Name: Silvano Ross
Date: February 22, 2022
Course: IT FDN 110A
GitHub: https://github.com/silvanoross/IntroToProg-Python

Assignment 06 - Agenda by a Function

INTRODUCTION

This week we had to transform a more simpler version of a to-do list into one that was mostly pre-written by our professor and included the extensive use of functions. I had originally started by trying to work from top to bottom as that seemed the most sensible way of tackling this project. Turns out everything I wrote prematurely did not work in the main script where all of the functions were called. I ended up having to start from the main script and piece each function together in a reverse order.

STARTING FROM THE BOTTOM

It was key to look at the main script first to see how the different functions were called and which local variables were already passed into these functions. Prior, all the work I did, creating unique variables just made trying to implement them into the prewritten main script a

headache.

```
Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst)  # read file data

# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.output_current_tasks_in_list(list_of_rows=table_lst)  # Show current data in the list/table
    IO.output_menu_tasks()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    # Step 4 - Process user's menu choice
    if choice_str.strip() == '1':  # Add a new Task
        task, priority = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '2':  # Remove an existing Task
        task = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '3':  # Save Data to File
        table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
        print("Data Saved!")
        continue  # to show the menu

    elif choice_str == '4':  # Exit Program
        input("Press the Enter key to quit: ") # Added a pause to allow for user input
        print("Goodbye!")
        break  # by exiting loop
```

Figure 1.1 Taken from Assignment 06

After reviewing how the main script called the functions I could better understand what each function needed to accomplish. I did not have to create multi-compound functions, but was better guided towards what I needed to accomplish. By doing reverse engineering it was easier to see the work I needed to do and the work I didn't need to do. It also helped me stop and slow down, fully analyze the situation and start where it seemed easiest rather than jumping headfirst into the deep end. This also helped me see how the Separation of Concerns (SOC) played out in the long run.

SEPARATION OF CONCERNS

Most of the choices the user enters are clearly divided into the "Presentation" and "Processing" areas of the SOC. First a function is called to get the user input. Next the user input is shunted off into a processing function that transforms what was entered into whatever choice the user

made. This makes the main code that calls the functions look very neat and it was nice to see this in action (Figure 1.1).

CHANGES MADE TO OLD CODE

In terms of writing the actual code it was easier than I had expected. The hardest part was figuring out where to start. First, I took on the simple input functions that returned a task or a task paired with a priority.

```
def input_menu_choice():
    """ Gets the menu choice from a user

    :return: string
    """
    choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
    print()  # Add an extra line for looks
    return choice

@staticmethod
def output_current_tasks_in_list(list_of_rows):
    """ Shows the current Tasks in the list of dictionaries rows

    :param list_of_rows: (list) of rows you want to display
    :return: nothing
    """
    print("******* The current tasks ToDo are: *******")
    for row in list_of_rows:
        print(row["Task"] + " (" + row["Priority"] + ")")
    print("*******************************************")
    print()  # Add an extra line for looks

@staticmethod
def input_new_task_and_priority():
    """  Gets task and priority values to be added to the list

    :return: (string, string) with task and priority
    """
    task = input("Enter the task you would like to add to your list: ")
    priority = input("Enter the priority for this task - low/med/high: ")
    return task, priority # maybe needs to be a tuple?

@staticmethod
def input_task_to_remove():
    """  Gets the task name to be removed from the list

    :return: (string) with task
    """
    task = input("Type in which task you would like to delete: ")
    return task
```

Figure 2.1 Taken from assignment 06

All of these functions were very straightforward and taken from the assignment 05 for the most part. It was pretty easy to put them in their place. The processing functions were also taken from the older assignment with some slight modifications of variable names so that they matched the global variables and function parameters. But the writing data to a file, menu option display and choice and adding data were all the same from last week's assignment. Just packaged in individual functions.

REMOVING DATA

The hardest part in last week's assignment for me was trying to figure out how to remove the items in the working table_list variable. I instead turned to the answer key from last week and used the code from that assignment as I just could not come up with an efficient way of deleting data without running around in circles. I found this code to be very tricky and interesting and would like to go into detail about it.

```python
@staticmethod
def remove_data_from_list(task, list_of_rows):
    """ Removes data from a list of dictionary rows

    :param task: (string) with name of task:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    b_ln_item_removed = False  # Creating a boolean Flag
    int_row_number = 0
    for row in list_of_rows:
        stask, priority = dict(row).values()
        if stask == task:
            del list_of_rows[int_row_number]
            b_ln_item_removed = True
        int_row_number += 1
    # Step 5b - Update user on the status
    if (b_ln_item_removed == True):
        print("The task was removed.")
    else:
        print("I'm sorry, but I could not find that task.")

    return list_of_rows
```

```python
elif choice_str == '2':  # Remove an existing Task
    task = IO.input_task_to_remove()
    table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
    continue  # to show the menu
```

Figures 3.1 & 3.2 Taken from assignment 06

This is the first time I had experienced working with a boolean flag and found it enlightening to see it in action. The function starts off labeling the variable for the boolean flag as FALSE. Only if the 'task' entered in by the user from the IO function was the same as the 'stask' that was pulled out of the list_of_rows variable would a deletion occur and then the flag would change to TRUE printing out a statement being triggered by the flag being TRUE. This code is still confusing to me, but it seems to work well.

SUMMARY

This week was more of a puzzle rather than just hard-coding a program. The layout was in front of us with a few missing essentials and we had to use the code from last week's assignment to create a ToDo list program using functions. All-in-all it was easy if you took a look at the whole program before trying to add anything. Seeing how functions can take the return results of other functions was very eye-opening. Finding an efficient way to remove items from a list with a user input was still difficult and still a bit unclear. I am hoping to figure this out over the course of next week.