Rufus Ayeni
November 6, 2020
IT FDN 110 A
Assignment 06
https://github.com/rayeni/IntroToProg-Python-Mod06

Modification of To-Do List Script

Introduction:

The requirements for Assignment 06 are essentially the same as those in Assignment 05-- to modify an existing script that manages a user's "To-Do" list (*Task and Priority*). Assignment 05 introduced a new data structure-- *the dictionary*. Assignment 06 introduced *classes* and *functions*. Functions enable developers to modularize a script in to discrete units of code that perform a single task. Classes are collections of functions. The justification for functions is to make code more readable and easier to modify. I must admit that the use of functions is a bit confusing, because I developed a level of comfort writing (*and reading*) code in a "top-down" format. The modification of the script entails writing functions to perform the same tasks in Assignment 05.

Requirements:

1. Modify an existing Python script by writing functions to perform processing and presentation tasks.

2. The processing tasks are:

    a. Read data from a file
    b. Add data to a list
    c. Remove data from a list
    d. Write data to a file

3. The presentation tasks are:

    a. Print menu tasks
    b. Receive input (menu choice)
    c. Print current tasks in list
    d. Receive input (y or no)
    e. Receive input (to continue)
    f. Receive input (task and priority)
    g. Receive input (task to remove)

4. Modify `if` code blocks to make calls to functions:

Modifications of Python Script:

The following modifications were made to the `class Processor` part of the script:

1. The function `def read_data_from_file()` was modified by adding the `try` and `except` clauses to the existing code block. The clauses were added to suppress errors attributable to the file not being present the first time the script is run:

```
    @staticmethod
    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
        """
        try:
            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, 'Success'
        except:
            print()
```

Figure 1

2. The function `def add_data_to_list()` was modified by adding code that creates a dictionary/row from the parameters `task` and `priority`, and adds the dictionary to the list, `list_of_rows`:

```
    @staticmethod
    def add_data_to_list(task, priority, list_of_rows):
        """ Adds data to dictionary, then adds dictionary to list """

        # add task and priority to dictionary and assign to row
        row = {"Task": task.strip().capitalize(), "Priority": priority.strip().capitalize()}
        # append dict to list_of_rows
        list_of_rows.append(row)
        return list_of_rows, 'Success'
```

Figure 2

3. The `def remove_data_from_list()` function was modified by creating a `status` variable to notify the user of success or failure in removing the `task` record. I encountered a little difficulty in creating the `for` loop. The difficulty occurred when I attempted to use `for i in range(0, len(list_of_rows)`. That condition produced an "`Indexerror: list index out of range`" error. To resolve the error, I changed the loop's condition to `for row in list_of_rows`, and then created a list index variable, `i`, and incremented at the end of the `for` loop block.

```
    @staticmethod
    def remove_data_from_list(task, list_of_rows):
        """ Receives task and lists of tasks (dicts).  Removes task from list.
            Returns updated list.
        """
        status = ""  # Returns confirmation of success or failure in removing task
        i = 0  # list index
        # for loop to check if task exists.  If exists, delete it. Else, notify user.
        for row in list_of_rows:
            if list_of_rows[i]["Task"] == task:
                del list_of_rows[i]
                status = 'Success'
            else:
                status = "Failed: please check spelling or if task exist."
            i += 1
        return list_of_rows, status

```

Figure 3

4. The `def write_data_to_file()` function was modified by adding code that creates/opens a file (*in write mode, "w"*), loops through the list of rows/dictionaries (*by **key***) and writes the **value** to the file. After the `for` loop completes, the script closes the file, and sends the list (`list_of_rows`) and a message (`'Data saved'`) to the main part of the script:

```
    @staticmethod
    def write_data_to_file(file_name, list_of_rows):
        """ Function receives name of file and list of tasks (dicts).
            Writes list elements to file.  Returns list and message.
        """
        # opens file strFileName
        objFile = open(strFileName, "w")
        # for each row/dict in the table, write the values to the file
        for row in lstTable:
            objFile.write(row.get("Task") + "," + row.get("Priority") + "\n")
        objFile.close()  # close the file
        return list_of_rows, 'Data saved'
```

Figure 4

In the `class IO` part of the script, only two (*of seven*) functions required modification:

5. The `def input_new_task_and_priority()` function was modified to get the `task` and `priority` from the user, and to return the two values to the main part of the script (the `while` loop) to be processed:

```
    @staticmethod
    def input_new_task_and_priority():
        """ Ask user for task and its priority.
            Returns task and priority to main part of script
        """
        task = input("Enter Task: ")
        priority = input("Enter Priority (High, Medium, Low): ")
        return task, priority
```

Figure 5

6. The `def input_task_to_remove()` function was modified to accept the `task` the user wants to remove and `return` it to the main part of the script to be processed:

```
    @staticmethod
    def input_task_to_remove():
        """ Ask user for task to remove.
            Returns task to main part of script.
        """
        task = input("Enter Task to Remove: ")
        return task
```

Figure 6

Four modifications were made in the main part of the script, which entails a `while` loop with several `if/elif` statements, representing the user's choice from the main menu of options.

7. The `if` code block (*Add a New Task*) was modified with code that calls functions to receive input (task and priority) from the user, and to add that data to the list:

```
    # Step 4 - Process user's menu choice
    if strChoice.strip() == '1':  # Add a new Task
        # Call input function, capture return in variables for task and priority
        strTask, strPriority = IO.input_new_task_and_priority()
        # Call function to add data to list, capture return in vars for table and status
        lstTable, strStatus = Processor.add_data_to_list(strTask, strPriority, lstTable)
        # Call function request input to continue
        IO.input_press_to_continue(strStatus)
        continue  # to show the menu
```

Figure 7

8. The first `elif` code block (***Remove an existing Task***) was modified to make a function call to `IO.input_task_to_remove()` to get the task to be removed, and another function call to `Processor.remove_data_from_list()` to remove the task from the list:

```
elif strChoice == '2':  # Remove an existing Task
    # Call functions to receive task to remove, remove task, and receive input to continue
    strTask = IO.input_task_to_remove()
    # Call function to add data to list, capture return in vars for table and status
    lstTable, strStatus = Processor.remove_data_from_list(strTask.strip().capitalize(), lstTable)
    # Call function request input to continue
    IO.input_press_to_continue(strStatus)
    continue  # to show the menu
```

Figure 8

9. The second `elif` code block (***Save Data to File***) was modified to make a function call to `IO.input_yes_no_choice()` to get the user's choice (`y/n`). Based on the user's choice, the code makes a function call to either save the data to a file, or to cancel request:

```
elif strChoice == '3':   # Save Data to File
    strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
    if strChoice.lower() == "y":
        # Call functions to write data to file, and receive input to continue.
        lstTable, strStatus = Processor.write_data_to_file(strFileName, lstTable)
        IO.input_press_to_continue(strStatus)
    else:  # Notify user Save operation was canceled.
        IO.input_press_to_continue("Save Cancelled!")
    continue  # to show the menu
```

Figure 9

10. The third `elif` block of code for ***Reload Data from File*** was modified with a nested `if-else` clause. The script makes a function call to the `IO.input_yes_no_choice`, which asks the user if he/she is sure they want to reload data from the file. If the user answers `yes(y)`, the script makes a call to the `Processor.read_data_from_file()` function. If the user answers `no(n)`, then the reload operation is canceled:

```
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':
        # Call functions to read data from file, and receive input to continue
        lstTable, strStatus = Processor.read_data_from_file(strFileName, lstTable)
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("File Reload  Cancelled!")
    continue  # to show the menu
```

Figure 10

Execution of Script:

1. To execute the script in PyCharm, simply right-click the code area and select "**Run 'Assignment06.**'"

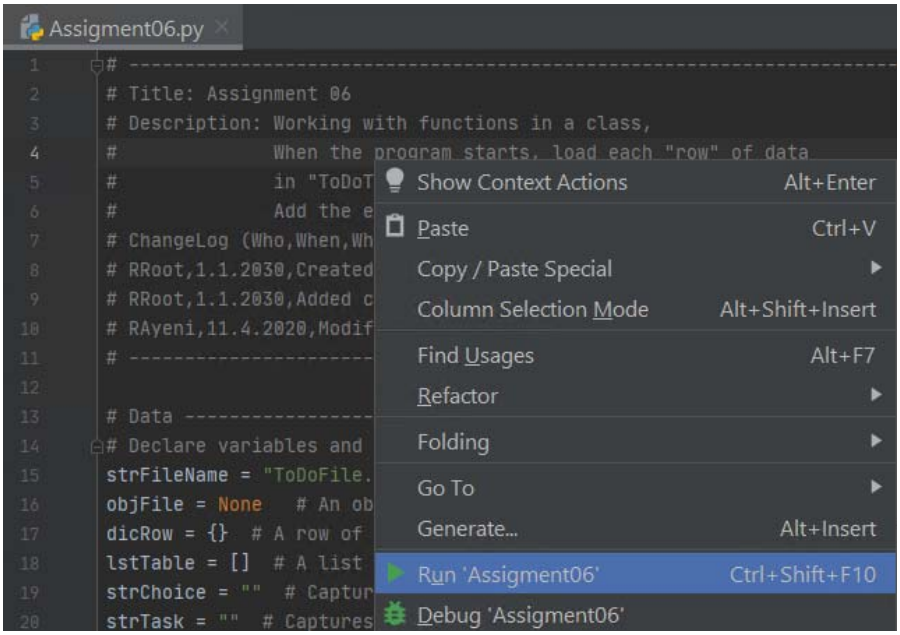

Figure 12

2. Running the script kicks off a command window through which a user can interface with the application, through a menu of options:

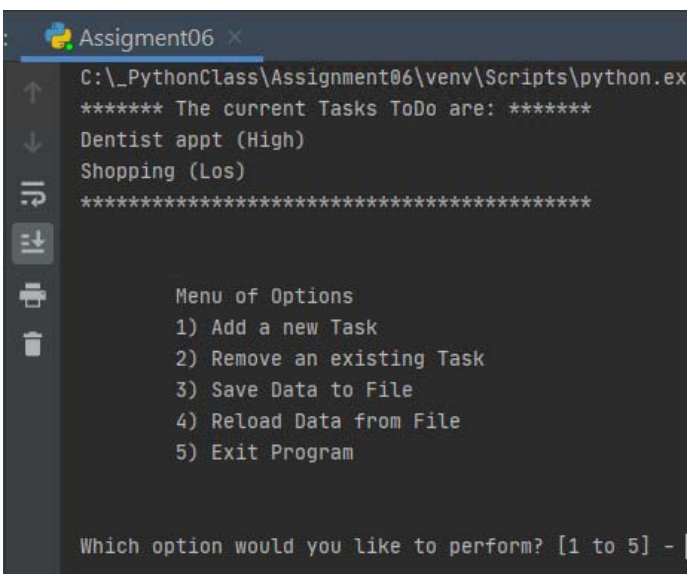

Figure 13

3. **Option #1** (*Add a new Task*), asks the user to enter a task, and its priority:

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

Enter Task: Mow Lawn
Enter Priority (High, Medium, Low): medium
Success
Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dentist appt (High)
Shopping (Los)
Mow lawn (Medium)
*****************************************
```

Figure 14

4. **Option #2** (*Remove an existing Task*), asks the user to enter the task to remove:

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

Enter Task to Remove: Shopping
Success
Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dentist appt (High)
Mow lawn (Medium)
*****************************************
```

Figure 15

5. **Option #3** (*Save Data to File*) asks the user if he/she wants to save data to a file. Here, the user selects `yes(y)`, and receives a confirmation message (*'Data saved'*):

```
Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y
Data saved
Press the [Enter] key to continue.
```

Figure 16

6. **Option #4** (*Reload Data from File*) reloads data from a file. When the application started, the following data was loaded from the `ToDoFile.txt` file:

```
******* The current Tasks ToDo are: *******
Dentist appt (High)
Shopping (Los)
*****************************************
```

Figure 17

Figure 18 confirms that **Option #4** reloaded data from the file, because it's different than the data that was initially loaded from the file in **Figure 17**:

```
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
Success
Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dentist appt (High)
Mow lawn (Medium)
*****************************************
```

Figure 18

7. **Option #5** (*Exit Program*) asks the user to press the `Enter` key to exit the application:

```
        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 19

8. The Project pane shows that the file `ToDoFile.txt` was created:

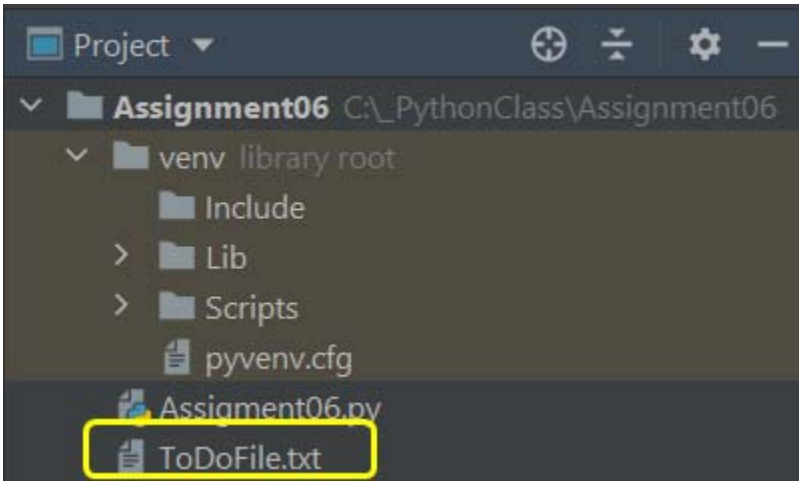

Figure 20

9. When the file is opened, in PyCharm, we can see that the user's entries were written to the file:

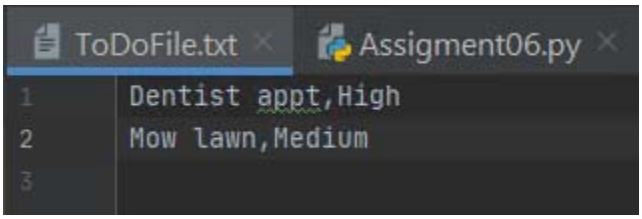

Figure 21

10. **Figure 22** shows the execution of the script in Windows a Command Prompt:

```
Command Prompt

C:\_PythonClass\Assignment06>python Assigment06.py
******* The current Tasks ToDo are: *******
Dentist appt (High)
Mow lawn (Medium)
*******************************************


        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] - 1

Enter Task: Housework
Enter Priority (High, Medium, Low): Low
Success
Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dentist appt (High)
Mow lawn (Medium)
Housework (Low)
*******************************************


        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] - 3

Save this data to file? (y/n) - y
Data saved
Press the [Enter] key to continue.
******* The current Tasks ToDo are: *******
Dentist appt (High)
Mow lawn (Medium)
Housework (Low)
*******************************************


        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!

C:\_PythonClass\Assignment06>_
```

Figure 22

11. **Figure 23** confirms that the data was written to the same file that was created when the application was run previously:

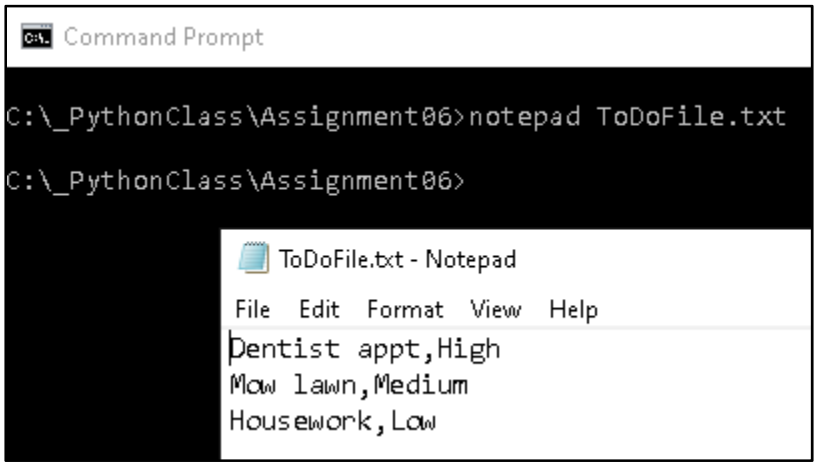

Figure 23

Summary:

In Assignment06, I continued to modify the To-do list Python script. The modifications in this assignment entailed creating functions for individual processing and presentation tasks.