Linda Shmait

8/16/2022

Foundations Of Programming: Python

Assignment 06  
https://github.com/lzs425/IntroToProg-Python

# To Do File

## Introduction

This article will build on the script built in A05, essentially centering on building a to do list with several action items the user can take. This script builds on last week’s general functionality, which involves displaying a menu, taking in user input, executing a process based on that selection, and returning some information or notification.

The difference this week is that instead of writing functions under a while loop, the script is organized in three main sections, the Processing section with the Processor class with methods operating on the data, the Presentation section with the I/O class that deals with the user input/output needs, and then the Main section that calls on those functions and methods with the relevant parameters to execute the required steps.

The code will appear like Figure 1 below when run, though there may be differences based on user selection.

***Figure 1. Portion of Completed Script Display***

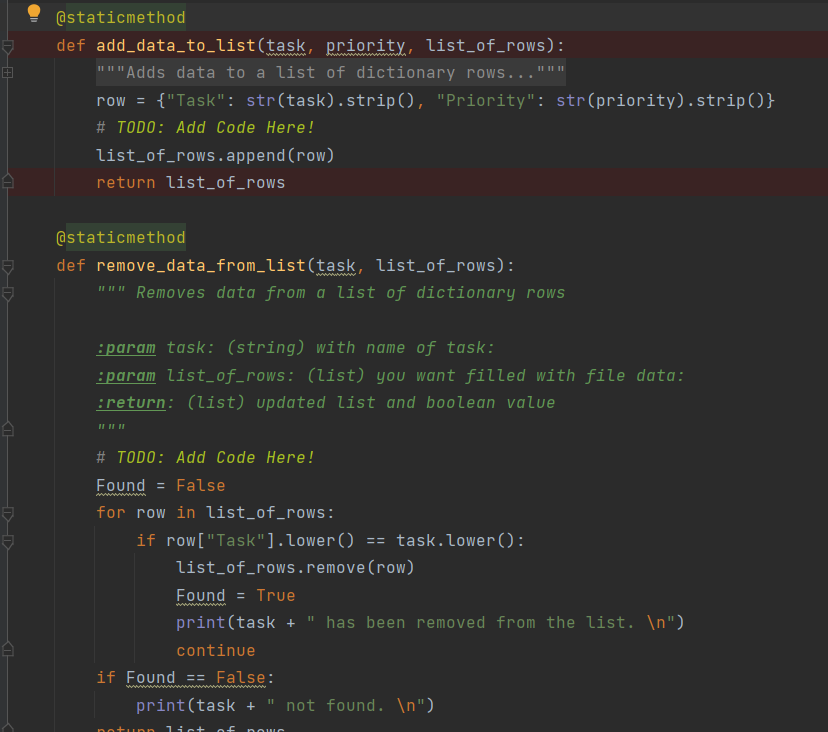
## Script

The script consists of Processing data, Presentation, and Main sections as mentioned above. Much of the code and structure was already provided, so the primary task in Assignment 06 was to complete the methods within the Processing and Presentation sections. This organization separates the script for completing certain actions from the section that executes them (Main) in order to lend the code structure and clear to anyone reviewing the code. Once the methods within the Processor and I/O classes were completed and key outputs determined in the return portion, the Main section called them to perform with the relevant parameters input.

### Processing Section

The processing section included methods to





### Processing

The processing section of this script pulls data from an external ToDoList.txt file and loads it into a table. This table will have rows made of dictionaries, similar to a list but leveraging key subscripts instead of indices, as seen in Figure 2 below. We need the code to read all lines of the file, which is why after I get the file using the open() function in the read format, I use a for loop to cycle through all lines of data and create a new table.

Within the loop to add the text file data to a table of dictionaries, I am separating the task and priority using the split() function for data stored in each row of the text file in line 32. I then store it in a dictionary in line 33 and creating the keys “Task” and “Priority”. The strip() function is used on the lstRow index 1 to get rid of any carriage returns. For each dictionary I create, I then append it to the lstTable using the append() function, and the for loop will continue this cycle until all rows of data in the text file have been read and my lstTable has all the data in lines of dictionaries.

Text

Description automatically generated

***Figure 2. Loading Data into a List of Dictionaries***

### Input/Output

Much of the structure of the Input/Output section was provided. Its overarching structure is a while loop with nested if statements. While(True) in this case would continue to run until it gets to a break. Within that while loop, the code displays the menu using the print function, captures the user input, and based on that routes to a particular section of code by way of a series of if/elif/else statements. Only Option 5 has the break option, so the user can continue looping through the options until they choose to exit.

#### Option 1: Show Current Data

When the user inputs 1, they are selecting to print any existing data we have loaded in the table lstTable, as seen in Figure 3. As we have in previous assignments, I used a for loop to go through each row and print. The advantage of dictionaries is evident in line 54 of Figure 3, where within the print function I am able to use the key, providing more meaningful information as to what I want to print, rather than an index value that requires more knowledge of the original table to understand. Once this is complete, line 55’s “continue” will put the user back to the main menu to input their next option.

Graphical user interface, text

Description automatically generated

***Figure 3. Printing a table of dictionaries***

#### Option 2: Add a New Item to the List

If the user wants to input a new task and priority level into the to do list stored in lstTable, they will select option 2 from the menu. The script will then route to the code displayed in Figure 4. I use two variables previously declared in the Data section, strTask and strPriority, and setting them equal to the inputs for task and priority respectively in rows 59 and 60. I can then store them in dicRow in line 61, and append the newly created dicRow to lstTble in line 62. It is convenient again to be able to reference the key, and ensure that I am inputting the correct variables with the corresponding keys.

Text

Description automatically generated

***Figure 4. Appending a Dictionary Row to a Table***

#### Option 3: Remove an Existing Item

If the user wants to remove a task and enters ‘3’ for option 3, the elif statement in Figure 5 on line 65 will be true. The code will prompt an input from the user for the task they want to be removed, stores it in a variable, and then a for loop goes through the table of to do items and checks for matches.

The lower() function is used to ensure uniformity of letter case and improve the chances of an intended match. The remove() function is responsible for eliminating the matching item to the user input. Because I have the print statement within the loop, I use a true/false if statement based on the variable Found to avoid printing the ‘not found’ note for every row the for loop goes through. The user will then get one message that the input was either removed or not found, and then looped back to the menu using continue.

Text

Description automatically generated

***Figure 5. Removing an Item from Tasks***

#### Option 4: Saving Tasks to the ToDoList File

In similar processes to our previous assignment, the user can choose to store their data in a text file. The code for this process first opens the file in write mode as seen in line 81 of Figure 6 below. The for loop that follows then turns the dictionary rows into strings separated by a comma, adding a carriage return. This transformation back into a string can be seen in line 83, putting the data in the format that the code in the Processing section basically undoes to return it to a dictionary.

Text

Description automatically generated

***Figure 6. Saving Tasks to the Text File***

#### Option 5: Exit

The last choice the user has in the menu is to exit the program. This is very similar to Assignment 04’s exit logic. Using an input(), the user is prompted to confirm the exit request, and if they enter ‘y’ or ‘Y’, the if statement logic and lower() will route the script to the break, ending the while loop that started the Input/Output section.

### Testing

While I was testing each section of the script as I went, my last check is to run through the program in Command Prompt as well as PyCharm. Once I changed the directory to Assignment 05’s location, I ran through the script in Command Prompt, an example of which is in Figure 7 below. I verified that all options performed as intended, and checked the ToDoList.txt file for the correct data.

Text

Description automatically generated

***Figure 7. Command Prompt Testing***

## Summary

This script used dictionaries to allow data to be referenced by key instead of index, and the strip() function to remove extra spaces and carriage returns, as well as using the functions and methods from previous assignments. While/break, for loops, and if statements were key to the structure of this script as well. In all, the script ToDoList will allow a user to view, add, remove, and save a to do list with priority level.