Tiffany Taylor

May 21, 2020

Classes & Functions

Assignment06

[GitHub Link](https://github.com/tiftaylor/IntroToProg-Python-Mod6)

Classes & Functions

# Introduction

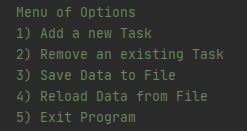
Assignment06 through “The Foundation of Programming” course with the University of Washington asks us to add code to an existing script that performs similar functions to Assignment05 (a menu of options for adding, removing, reading and writing to an external file). This time, the script divides the input parts of the program from the processing parts of the program and calls functions.

# Read the Existing Script

This exercise started with us needing to utilize a start script from the teacher. Before attempting to write the needed code, I read through each line of both comments and code of the existing script to make sense and build a framework in my mind of what I was working with and what already exists. This helped me to understand what still needs to be added using my own analytic thinking and compare it to what was written and being asked.

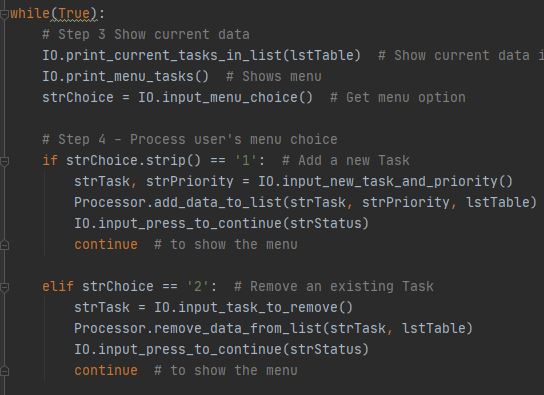
# Understanding the Result

The existing code gave us a pre-designed menu of options for manipulating data in a to-do list table. This data, which is stored in the memory, could be controlled via the menu of options selection. (Figure 1)



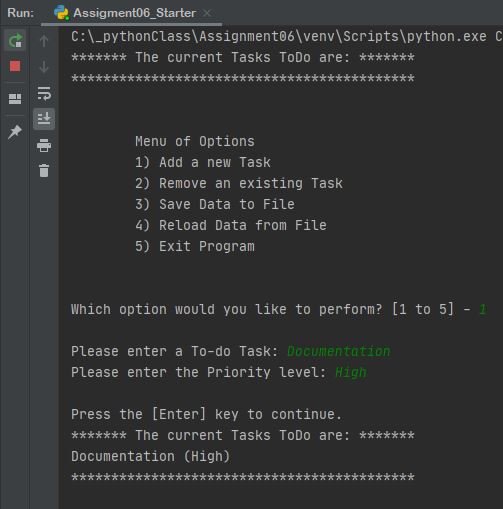
***Figure 1: Pre-designed Menu of Options***

To understand the ultimate job of the main script, we see a chain of if else statements that call functions from the “IO” class (where functions for input and print features exist) and from the “Processor” class (where functions that process the actions of the menu option. The class is attached in front of the function name and joined by the “.” dot notation. Figure 2 shows a portion of what this looks like:



***Figure 2: Main body of script calling class functions***

Choosing Option 1 (Add a new Task), will run the function “IO.input\_new\_task\_and\_priority()” asking the user for input. We store the input in a packed variable tuple so the next function on the line “Processor.add\_data\_to\_list()” can execute. We pass these variables into the parameters of the second function, including it’s expected 3rd variable the “lstTable” (a global variable). Selecting Option 1 creates a user interface that looks like Figure 3:

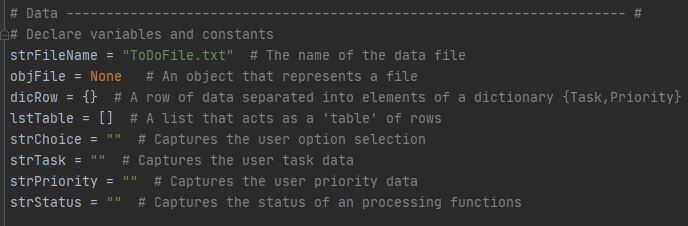


***Figure 3: Option 1 User Interface prompted by called functions***

The rest of the selections and user interface will behave the same way as the figures above except with their own different code to execute the specific need (like code to remove instead of add).

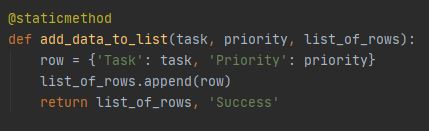
# Deeper Dive into the Code

It’s ideal to store any Global Variables used in the document at the top under the self-identified “Data” section, see Figure 4:



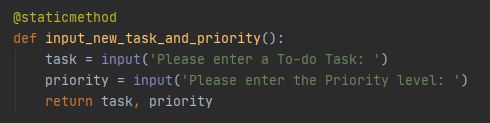
***Figure 4: Global variables at the top of script***

The next section of the script is the “Processing” section where we can create multiple functions that dictate how the data is processed literally. In this section, we avoid any print or input commands as it’s purely the part of the script focusing on what actions the code should take; in other words, the code for these functions will never have changing values (like an input would). These are static, process only style functions. Because all of these functions do something to the same file and list table array, we nest them under a python “class” and called it whatever name we wanted, in this case “Processor”. To keep code from feeling too long, anything within a class can be collapsed (hidden) under the main class line of code for easier script navigation (at least when using an IDE like PyCharm that gives you this feature). It will be used as a navigation word when the functions within it are later called (see back up in Figure 2). Figure 5 gives us an example of the processing function that doesn’t involve user interaction:



***Figure 5: Process function***

The last section before the main script is the “Presentation” section. Here, we followed the same structure as the Processing section but a new class is defined and new functions nested under that new class have code that revolves around and defines various values based on interaction with the user via print or input functions. To achieve asking the user for something like in Figure 3, we have stored the function to print the input for this information in this Presentation section, Figure 6 shows you what this looks like:



***Figure 6: Presentation function in a new class***

# Summary

In summary, creating functions in python can help us minimize typing repeat code. And further, nested functions under a class can help us keep functions that perform similar needs organized for easier reference and collapsible for easier script navigation. It’s important to remember if the functions called in the main script body need variables if information is being stored from the user and then used as parameter input for other functions that are not interacting with the user, but need the info from the users input. Be care of shadowing any global variables, it could mess stuff up and is not best practice. Best to just keep local variables in your function’s different names than any from the global variable list at the top of a script.