Aisha Shafique
05/19/2022
Foundations of Programming: Python
Assignment 06
https://github.com/Ai-Shaf/IntroToProg-Python-Mod06

Programming with Classes & Functions

Introduction

This week's assignment focuses on using classes and functions to better divide our code into data, processing, and presentation. By having a separate class of presentation functions and processing functions, we can ensure that even our functions are appropriately divided by best practice.

We were given starter code that already lists out the classes and functions that will be used in the program and separate them from the main script. Doing this assignment helped to reinforce the division between data, processing, presentation; to get us used to practice calling functions from within classes; and learning how to execute the functions systematically in the main script to accomplish our tasks.

Writing the Script

Data

The script started out naming the global variables that would be used in the program. The local variables that are encapsulated in the functions are not similarly listed out. For the most part, this is because they are localized enough that we don't need to reuse them as much and so the need for having them in an easily accessible place to follow them along is not as important. To a lesser extent, it would become quite tedious to list out every variable that was ever used. In general, the rule of thumb seems to be to list variables that will be throughout the code.

Processing

The first class in the starter script for Assignment 06 is the class Processor. Here, we have the functions that read data from file, add data to the list, remove data from the list, and write data to file. These functions are processing the data.

Though the read data from file was already complete in the assignment, the other three tasks processes data were part of the assignment for us to do.

I. Add data to the list

To add data to the list, I first added a doc string that explained my parameters and what the function would be returning. The function takes in arguments for the parameters task, priority, and list_of_rows. The function then simply states that task and priority are added as values to the keys "Task" and "Priority" and appended to the list_of_rows. The function then returns the updated list_of_rows as well as a string 'Success'. See *Figure 1* for the code.

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    # TODO: Add Code Here! Done
    """
    :param task: inputted task that user wants to add to list
    :param priority: inputted priority user wants to add to list
    :param list_of_rows: the list of dictionary rows tasks and priorities are
appended to
    :return: the updated list of rows with the added tasks
    """
    list_of_rows.append({"Task": task, "Priority": priority})
    return list_of_rows, 'Success'
```

Figure 1: Adding task and priority arguments and appending them to list_of_rows

II. Remove data from the list

The second function removes data from the list. It has two parameters task and list_of_rows. It takes an argument for task, checks to see if that is within a row of the table list argument passed to list_of_rows, and then, it removes the row the task is found in. If the task was found, it returns the updated list_of_rows (or rather, the argument that is passed into that parameter) as well as the string 'Success.' If, however, the task was *not* present in the rows, then it still returns the list_of_rows but the string printed out is 'Task not found.' We can see the script for this below in Figure 2.

```
@staticmethod
def remove_data_from_list(task, list_of_rows):
    # TODO: Add Code Here! Done
    """

    :param task: inputted task the user wants to delete the task/priority
pair for
    :param list_of_rows: the list of dictionary rows tasks and priorities are
saved in
    :return: the updated list_of_rows with the inputted task row removed (if
it was in it),
             whether the entered task was successfully removed or whether it
was not found
    """
    Present = False
    for row in list_of_rows:
        if row["Task"].lower() == task.lower():
            list_of_rows.remove(row)
            Present = True
    if Present == True:
        return list_of_rows, 'Success'
    else:
        return list_of_rows, 'Task not found. See available current tasks.'
```

Figure 2: Removing the row with specified task and using conditional statements for return

III. Write Data to File

Similar to the last assignment, this function writes data to file. This function takes two parameters—one for the file_name, and one for the table list from which data is to be extracted and written into the file. As in the earlier assignment, after opening the file, we use a for loop to loop over the list and write it into the file as strings separated by a comma. We use keys to extract the string values from dictionary rows. Again, as shown in *Figure 3*, the function returns whatever argument is passed into the parameter list_of_rows, as well as a string 'Success.'

```
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    # TODO: Add Code Here! Done
    """

    :param file_name: the name of the file that the data will be written to
    :param list_of_rows: the list of dictionaries the data is saved in
    :return: list_of_rows that has just been written to the file
    """
    file = open(file_name, "w")
    for row in list_of_rows:
        file.write(row["Task"] + "," + row["Priority"] + "\n")
    file.close()
    return list_of_rows, 'Success'
```

Figure 3: Writing data to file

Presentation

The presentation part of the code deals with inputs and outputs of data. So, here we have the functions that use the input() function and the print() functions. The class for this is set as class IO (for input output).

The starter script already has the following functions completed here: print_menu_Tasks(), which prints the menu of tasks, input_menu_choice(), which asks user to choose which of the menu items s/he would like to do & returns that selection, print_current_tasks_in_list(list_of_rows), which takes one parameter for the list of data rows, extracts strings of data from the dictionary rows and prints them out.

We also have input_yes_no_choice(message), which takes a message parameter. Here, we can add any string inside to ask/tell user to respond to. This function returns the response of the user. Input_yes_to_continue(optional_message= ' ') has a parameter that already has a default value of a space set. This allows this parameter to be an optional message. If we don't add in anything, it'll simply print a space and move on to the next line. Then, it'll ask the user to press enter. That way, the while loop in the main script can continue.

Now, the next script we have to do ourselves. This is input_new_task_and_priority() function. Here, user input asking them to enter a task is captured in a variable task, and user input on priority Is captured in priority, and then, the variables task and priority are returned. See Figure 4 below.

```
        @staticmethod
def input_new_task_and_priority():
    pass  # TODO: Add Code Here! Done
    """
    :return: returns user inputted task, priority data
    """
    task = input("Enter task you wish to be added: ")
    priority = input("Enter priority of the added task [Low, Medium, High]:
")
    return task, priority
```

Figure 4: return variables saving the task and priority input

The last function we have in the IO class is input_task_to_remove(). Here, again, I capture the task the user wants to remove in a variable and then I return it. This is shown in Figure 5.

```
@staticmethod
def input_task_to_remove():
    pass  # TODO: Add Code Here! Done
    """
    :return: returns the task that user wants to remove the row of
    """
    task = input("Enter task to be removed: ")
    return task
```

Figure 5: Task user wants to remove

Main Script

The main script follows much the same as Assignment 05, except that we don't have to write the entire code out. Now, we have classes and functions to implement the program. We begin with the loop *while True:* and then in the code block, we first display the current data we have by calling the function *IO.print_current_Tasks_in_list(lstTable)*. We follow this by calling the function that displays the menu of items, also in the class name. function name() format. Next, we call the function that captures the user's choice and we capture the return in the variable strChoice. This is all part of the starter code (see below *Figure 6*).

```
while (True):
    # Step 3 Show current data
    IO.print_current_Tasks_in_list(lstTable)  # Show current data in the
list/table
    IO.print_menu_Tasks()  # Shows menu
    strChoice = IO.input_menu_choice()  # Get menu option
```

Figure 6: Starter code for beginning program loop

Our assignment begins as we define what happens as *strChoice* values change from 1 to 5. Here, I use *strTask* and *strPriority* to capture the return of the function *IO.input_new_task_and_priority()*. I can then feed these variables as arguments for my next function, which is *Processor.add_data_to_list(strTask, strPriority*). I capture the return of this function as *lstTable*, *strStatus*. Now, when I run the function *IO.print_current_Tasks_in_list(lstTable*), I get an output of the updated function with the added item. Lastly, when I run *IO.input_press_to_contine(strStatus)*, I get the returned 'Success' from my processor class function as well as the instruction to press enter to continue. Pressing enter will then *continue* the program, restarting it from the beginning of the while loop.

```
if strChoice.strip() == '1':  # Add a new Task
    # TODO: Add Code Here Done
    strTask, strPriority = IO.input_new_task_and_priority()  # Capture the
returned values from the function in variables
    lstTable, strStatus = Processor.add_data_to_list(strTask, strPriority,
lstTable) # Add the captured data to the list
    IO.print_current_Tasks_in_list(lstTable) # Show the user the updated list
    IO.input_press_to_continue(strStatus) # Ask the User to press enter to
continue after displaying 'Success'
    continue  # to show the menu
```

Figure 7: Code when User chooses to input new task.

If the user selects '2,' the program asks the user what task to remove. This is done through the function *IO.input_task_to_remove().* Again, my *strTask* variable captures the return and feeds it into the next function, *Processor.remove_data_from_list(strTask, lstTable).* This second function returns *lstTable* and *strStatus*, which I capture below. I can then print out the updated list of Tables. If the specified task was

in the dictionary rows, it will have been removed from *lstTable* and the *IO.input_press_to_continue(strStatus)* function will display 'Success.' However, if the task was not in *lstTable*, then *lstTable* will be unchanged and my status message will be 'Task not found.' The user then hits enter and taken back to the menu of tasks to choose from to try again (Figure 8).

```
elif strChoice == '2':  # Remove an existing Task
    # TODO: Add Code Here Done
    strTask = IO.input_task_to_remove()  # user inputs task to remove
    lstTable, strStatus = Processor.remove_data_from_list(strTask, lstTable)
# row of entered task is removed, updated table & status saved
    IO.print_current_Tasks_in_list(lstTable)  # displays updated list of
current tasks
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

Figure 8: Main script for removing task from list

If the user selects '3', the program saves the data. If the user chooses 'y' to saving the data to file (which is run through calling *IO.input_yes_no_choice()),* I call the function *Processor.write_data_to_file(strFileName, lstTable),* capturing the returns in *lstTable, strStatus.* I then output the status 'Success' and ask user to press any key.

If, however, the user chooses other than 'y' (implicit no), then the message "Save Cancelled" shows up with instructions for user to hit enter to continue. The continue takes the user back to the menu of tasks.

```
elif strChoice == '3':  # Save Data to File
    strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
    if strChoice.lower() == "y":
        # TODO: Add Code Here! Done
        lstTable, strStatus = Processor.write_data_to_file(strFileName,
lstTable)  # write data to file & save return variables
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("Save Cancelled!")
    continue  # to show the menu
```

Figure 9: Save Data to File

The fourth menu item allows the user to reload the data. We begin this by printing a warning that 'Unsaved data will be lost!' Then the user is asked if they still want to reload the data. If 'y', we call the function *Processor.read_data_from_file(strFileName, lstTable)* and save the output into *lstTable* and *strStatus*. I then display *lstTable* by passing it as an argument in *IO.print_current_Tasks_in_list(lstTable*). Lastly, I display the *strStatus* variable and request the user to hit enter to continue, which will take the user back to the menu of tasks.

If the user had decided not to reload the data, then the message 'File Reload Cancelled' is displayed and the user is asked to hit enter to continue, again being taken back to the menu of tasks.

```
elif strChoice == '4':  # Reload Data from File
    print("Warning: Unsaved Data Will Be Lost!")
    strChoice = IO.input_yes_no_choice("Are you sure you want to reload data
from file? (y/n) -  ")
    if strChoice.lower() == 'y':
        # TODO: Add Code Here! Done
        lstTable, strStatus = Processor.read_data_from_file(strFileName,
lstTable)
        IO.print_current_Tasks_in_list(lstTable)
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("File Reload Cancelled!")
    continue  # to show the menu
```

Figure 10: Reload data

The last option in the menu is number five, which is to exit the program. If the user chooses this, the message 'Goodbye!' is displayed and the program breaks out of the loop.

```
elif strChoice == '5':  # Exit Program
    print("Goodbye!")
    break  # and Exit
```

Figure 11: Exiting the Program

Running the Script

I ran the program for each option; the outputs are displayed below in the figures.

We can see that the current tasks are shown, followed by the menu of options. Selecting 1 leads to being asked to enter task to add and then to add the priority of the task. The new tasks are then printed out.

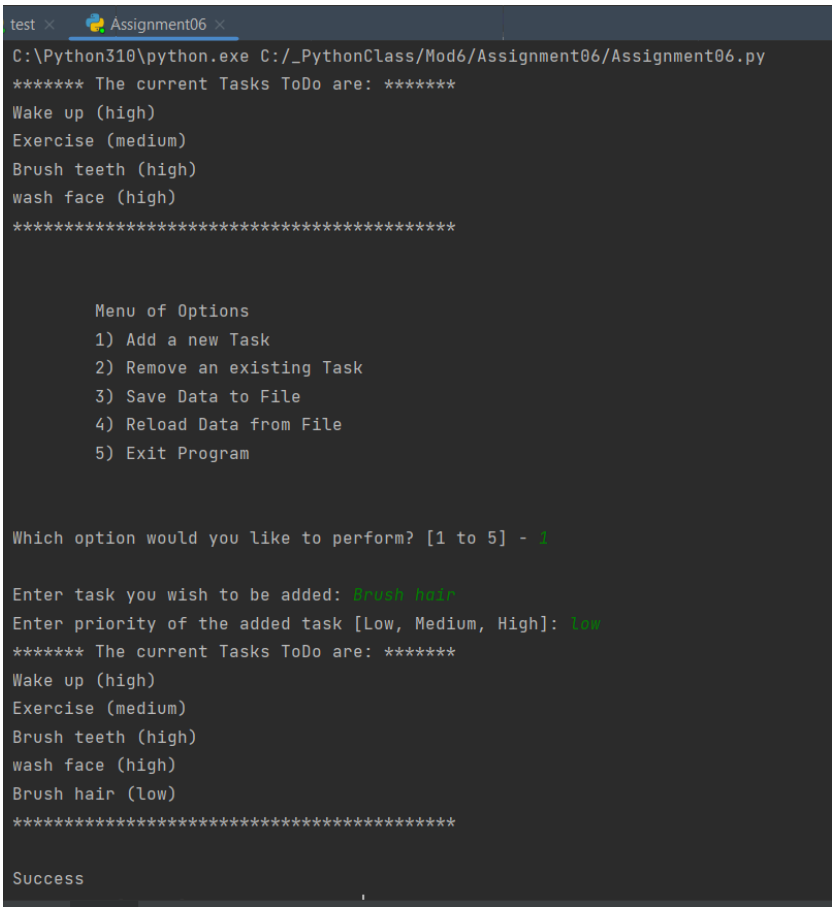

Figure 12: Exiting the Program

Selecting 2 leads to being asked what task we want to remove. If I choose a task that is not in the list, as shown in *Figure 13*, the program outputs 'Task not found.'

If, however, I choose a task that is found in the table, I am shown the updated tasks list as well as the message 'Success' (*Figure 14*).

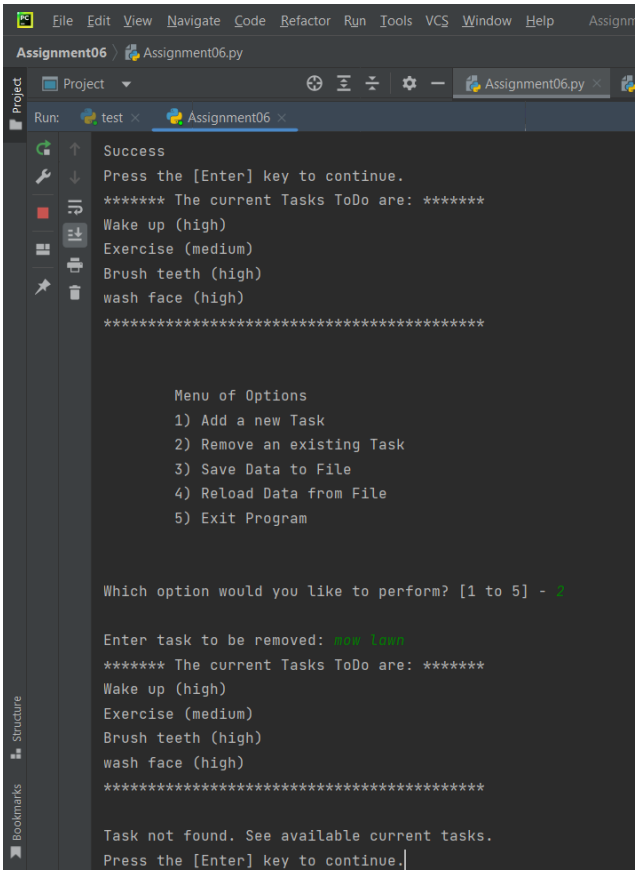

Figure 13: 'Task not found'

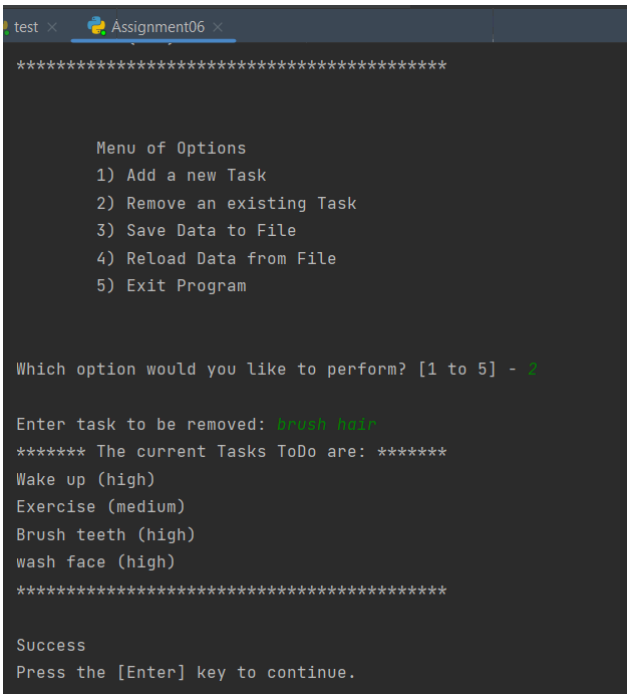

Figure 14: Successful removal of task

The third option allows me to save the file, as shown in Figure 15. I choose ‘y’ to save. If I chose ‘n’, I would return the menu of tasks.

The fourth option allows me to reload the file, but after warning that if I do so, I will lose unsaved data. If I had pressed ‘n’, I would have gone back to the main menu without reloading. Figure 16 shows when I do reload.

```
Which option would you like to perform? [1 to 5] -
Save this data to file? (y/n) - y
Success
Press the [Enter] key to continue.
```

Figure 15: Saving the Data

```
        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] - 4

Warning: Unsaved Data Will Be Lost!
Are you sure you want to reload data from file? (y/n) -  y
******* The current Tasks ToDo are: *******
Wake up (high)
Exercise (medium)
Brush teeth (high)
wash face (high)
```

Figure 16: Reloading the Data

The last option, option 5, simply prints out ‘Good-bye,’ breaking out of the loop and ending the program.

```
        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] - 5

Goodbye!

Process finished with exit code 0
```

Figure 17: Exiting the program

I can run this program in the Windows OS Console as well. *Figure 18* shows a sample of the program being run in the Windows OS shell.

```
C:\WINDOWS\system32\CMD.exe - python.exe  assignment06.py
C:\Users\Herderess>cd C:\_PythonClass\Mod6\Assignment06

C:\_PythonClass\Mod6\Assignment06>python.exe assignment06.py
******* The current Tasks ToDo are: *******
Wake up (high)
Exercise (medium)
Brush teeth (high)
wash face (high)
*******************************************


        Menu of Options
        1) Add a new Task
        2) Remove an existing Task
        3) Save Data to File
        4) Reload Data from File
        5) Exit Program


Which option would you like to perform? [1 to 5] - 2

Enter task to be removed: wash face
******* The current Tasks ToDo are: *******
Wake up (high)
Exercise (medium)
Brush teeth (high)
*******************************************

Success
Press the [Enter] key to continue.
```

Figure 18: Running the program in the Windows OS console

Summary

In this assignment, we learned to form classes and functions. After listing the global variables to be used in the beginning of the program, we formed one class of functions that would be used for processing and another class of functions used for presenting. We then ran the main script of the program calling on those functions, with very minimal code employed outside of calling those functions. We captured the returns of the functions in variables, which were then passed as arguments into other functions that we called. In this way, we could achieve the division of our program into data, processing, presenting components.