

Jonah Jepson

February 21, 2023

IT FDN 110

Assignment06

<https://jjepson3.github.io/IntroToProg-Python-Mod06/>

Using Functions to Organize Code

Introduction

For this assignment I was asked to take code that was provided by my professor and complete it following the Separation of Concerns (SOC) design principle. According to SOC, code should be separated into sections based on the type of code. For this assignment the code was split into Data, Processing, Presentation, and then the Main Script Body. Data code is for initializing variables, lists, dictionaries etc. Processing code is for performing tasks like reading and writing to a text file and updating lists and dictionaries. Presentation code is for taking inputs from users and displaying. Code in these areas is organized into custom functions and then those functions are called into the main script body when they are needed. This keeps the script compact and easier to use. It also allows for easier debugging. For this code, all the functions were provided and defined by the professor and the main script was largely complete. However, some of the functions were missing code so they didn't do anything. It was my job to add the necessary code to make these functions work (Note: I largely copy and pasted code from solutions to Assignment05). The purpose of the code provided was to allow a user to load data for a to do list from a text file if one already exists or create one if one does not exist. Then the code must be able to add and remove tasks and then save the data back to the text file.

Updating Code

The first thing I did was add my name to the change log. Then I went started going through each section of the code and adding code where required (Note: for each custom function that was using the variables task and priority I was getting a weak error 'Shadow name 'variable' from outerscope' so I changed all instances of task and priority to str_task and str_priority respectively).

Processing Code

The first thing I changed was the function designed to load data from a text file. If the text file doesn't already exist, the code as it was written would produce an error. I changed this by using an if statement to check if the file exist before trying to read the file (See Figure 1).

```
def read_data_from_file(file_name, list_of_rows):
    """ Reads data from a file into a list of dictionary rows

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    list_of_rows.clear() # clear current data
    if os.path.exists(file_name):
        file = open(file_name, "r")
        for line in file:
            str_task, str_priority = line.split(",")
            row = {"Task": str_task.strip(), "Priority": str_priority.strip()}
            list_of_rows.append(row)
        file.close()
    return list_of_rows
```

```
def read_data_from_file(file_name, list_of_rows):
    """ Reads data from a file into a list of dictionary rows

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    list_of_rows.clear() # clear current data
    file = open(file_name, "r")
    for line in file:
        task, priority = line.split(",")
        row = {"Task": task.strip(), "Priority": priority.strip()}
        list_of_rows.append(row)
    file.close()
    return list_of_rows
```

Figure 1: Changed Version (Left), Original (Right)

The next function 'add_data_to_list', was only missing one line. The function adds data to a row but was missing the line of code that adds that row to a list. I added the line `list_of_rows.append(row)` to complete the function. Then I moved on the 'remove_data_from_list' function. This function is designed to remove rows from the table. To make this function work I just pasted code from my answer to assignment 5 (See Figure 2). Then I changed the variables to match the variables in my new code.

```
for row in lstTable:
    if strRemove.lower() == row['Task'].lower():
        lstTable.remove(row)

for row in list_of_rows:
    if str_task.lower() == row["Task"].lower():
        list_of_rows.remove(row)
```

Figure 2: Original (Top), Changed Version (Bottom)

For the 'write_data_to_list' function I did something like the above solution, but I copied the code from the professors answer to assignment 5 instead of mine. I did this because he opens files differently than I usually do and I wanted to keep it consistent throughout the code (See Figure 3).

```
objFile = open(objFileName, "w")
for dicRow in lstTable:
    objFile.write(dicRow["Task"] + "," + dicRow["Priority"] + "\n")
objFile.close()

file = open(file_name, "w")
for row in list_of_rows:
    file.write(row["Task"] + "," + row["Priority"] + "\n")
file.close()
return list_of_rows
```

Figure 3: Original (Top), Changed Version (Bottom)

Presentation Code

For the presentation code there was two functions that needed code written for them: 'input_new_task_and_priority' and 'input_task_to_remove'. I also used code from the professors answer to assignment 5 and pasted it in (See Figures 4 and 5). Like before, I changed the variables to match the new code.

```
# Step 4 - Add a new item to the list/Table
elif strChoice.strip() == '2':
    strTask = str(input("What is the task? - ")).strip()
    strPriority = str(input("What is the priority? [high|low] - ")).strip()

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

    :return: (string, string) with task and str_priority
    """
    str_task = str(input("What is the Task? - ")).strip()
    str_priority = str(input("What is the Priority? [high|low] - ")).strip()
    print() # Print For Spacing
    return str_task, str_priority
```

Figure 4: Original (Top), Changed Version (Bottom)

```
# Step 5 - Remove a new item to the list/Table
elif strChoice == '3':
    # Step 5a - Allow user to indicate which row to delete
    strKeyToRemove = input("Which TASK would you like removed? - ")

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

    :return: (string) with str_task
    """
    str_task = str(input("Which TASK would you like removed? - ")).strip()
    print() # Add an extra line for looks
    return str_task
```

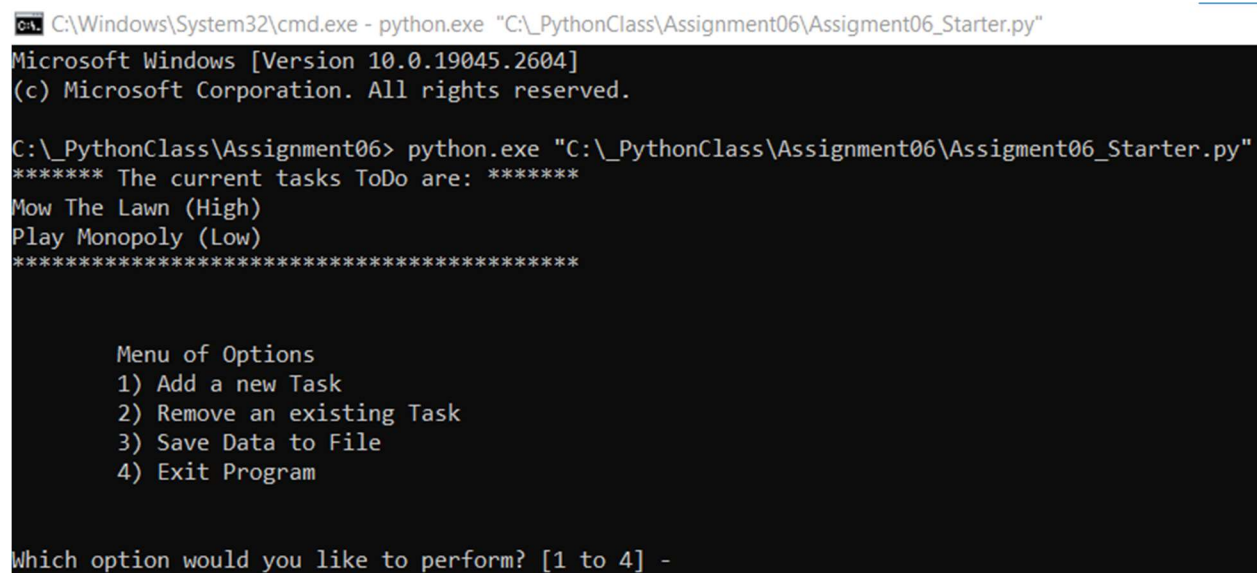
Figure 5: Original (Top), Changed Version (Bottom)

Main Script Body

The main script body didn't need any changes. I did add an else statement to the end of the while loop to show the user an error if they enter something other than 1-4.

Conclusion

With all the code I added, the program now runs like it is supposed to. Proof shown below in figures 6, 7, and 8. This assignment was both easy and difficult. The reason why I found it easy was I didn't really need to write any new code. I just pasted code from other files I already had. The reason I found it difficult was that my other code was not organized using SOC design principles. It was difficult to find the right parts to copy and paste from each block of code since the original code was organized differently.



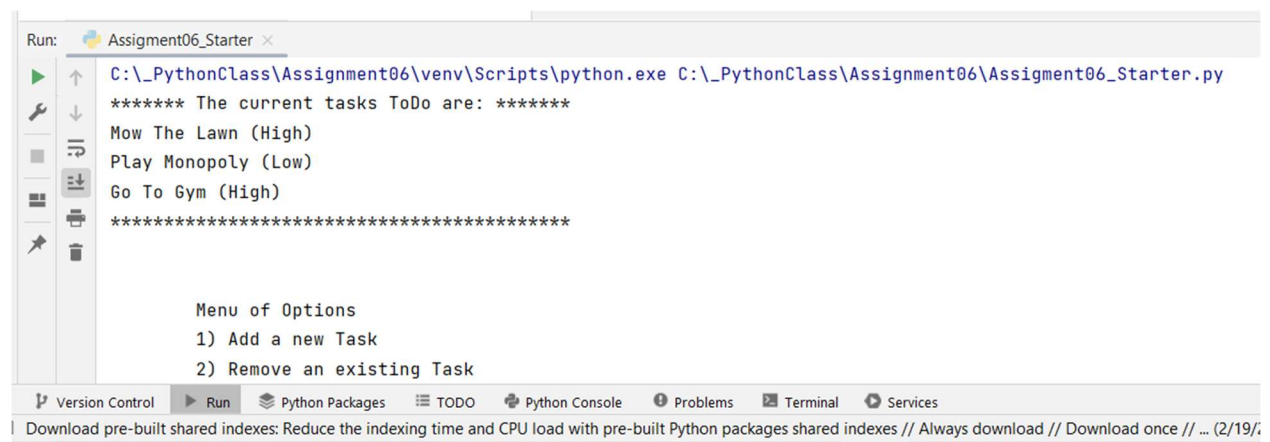
```
C:\Windows\System32\cmd.exe - python.exe "C:\_PythonClass\Assignment06\Assignment06_Starter.py"
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\_PythonClass\Assignment06> python.exe "C:\_PythonClass\Assignment06\Assignment06_Starter.py"
***** The current tasks ToDo are: *****
Mow The Lawn (High)
Play Monopoly (Low)
*****

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

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

Figure 6: Code Running in Command Shell



```
Run: Assignment06_Starter x
C:\_PythonClass\Assignment06\venv\Scripts\python.exe C:\_PythonClass\Assignment06\Assignment06_Starter.py
***** The current tasks ToDo are: *****
Mow The Lawn (High)
Play Monopoly (Low)
Go To Gym (High)
*****

Menu of Options
1) Add a new Task
2) Remove an existing Task
```

Figure 7: Code Running in PyCharm

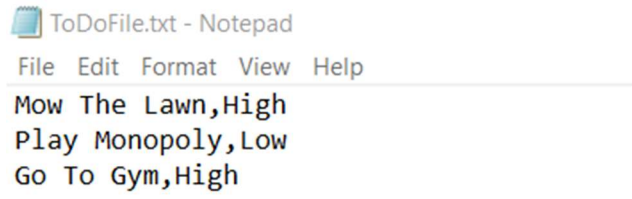


Figure 8: Text File