing phone, meter

Description automatically generated

Figure 7: write data back to file

Now that all the functions are created, I did a few more tests of the script.

A screenshot of a cell phone

Description automatically generated

Figure 8: Removing a task in PyCharm

As I have been using the PyCharm’s interpreter for tests & debugging, I was certain the code would run the same as the starter code. So I tried running the code in the Command prompt and was relieved to have the same results.

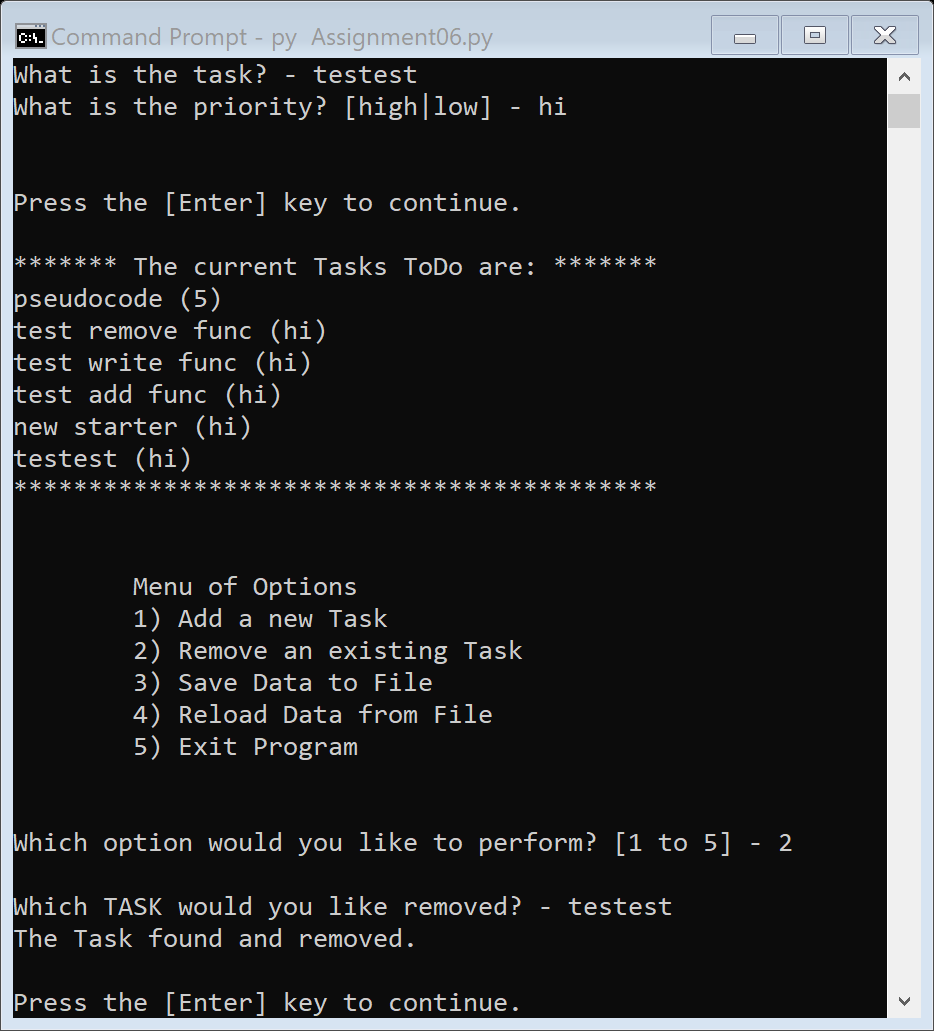


Figure 9: Script running in Command prompt to add data

Now that I’ve tested the functions and have attained the desired outputs, I proceeded to create a landing page for this and our remaining python assignments in GitHub. Took a little searching only to find the right help but was worth it to use their app then coding from scratch onto another server. Adjusting the look using the markdown for GitHub was straightforward and user friendly. After formatting the landing page, keeping it simple for now, I kept the link for how to use Markdown’s on the page for my (and my peers) reference.

Summary

This time around, we were to put into practice streamlining the main code of an existing script by creating functions without effecting the front end, UI, of the application. By writing the processing statements in functions makes reading through the main code easier and just looks better. Also, having the processes separated into different functions allows for easier debugging and, as learned in class, adjustments/editing. Although I used the debugger in our previous assignments, it is always good practice to use it when problem solving new or legacy code. It helps you see syntax errors or thought processes (if the code is extensive) within loops or in this scenario, class & function blocks. Overall, another opportunity to grow my understanding of python, its similarities to other coding languages and learning more about GitHub’s plethora of uses.