Kristen Doehlert

August 17, 2022

ITFDN 110 B

Assignment 06

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

Functions and Classes

# Introduction

In module 6 we learned how to organize our code into functions and classes. In this paper I am going to discuss how I modified the To Do List program from module 5 from a starter file that includes multiple classes and functions.

# Processor class functions

The processor class includes the functions used to process the data, like saving the data to a file or removing an item from a list in memory. The *read\_data\_from\_file* function was already completed in the starter file. The *add\_data\_to\_list* function was partially completed, with the code included to create a dictionary row based on the task and priority parameters. I added the missing code to append the new dictionary row to the *list\_of\_rows* that was passed into the function, then return the updated *list\_of\_rows*. This code can be found in Figure 1.

Graphical user interface, text

Description automatically generated

***Figure 1. Code for the add\_data\_to\_list function in PyCharm***

For the function *remove\_data\_from\_list*, I added a similar for loop to the one used in module 5. The primary difference is that the list (*list\_of\_rows*) and the dictionary element being searched for (*task*) are arguments passed into the function. The function returns *list\_of\_rows*, which is the new list with the matching dictionary row removed.

Text

Description automatically generated

***Figure 2. Code for the remove\_data\_from\_list function in PyCharm***

The code for the function to write data to a file is almost identical to the module 5 code. The only difference is that the file name (*file\_name*) and list of dictionaries (*list\_of\_rows*) are passed in as arguments. This function also returns *list\_of\_rows*, so that one of the IO functions can display the To Do List to the user after the data is saved. Figure 3 shows the code for writing data to a file.

Graphical user interface, text, application

Description automatically generated

***Figure 3. Code for the write\_data\_to\_file function in PyCharm***

# IO class functions

The IO class includes functions used to receive input from the user and present output to the user, like showing the menu, choosing a menu item, and choosing an item to remove. The functions *output\_menu\_tasks*, *input\_menu\_choice*, and *output\_current\_tasks\_in\_list* were already completed in the starter file. I added code to the *input\_new\_task\_and\_priority* function to get user input for the task and priority and assigned them to the local variables *local\_task* and *local\_priority*. These variables are returned at the end of the function, so that the values can be unpacked into a tuple and passed to the *add\_data\_to\_list* function when it is called in the main section of the script.

Text

Description automatically generated

***Figure 4. Code for the input\_new\_task\_and\_priority function in PyCharm***

The function *input\_task\_to\_remove* only has one input statement, which is assigned to the local variable *local\_task* and then returned for later use with *the remove\_data\_from\_list function*. Neither of these two IO functions have parameters since they are simply getting input from the user and returning it. Figure 5 includes the code to get the input for the task to remove.

Graphical user interface, text, application, email

Description automatically generated

***Figure 5. Code for the input\_task\_to\_remove function in PyCharm***

# Running the code

The code to call the Processor and IO functions was already completed in the main section of the starter file. I tested this script by running it in both PyCharm and Command Prompt. The screenshots in Figures 6-8 below are from testing in Command Prompt. First, I added the task “Do dishes, High” and this was successfully added to the list loaded from the text file, which only included “Mop Floors, Medium”.

Text

Description automatically generated

***Figure 6. Adding a new task in Command Prompt***

I removed the task “Mop Floors” from the list, and the new list displayed only included “Do dishes”.

Text

Description automatically generated

***Figure 7. Removing a task in Command Prompt***

Finally, I saved the data to a file. After the data was saved, the existing list was displayed again.

Text

Description automatically generated

***Figure 7. Saving to file in Command Prompt***

I checked the text file after saving the data and saw that it had been correctly updated from “Mop Floors, Medium” to “Do dishes, High”.

Graphical user interface, text, application

Description automatically generated

***Figure 8. Updated text file after saving to file***

# Summary

In this paper I discussed how I modified the to-do list script from the previous module to use functions and classes.