

# To Do List: Classes and Functions

## Introduction

This paper gives an overview of a program designed to manage elements of a “to do” list, including tasks and their corresponding level of priority. The starter code was provided as part of the assignment, and I supplemented the “TODO” sections with my own code. This program displays current tasks in the list to the user and allows them to add tasks to the list, remove tasks from the list, reload the list of tasks from a saved file, and save the list of tasks to that same file, all utilizing the dictionary data type to store the elements of the list.

## Declaring Variables

The first section of the script declares variables to be used in the program (Figure 1).

```
# Data ----- #
# Declare variables and constants
strFileName = "ToDoFile.txt" # The name of the data file
objFile = None # An object that represents a file
dicRow = {} # A row of data separated into elements of a dictionary {Task,Priority}
lstTable = [] # A list that acts as a 'table' of rows
strChoice = "" # Captures the user option selection
strTask = "" # Captures the user task data
strPriority = "" # Captures the user priority data
strStatus = "" # Captures the status of processing functions
```

**Figure 1: Declared variables from the Assignment06\_Starter program**

## Separation of Concerns

This script applies the “Separation of Concerns” pattern to its organization. The script declares a class for processing and a class for presenting input/output, with numerous functions defined within these classes to perform the corresponding actions (Figure 2). Functions from these classes are called in the “main” portion of the script.

```

# Processing ----- #

class Processor:...

# Presentation (Input/Output) ----- #

class IO:...

# Main Body of Script ----- #

# Step 1 - When the program starts, Load data from ToDoFile.txt.

Processor.read_data_from_file(strFileName, lstTable) # read file data

# Step 2 - Display a menu of choices to the user
while(True):...

```

**Figure 2: Layout of Assignment 6 script applying the 'Separation of Concerns' organization**

## Processing: Adding Data to a List

For adding data to the list, I utilized the provided function name *add\_data\_to\_list* and supplemented the code for this function to operate. This function requires the list of tasks as its parameter.

Ideally, the code for input from the user naming the new task and priority to be added would be in the input/output presentation class defined separately, but unfortunately I could not figure out how to make the program run using that organization, so I simply included the input code within my processing class until I am able to improve this separation of concerns.

The task and priority inputs from the user are assigned to a dictionary row and appended to the current list of tasks. The completed function returns this updated list of tasks (Figure 3).

```

@staticmethod
def add_data_to_list(list_of_rows):
    # Code added to complete assignment 6
    """ Adds data to a list of dictionary rows

    :param list_of_rows: (list) you want to add data to:
    :return: (list) of dictionary rows
    """
    print("Enter a task and the level of priority.")
    new_task = input("Task: ")
    new_priority = input("Priority: ")

    row = {"Task": new_task.strip(), "Priority": new_priority.strip()}
    list_of_rows.append(row)
    return list_of_rows, 'Success'

```

**Figure 3: Defining the `add_data_to_list` function within class `Processor`**

## Processing: Removing Data from a List

For removing data from the list, I similarly added code for this function to operate under the already-defined function name `remove_data_from_list`. As with adding data to the list, this function requires the list of tasks as its parameter.

And again, as with adding data to the list, I unfortunately could not figure out how to effectively separate my input code and my processing code, so I included them both within this function to at least allow the program to operate.

The input from the user requesting which task to remove is looped through the list of tasks; if the loop finds a “match” in one of the rows, it removes that task (and its priority) from the list. The completed function returns the updated list of tasks (Figure 4).

```

@staticmethod
def remove_data_from_list(list_of_rows):
    # Code added to complete assignment 6
    """

    :param list_of_rows: (list) you want to remove data from:
    :return: (list) of dictionary rows
    """
    remove_task = input("Task to remove: ")
    for row in list_of_rows:
        if row["Task"].lower() == remove_task.lower():
            list_of_rows.remove(row)
    return list_of_rows, 'Success'

```

**Figure 4: Defining the `remove_data_from_list` function within class `Processor`**

## Processing: Writing Data to a File

For writing the list data to a file, I added code under the defined function *write\_data\_to\_file*. The parameters for this function are the name of the file (to which the data will be written) and the list of tasks (the data to be written).

When this function is called, the *ToDoFile.txt* file is opened and a *for* loop writes each “row” of the list of dictionary items to the file, before the file is closed. The completed function returns the list of tasks, which hasn’t changed, but has now been written to the file (Figure 5).

```
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    # Code added to complete assignment 6
    """
    :param file_name: object representing file you want to write to
    :param list_of_rows: (list) of data you want written to file
    :return: (list) of dictionary rows
    """
    file = open(file_name, 'a')
    for row in list_of_rows:
        file.write(str(row["Task"]) + "," + str(row["Priority"])) + "\n")
    file.close()
    return list_of_rows, 'Success'
```

**Figure 5: Defining *write\_data\_to\_file* function within class *Processor***

## Presentation: Input and Output

Based on the code provided in *Assignment06\_Starter.py*, I could tell that I needed to add code within functions for capturing a couple different inputs from the user. These functions are *input\_new\_task\_and\_priority* and *input\_task\_to\_remove* (Figure 6).

```
@staticmethod
def input_new_task_and_priority(message):
    pass # Code added attempting to complete assignment 6
    new_task = input("Task: ")
    new_priority = input("Priority: ")
    return new_task, new_priority # return task, priority

@staticmethod
def input_task_to_remove():
    pass # Code added attempting to complete assignment 6
    remove_task = input("Task to remove: ")
    return remove_task # return task
```

**Figure 6: Functions within class *IO* for capturing input from the user**

I added code for these functions, returning the corresponding arguments needed to then pass along to the processing functions. However, when calling the input functions and the processing functions separately, I was not able to generate updated tasks in the list. I'm sure the solution is something simple that needs to be tweaked with my arguments and/or parameters. I hope to update my script as soon as I understand the cause for this bug, but for now, I added the keyword *pass* so the program ignores these functions.

## Main

When both the processing and input/output classes and their included functions have “loaded,” the main body of the script begins by loading the data from the `ToDoFile.txt` file (Figure 7).

```
# Step 1 - When the program starts, Load data from ToDoFile.txt.

Processor.read_data_from_file(strFileName, lstTable) # read file data
```

**Figure 7: Loading data from `ToDoFile.txt` at start of program**

A *while* loop is employed to display the current data and the menu of options each time the loop executes (Figure 8).

```
# Step 2 - Display a menu of choices to the user
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 8: while loop for displaying current list of tasks and menu of options**

A series of *if* statements are then used to capture the user's menu choice for the requested action to perform (Figure 9).

```
# Step 4 - Process user's menu choice
if strChoice.strip() == '1':...

elif strChoice == '2':...

elif strChoice == '3':...

elif strChoice == '4':...

elif strChoice == '5':...
```

**Figure 9: Organization of if statements for processing user's menu choice**

If option 1 (add a new task) is selected, the *Processor* class and *add\_data\_to\_list* function are called (Figure 10). This executes the code for capturing input from the user for a new task and its priority and adding that new input to the existing list of tasks. As noted previously, ideally this input code would be in a separate class and function, which would be called prior to the processing class. I will update the program once I learn how to make this operate correctly.

```
# Step 4 - Process user's menu choice
if strChoice.strip() == '1': # Add a new Task
    # Code add to complete assignment 6
    # IO.input_new_task_and_priority() # Commented out
    Processor.add_data_to_list(lstTable)
    IO.input_press_to_continue(strStatus)
    continue # to show the menu
```

**Figure 10: Calling functions to perform user's choice to add a task**

If option 2 (remove an existing task) is selected, the *Processor* class and *remove\_data\_from\_list* function are called (Figure 11). This executes the code for capturing input from the user for which task to remove, and then removing that task from the list. Again, ideally this input code would be in a separate class and function, which would be called prior to the processing class. I will update the program once I learn how to make this operate correctly.

```
elif strChoice == '2': # Remove an existing Task
    # Code added to complete assignment 6
    # IO.input_task_to_remove() # Commented out
    Processor.remove_data_from_list(lstTable)
    IO.input_press_to_continue(strStatus)
    continue # to show the menu
```

**Figure 11: Calling functions to perform user's choice to remove a task**

If option 3 (save data to file) if selected, the *Processor* class and *write\_data\_to\_file* function are called (Figure 12). This executes the code for opening the file and writing the list data to it before closing.

```
elif strChoice == '3': # Save Data to File
    strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
    if strChoice.lower() == "y":
        # Code added to complete assignment 6
        Processor.write_data_to_file(strFileName, lstTable)
        IO.input_press_to_continue(strStatus)
    else:
        IO.input_press_to_continue("Save Cancelled!")
    continue # to show the menu
```

**Figure 12: Calling functions to perform user's choice to save data to file**

If option 4 (reload data from file) is selected, the *Processor* class and *read\_data\_from\_file* function are called (Figure 13). This executes code similar to writing data to the file, but in this case the opposite action is performed, and the existing data in the file is loaded into the program's memory, overwriting any data currently in memory.

```
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")
    if strChoice.lower() == 'y':
        # Code added to complete assignment 6
        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 13: Calling functions to perform user's choice to reload data from file**

When the program is run, it displays the current list of tasks to the user and prints the menu of options to perform. When option '1' is selected, the user is prompted to enter a task and a priority. These are then added to the list, and the updated list of tasks is displayed (Figure 14).

```
***** The current Tasks ToDo are: *****
Do homework (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] - 1

Enter a task and the level of priority.
Task: Wash car
Priority: Low

Press the [Enter] key to continue.

***** The current Tasks ToDo are: *****
Do homework (High)
Wash car (Low)
*****
```

**Figure 14: Option 1 running in Terminal**

When option '2' is selected, the user is prompted to enter the name of a task to remove. This task is then removed from the list, and the updated list of tasks is displayed (Figure 15).

```
***** The current Tasks ToDo are: *****
Do homework (High)
Wash car (Low)
Vacuum (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] - 2

Task to remove: Wash car

Press the [Enter] key to continue.

***** The current Tasks ToDo are: *****
Do homework (High)
Vacuum (Low)
*****
```

**Figure 15: Option 2 running in Terminal**

When option '3' is selected, the user is prompted to enter 'y' or 'n' to save the data to a file. If they select 'y', the list data is written to ToDoFile.txt, and the current list of tasks is displayed (Figure 16). The list has not changed in the program's memory, but the data is now saved.



```

***** The current Tasks ToDo are: *****
Do homework (High)
Vacuum (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

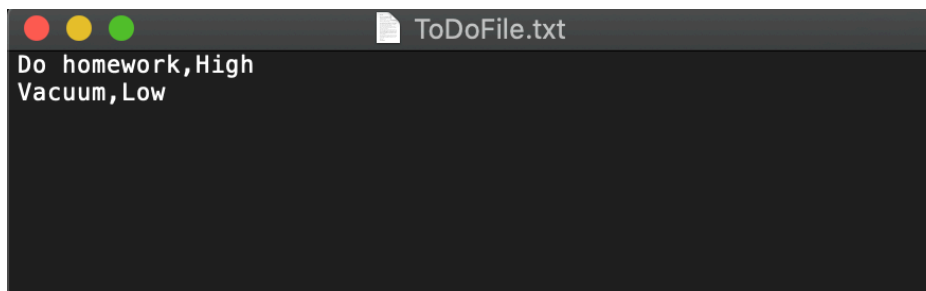
Press the [Enter] key to continue.

***** The current Tasks ToDo are: *****
Do homework (High)
Vacuum (Low)
*****

```

**Figure 16: Option 3 running in Terminal**

To verify that the list data saved, I checked the ToDoFile.txt file, and as expected, the saved data was there (Figure 17).



```

ToDoFile.txt
Do homework,High
Vacuum,Low

```

**Figure 17: Verifying data in ToDoFile.txt**

When option '4' is selected, the user is prompted to enter 'y' or 'n' to reload the data from the file. A warning is displayed, because this means that any tasks in the program's memory that have not been saved to the file will be overridden. If the user selects 'y', the file data is reloaded, and the current list of tasks from the file is displayed (Figure 18).

```

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

Enter a task and the level of priority.
Task: Fold laundry
Priority: High

Press the [Enter] key to continue.

***** The current Tasks ToDo are: *****
Do homework (High)
Vacuum (Low)
Fold laundry (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] - 4

Warning: Unsaved Data Will Be Lost!
Are you sure you want to reload data from file? (y/n) - y

Press the [Enter] key to continue.

***** The current Tasks ToDo are: *****
Do homework (High)
Vacuum (Low)
*****

```

**Figure 18: Option 4 running in Terminal**

When option '5' is selected, the program completes and exits (Figure 19).

```

***** The current Tasks ToDo are: *****
Do homework (High)
Vacuum (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!

```

**Figure 19: Option 5 running in Terminal**

## Summary

This program demonstrates the use of functions to organize a script. Using the principle of the “Separation of Concerns,” processing code and presentation code are defined separately and then called in the main body of the script. This requires an element of organization and understanding how the program jumps around to run various sections of code. Functions are also extremely useful for running the same bit of code in multiple instances.

This program helped me understand the need for classes and functions and how they are utilized in a larger script. The aspect I found challenging was making sure my function parameters corresponded to the provided arguments, in particular when calling more than one function to achieve a task. I look forward to working more with functions to become comfortable with how they can be used to separate sections of a script with different concerns.