

Working With Functions and Classes

Introduction

Last week we saw that code can become repetitive and lengthy. Even with thorough commenting, this can lead to a script that is difficult to follow. Classes and functions¹ can help resolve this by organizing coding tasks and bundling code functionality. Within a class, you can organize code into functions (outside of the main body of the code) that you can then call in the main body of the code.

In Figure 1, you can see some of the code for this assignment organized by class (within the processing and presentation sections) and the main body. These classes contain custom functions. It was our duty to complete the script and fill in the missing code sections.

```
12  | # Data ----- #
13  | # Declare variables and constants
14  | file_name_str = "ToDoFile.txt" # The name of the data file
15  | # file_obj = None # An object that represents a file
16  | # row_dic = {} # A row of data separated into elements of a dictionary {Task,Priority}
17  | table_lst = [] # A list that acts as a 'table' of rows
18  | choice_str = "" # Captures the user option selection
19  | task_str = "" # String argument passed to functions (represents a task)
20
21
22  | # Processing ----- #
23  | class Processor:...
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80  | # Presentation (Input/Output) ----- #
81
82
83  | class IO:...
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149  | # Main Body of Script ----- #
150
151  | # Step 1 - When the program starts, Load data from ToDoFile.txt.
152  | Processor.read_data_from_file(file_name_str, table_lst) # read file data
153
154  | # Step 2 - Display a menu of choices to the user
155  | while (True):...
```

Figure 1. Shows the two classes that are used in this assignment.

¹I like to think of a class as a toolbox and the functions as the tools inside that tool box.

Classes, Functions, and Separation of Concerns

To complete this script², I took code I wrote for the previous assignment (Assignment 05 - Lists and Dictionaries) and re-purposed it. I had to modify it by splitting it into the appropriate processing and input and output functions. This was fairly straight forward with the exception of making sure to adhere to the code sections (keeping processing code in the processing section and any input/output remaining outside of processing section). I also used the debugging tools in PyCharm, albeit briefly as I also made some scratch scripts to test the logic of certain functions.

Listing 1 shows the scratch script that helped me sort out why my 'remove data from list' function was not working properly. Please note that it is not properly commented out as it is a rough, rough draft.

```
# Need to sort out how to remove data from a list
strFileName = "ToDoFile.txt"
lstTable = []

# this function taken from starter script
def read_data_from_file(file_name, list_of_rows) -> object:
    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'

# test logic
def remove_from_list(task, list_of_rows):
    for row in list_of_rows:
        if row["Task"].lower() == task.lower():
            list_of_rows.remove(row)
    print("This task was removed.")
    return list_of_rows

def data_from_user():
    task = str(input("remove this: "))
    return task

read_data_from_file(strFileName, lstTable)
print(lstTable)
strTask = data_from_user()
remove_from_list(strTask, lstTable)
print(lstTable)
```

Listing 1

² We are still working with a to do list in a text file.

Now to break down the code by section, class, and the functions I completed. Firstly, this code is organized with a header, a section for data, another for processing, one for presentation, and then the main body of the script. The data section lists all the variables used in the script. As they exist outside of functions, they are global variables. Snippet 1 shows the data section.

```
# Data ----- #
# Declare variables and constants
file_name_str = "ToDoFile.txt" # The name of the data file
# file_obj = None # An object that represents a file
# row_dic = {} # A row of data separated into elements of a dictionary
# {Task,Priority}
table_lst = [] # A list that acts as a 'table' of rows
choice_str = "" # Captures the user option selection
task_str = "" # String argument passed to functions (represents a task)
```

Snippet 1

The processing section of the script contains a class aptly named, 'Processor.' Within this class there are four functions meant to process data, these are listed below a bulleted list. While the parameters were given to us, for the functioned that are in bold type, code needed to be added to complete the function. I have included the code I used to complete each with a brief description of how that code works. I included function documentation and attempted to replicate the same format as the given code.

Functions in Class called Processor:

```
* read_data_from_file(file_name, list_of_rows)
* add_data_to_list(task, priority, list_of_rows)

def add_data_to_list(task, priority, list_of_rows):
    """ Adds data into a dictionary row and appends that row to a list
    (table)
    :param task: (string) of the new task
    :param priority: (string) of the priority level of the new task
    :param list_of_rows: (list) you want to add the data to
    """
    row = {"Task": task, "Priority": priority}
    list_of_rows.append(row)
    return list_of_rows
```

Snippet 2

The function parameters, task and priority, are as assigned to the appropriate key in the dictionary row. That row is then appended to a list using the parameter, list_of_rows. That parameter will have a list argument passed to it. The list is then returned.

*** remove_data_from_list(task, list_of_rows)**

```
def remove_data_from_list(task, list_of_rows):  
    """ Removes data from a list by searching a list by dictionary rows for  
    the task  
    and removing the task (row) when/if a match is found  
    :param task: (string) task to be removed  
    :param list_of_rows: (list) the list of dictionary rows in which a row  
    will be removed  
    """  
    for row in list_of_rows:  
        if row["Task"].lower() == task.lower(): # Looks in dictionary row  
            using key = Task  
            list_of_rows.remove(row) # removes row from list  
            print("This task was removed.")  
    return list_of_rows
```

Snippet 3

The function uses a *for* loop to iterate through the dictionary 'rows' in a list and looks for text that matches what the user entered. This code is simple and does not check for mistakes or partial data. It does require the user to enter the full text of the dictionary key, "Task." Once the item is removed, a print statement assures the user the function worked and the list is returned.

*** write_data_to_file(file_name, list_of_rows)**

```
def write_data_to_file(file_name, list_of_rows):  
    file = open(file_name, "w")  
    for row in list_of_rows:  
        file.write(row["Task"] + ", " + row["Priority"] + "\n")  
    file.close() # code worked and then added this  
    return list_of_rows
```

Snippet 4

This function opens a file in write mode and then iterates through a list table writing row by row — on a new line — to file (formatted task, priority). It closes the file when it is done and returns the list.

Each of these functions have parameters that will be passed arguments via later portions of the code. That part of the code falls under the presentation section. The class named, IO, contains functions that deal with outputting data (like displaying a menu or the contents of a list) and gathering data from the user (through input statements). Just as before, the functions in bold type were completed

Functions in Class called IO:

```
* output_menu_tasks():  
* input_menu_choice():  
* output_current_tasks_in_list(list_of_rows):  
(list continued on next page)
```

*** input_new_task_and_priority():**

```
def input_new_task_and_priority():
    """ Gets a new task and that task's priority level from the user
    :return: string values for task and priority
    """
    print("Let's add a new task to your To Do list.\n")
    task = str(input("\tPlease enter a task: "))
    priority = str(input("\tPlease enter a priority level [High, Medium,
Low] for this task: "))
    print("The new to do item is, ", task, ", with a priority level of, ",
priority, ".", sep = "")
    print() # Add an extra line for looks
    return task, priority
```

Snippet 6

The user is prompted to enter a new task to their list. Two local variables, task and priority, capture user input (the new task and a priority for that task). For good measure, the newly entered data is repeated back to the user via a print statement.

*** input_task_to_remove():**

```
def input_task_to_remove():
    """ Gets task to remove from list from user
        uses for loop to iterate through rows and look for a string
        matching the text
        the user inputs

        :return: removed task
    """
    task = str(input("Enter the complete text of the task to be removed:
\n"))
    return task
```

Snippet 7

This function asks a user for the text of that task they would like to remove from the list. It then returns the task. That task is then available to be passed as an argument to the processing function that will remove that entry from the list.

All of this comes together in the main body of the code, see Snippet 8. The functions are called and passed arguments. This portion was tricky because while it looks succinct, there is a lot happening line by line.

```

# Main Body of Script
----- #

# Step 1 - When the program starts, Load data from ToDoFile.txt.
Processor.read_data_from_file(file_name_str, 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(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_str, strPriority = IO.input_new_task_and_priority()
        Processor.add_data_to_list(task_str, strPriority, table_lst)
        continue # to show the menu

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

    elif choice_str == '3': # Save Data to File
        Processor.write_data_to_file(file_name_str, table_lst)
        print("Your data was saved.")
        continue # to show the menu

    elif choice_str == '4': # Exit Program
        print("Goodbye!")
        break # and Exit

```

Snippet 8

First, data is read from a text file using our first function³ call. A *while* loop continuously presents menu options, the current contents of the list (as it is or is not manipulated), and asks the user to make a menu choice. *If* and *elif* statements present different functions depending on what the user chooses. Then the various processing and input/output functions are called to create a final script that reads data from a text file, manipulates that data, and saves it to file at a user's choosing.

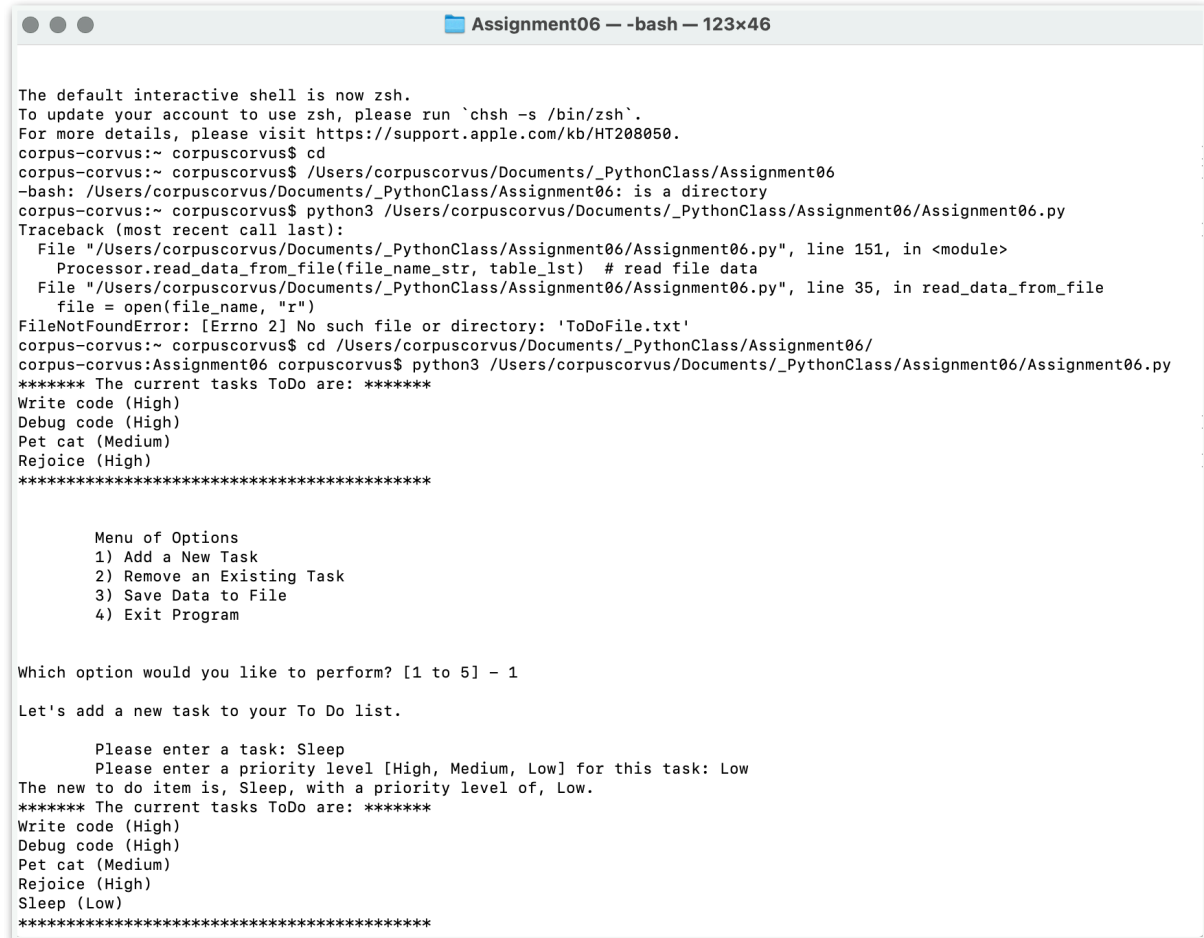
³ You can see the syntax for calling the function is: `class.function(arguments)`.

Final Code

Below you will see screen grabs of my code functioning in both Terminal⁴ and PyCharm.

Terminal

The first time I ran my script in Terminal it did not work because I did not change the directory correctly. I have intentionally left that fumble in so that others could see it (see Figure 2) and learn from my mistake. Once I changed the directory correctly, the code successfully ran⁵ and modified a text file in that same folder location.



```
The default interactive shell is now zsh.
To update your account to use zsh, please run `chsh -s /bin/zsh`.
For more details, please visit https://support.apple.com/kb/HT208050.
corpus-corvus:~ corpuscorvus$ cd
corpus-corvus:~ corpuscorvus$ /Users/corpuscorvus/Documents/_PythonClass/Assignment06
-bash: /Users/corpuscorvus/Documents/_PythonClass/Assignment06: is a directory
corpus-corvus:~ corpuscorvus$ python3 /Users/corpuscorvus/Documents/_PythonClass/Assignment06/Assignment06.py
Traceback (most recent call last):
  File "/Users/corpuscorvus/Documents/_PythonClass/Assignment06/Assignment06.py", line 151, in <module>
    Processor.read_data_from_file(file_name_str, table_lst) # read file data
  File "/Users/corpuscorvus/Documents/_PythonClass/Assignment06/Assignment06.py", line 35, in read_data_from_file
    file = open(file_name, "r")
FileNotFoundError: [Errno 2] No such file or directory: 'ToDoFile.txt'
corpus-corvus:~ corpuscorvus$ cd /Users/corpuscorvus/Documents/_PythonClass/Assignment06/
corpus-corvus:Assignment06 corpuscorvus$ python3 /Users/corpuscorvus/Documents/_PythonClass/Assignment06/Assignment06.py
***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Pet cat (Medium)
Rejoice (High)
*****
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 5] - 1

Let's add a new task to your To Do list.

Please enter a task: Sleep
Please enter a priority level [High, Medium, Low] for this task: Low
The new to do item is, Sleep, with a priority level of, Low.
***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Pet cat (Medium)
Rejoice (High)
Sleep (Low)
*****
```

Figure 2. A screen shot of my coding running in Terminal with my error — incorrectly changing the directory. You can also see the code running successfully.

⁴ I am programming on a Mac. For those that are unfamiliar with MacOS, the Terminal is where you can run your programs in a command line environment.

⁵ If you look closely you can see typo from one of the print statements. It asks the user to make a selection from the menu (1 - 5) but there are only 4 options in the menu. I corrected this in the code and the correction can be seen in the PyCharm screen shots in that section.

```

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 5] - 2

Enter the complete text of the task to be removed: Rejoice
This task was removed.
***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Pet cat (Medium)
Sleep (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 5] - 3

Your data was saved.
***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Pet cat (Medium)
Sleep (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 5] - 4

Goodbye!
corpus-corvus:Assignment06 corpuscorvus$ █

```

Figure 3. A continuation of the script running in Terminal.

PyCharm

Figures 5 and 5 show the code functioning in PyCharm. You can also see the preview of the text file in Figure 6.


```

Project: Assignment06
Assignment06.py
ToDoFile.txt
IO > output_current_tasks_in_list()

Run: Assignment06
/Users/corpuscorvus/Documents/_PythonClass/Assignment06/venv/bin/python /Use
***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Sleep (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] - 1

Let's add a new task to your To Do list.

Please enter a task: Make coffee
Please enter a priority level [High, Medium, Low] for this task: Medium
The new to do item is, Make coffee, with a priority level of, Medium.

***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Sleep (Low)

```

Figure 4. The program running in PyCharm.

```

Assignment06
***** The current tasks ToDo are: *****
Write code (High)
Debug code (High)
Sleep (Low)
Make coffee (Medium)
*****

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] - 2

Enter the complete text of the task to be removed:
Write code
This task was removed.
***** The current tasks ToDo are: *****
Debug code (High)
Sleep (Low)
Make coffee (Medium)
*****

Assignment06
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] - 3

Your data was saved.
***** The current tasks ToDo are: *****
Debug code (High)
Sleep (Low)
Make coffee (Medium)
*****

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] - 4

Goodbye!

```

Figure 5. The continuation and conclusion of running the script in PyCharm.

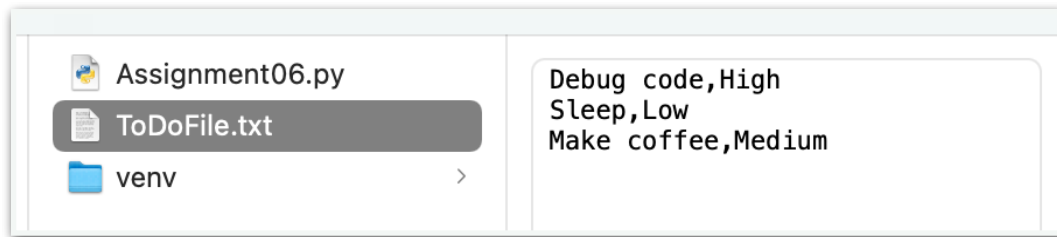


Figure 6. My text file, as previewed in Finder, neatly organized within my assignment folder.

Summary

What made this so challenging was not slipping into coding 'stuff' that didn't belong in that particular section. As it stands, I did slip in putting a print statement in one of my processing functions but as I am new to this — perhaps I can be pardoned. The other troublesome spot for me was the adding and removing data functions where the input is captured in a different function than the functions that process the input. I couldn't figure out how to get variables assigned to the input data I was capturing from the user into the processing functions as arguments. I was able to write scratch code that helped me sort this but the truly helpful, AHA!, moment was reviewing the lecture materials (timestamp 52:57). Randal showed us code that drove it home for me. I needed to capture user data in variables that I could pass to the input function this skeletonized way: `variable = input_function()`. Then I could use that variable as an argument in the processing function. Figure 7, shows the helpful code.

```
# test new data input
strItem, strValue = IO.input_item_and_value()
print(strItem, strValue)

# test remove data input
strItem = IO.input_item_to_remove()
print(strItem)
```

Figure 7. Code that helped me out of my insanity loop of trying the same thing over and over and expecting different results.