Dave Phillips

2021-08-11

IT FDN 110 A Su 21: Foundations Of Programming: Python

Module 6 – Assignment 1

To-Do or Not To-Do

# **Introduction:**

In this assignment we’ll be completing portions of a template script provided by the instructor. The goal is not only to work with classes and functions, but also to practice working with, and altering formatting styles to match, another person’s code. I’ll include a brief overview of the already-completed portions of code and will only go in depth about the sections which were part of the assignment – those having to do with adding, subtracting, and saving records.

# **Section 1: Variables and Processor Class**

**Lines 14 – 21 were already present in the file. Simple variable definitions for objects used throughout the script.**

**Beginning at line 24 is the Processor class, which handles the internal workings of the script, that is, those that don’t require communication with the user. The read\_data\_from\_file method simply reads data in a previously written text file and loads it into the program’s memory, closing the file when complete. This was pre-coded.**

**The following method, add\_data\_to\_list, is defined on lines 54 – 57 (Figure 1) with a docstring preceeding, and receives both the task and priority level (previously captured using the input\_now\_task\_and\_priority method – discussed further down) as well as the master list\_of\_rows object. The values are stored in a dictionary object with the labels “Task” and “Priority” respectively. It then appends the dictionary to the master list of values in memory.**

1. **dicRow["Task"] = task**
2. **dicRow["Priority"] = priority**
3. **list\_of\_rows.append(dicRow)**
4. **return list\_of\_rows, 'Success'**

**Figure 1. Add new data to master list**

**The remove\_data\_from\_list method performs the opposite task. (Figure 2) Only taking in the task name and master row list as arguments, the function iterates through all items in the master list\_of\_rows and, if a [“Task”] is found matching the value passed in the input\_task\_to\_remove function (explained shortly), it is removed from the master list via the pop method.**

1. **for item in list\_of\_rows:**
2. **if item["Task"] == task:**
3. **list\_of\_rows.pop(item)**
4. **return list\_of\_rows, 'Success'**

**Figure 2. Remove record from master list**

**The final method in this class is the write\_data\_to\_file function which opens a file handler, writes the content of list\_of\_rows, and closes the file. Simple, easy step. (Figure 3)**

1. **filehandle = open(file\_name, "w")**
2. **filehandle.write(list\_of\_rows)**
3. **filehandle.close()**
4. **return list\_of\_rows, 'Success'**

**Figure 3. Write master list to file**

# **Section 2: Input/Output Class**

**The presentation portion of the script is contained in the IO class (short for Input/Output). A majority of the functions were pre-coded into the assignment, including some self-explanatory functions including print\_menu\_tasks, input\_menu\_choice, print\_current\_Tasks\_in\_list, input\_yes\_no\_choice, and input\_press\_to\_continue.**

**The first function in this class required for the assignment is the input\_new\_task\_and\_priority method. This is ultimately a simple input function designed to capture the Task and Priority values and return a tuple for further processing. Note the additional strip and title methods applied to the input on line 157. (Figure 4) I’ve removed any leading or trailing whitespaces and capitalized each word in an effort to create as clean and uniform a list as possible. To make entry as simple as possible for the user, I’ve designed the priority input section to receive a numerical value and automatically apply the string value of the priority level instead of asking for the fully typed out word(s).**

1. **task = input("Please enter the name of the task you wish to add - ").strip().title()**

**Figure 4. Input formatting**

**The other method written for this assignment is the input\_task\_to\_remove function which, on line 185, simply requests the name of a task from the user and formats it with the same strip and title methods outlined in the previous function.**

# **Section 3: Main Loop**

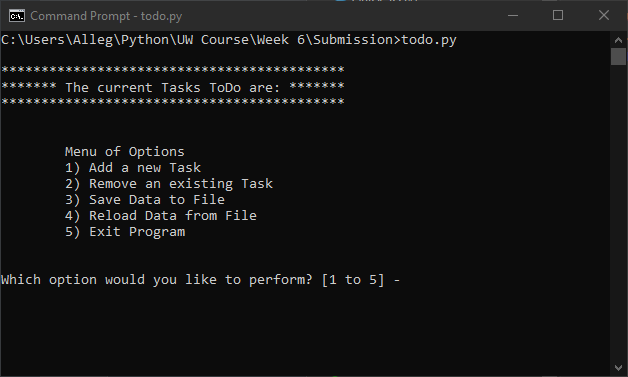
Before starting the primary loop of the function, on line 192 the script calls the **read\_data\_from\_file** method, defined in the Processor class, to open the target file and load its contents into the list object that will be manipulated by the program.

The **while** loop beginning on line 195 begins the primary body of the script. The layout is relatively uniform and each **if/elif** option is written in essentially the same way. So, depending upon the input from the user (an integer between 1-5, corresponding to the options displayed in the **print\_menu\_tasks** function), this function will perform an action from the **Processor** class followed by an **IO** class function. Setting the main loop up in this way helps maintain a clear code style and usage can be inferred easily by an individual who didn’t originally program the script.

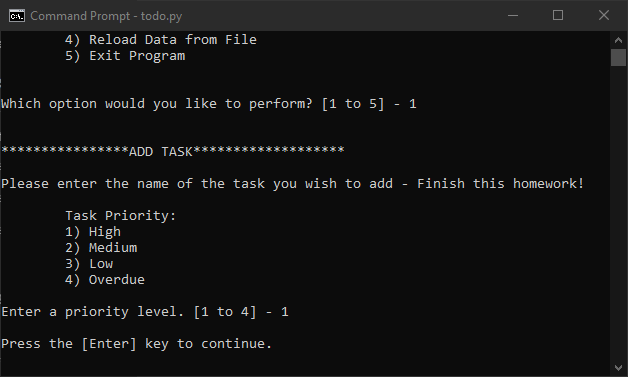
# **Summary:**

The difficulty in this assignment is how best to keep code clean and organized while still maintaining uniformity with the pre-existing script. The use of classes and functions significantly reduces the amount of duplicate code and, additionally serves to further differentiate between the various stages of interaction with the program (Data/Processing/I-O model).

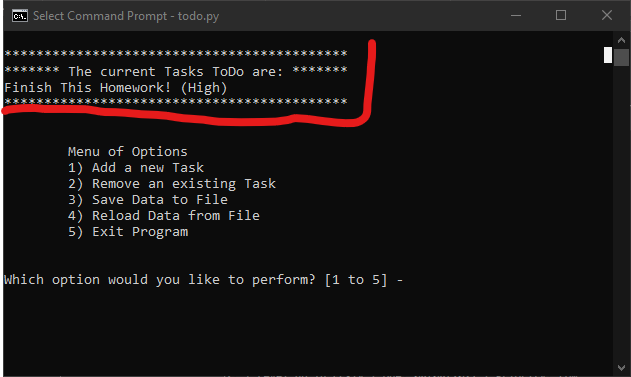
Runtime screenshots can be found in the following appendices.



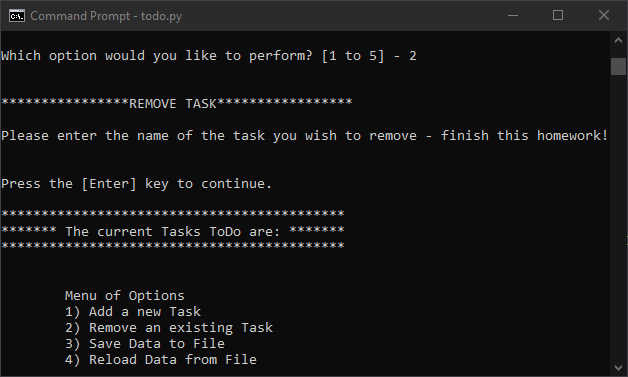
Appendix A. Initial menu via command line



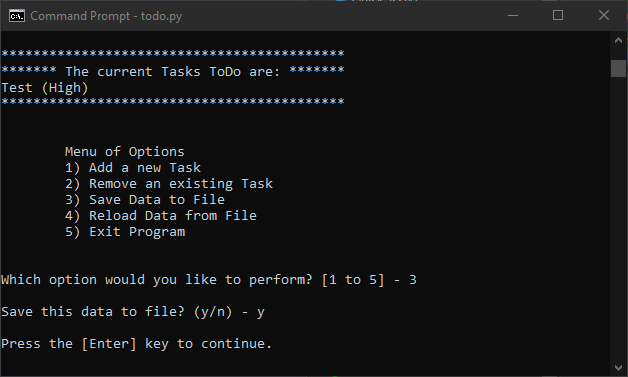
Appendix B. Add Record



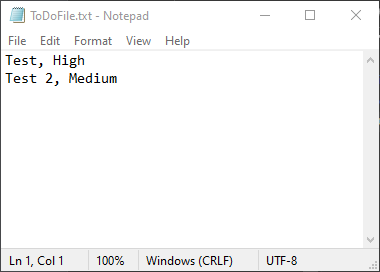
Appendix C. New record appears in list



Appendix D. Remove Record



Appendix E. Save to file



Appendix F. Raw .txt file