Jon Larson

08-17-2022

Foundations of Programming: Python

Assignment 06

https://github.com/jtlarson/IntroToProg-Python-Mod06

Using Classes and Functions

Introduction

This paper will demonstrate the use of classes and functions to help achieve the "separation of concerns." Like the previous assignment, this will involve modification of an existing script to complete the desired functionality. We will look at the existing code structure and why it's useful to segment code into functions and classes. The description of the coding process will be divided into sections covering the development of different functional elements of the completed program. The goal is to add missing code within the function blocks to allow the program to load data from a file, display and modify the data, and write it back to the file if desired. And of course, we will show samples of the running code at the conclusion of this document.

Existing Code Structure

The assignment requires that we start with an existing code template and fill in marked "TODO" sections with additional code to complete the functionality described in the pseudo code. The code template begins with a header and a 'data' section (where variables are declared (Figure 1):

```
# ------------------------------------------------------------------------ #
# Title: Assignment 06
# Description: Working with functions in a class,
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.txt" into a python Dictionary.
#              Add each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# jtlarson,8.10.2022, Modified code to complete assignment 06
# jtlarson,8.15.2022, Edited variable names to avoid global shadowing, refined function header descriptions
# ------------------------------------------------------------------------ #

# 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
```

Figure 1 - Header and "Data" sections

As mentioned in the introduction, this template separates many of the operational code elements into classes and functions which follow the principle of "separation of concerns." The idea behind "separation of concerns" is to identify discrete code segments by the type of work each segment performs, which helps make the complete program easier to understand and maintain. Along these lines, this program has a "Processor" class (read/write data from lists and files) a "IO" class (input/output to user), and the "Main body" of the script, which calls the class functions as needed (Figure 2):

```
20
21    # Processing -------------------------------------------------------- #
22    class Processor:...
83
84    # Presentation (Input/Output) --------------------------------------- #
85
86    class IO:...
153
154   # Main Body of Script ----------------------------------------------- #
155
156   # Step 1 - When the program starts, Load data from ToDoFile.txt.
157   Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst)  # read file data
158
159   # Step 2 - Display a menu of choices to the user
160   while True:...
185
```

Figure 2 - Processing, presentation and main body sections

Code Explanation

The "TODO" segments in the supplied template are all contained within the class functions, However, I chose to make some small modifications outside the "TODO" areas to improve several aspects of the template code:

- I changed variable names used in functions that match (shadow) global variables. This doesn't necessarily keep the program from working, but it can lead to unnecessary confusion.
- In the previous assignment, I employed index numbers to identify each row of the table, since it made it easier (vs typing out the whole task name) for the user to correctly identify a row to be deleted. Since this template is doing essentially the same task, but with functions (which are designed to promote the re-use of established code), I thought it made sense to make the modifications necessary to reuse my methods which provide a better experience for the user.
- Several of the function headers contained inaccurate descriptions of the parameters, so I improved the descriptions along with factual updates required by my other changes.

Below I will describe the changes made to each function and the requisite changes to variables within the "Main body" of the script.

The Processor Class

The first function in the "Processor" class is "read_data_from_file". Since there was no 'TODO' section in this class, my modifications were limited to renaming variables "task" and "priority" as "a_task" and "a_priority" respectively.

The "Processor. add_data_to_list" function requires that we add code to append a (dictionary) row containing 'name' and 'priority' into an existing "list_of_rows." This can be done with a single line of added code, as seen on line 53 (Figure 3).

```
43      def add_data_to_list(task_name, task_priority, list_of_rows):
44          """ Adds data to a list of dictionary rows
45
46          :param task_name: (string) with name of task
47          :param task_priority: (string) with name of priority
48          :param list_of_rows: (list) table list where row will be appended
49          :return: (list) of dictionary rows
50          """
51          row = {"Task": task_name, "Priority": task_priority}
52          # Append row dictionary to list of rows (table)
53          list_of_rows.append(row)
54          return list_of_rows
```

Figure 3 - "Processor.add_data_to_list" function

The "Processor. remove_data_from_list" function requires that we add code to remove a (dictionary) row from the "list_of_rows." As mentioned previously, I chose to edit this function to work with an integer representing the index of the row to be deleted ("row_id" parameter) instead of expecting a task name that is an exact match of an existing row. I encased the removal code in a 'if' conditional to avoid program termination if the user entered a row number that isn't within a valid range (Figure 4).

```
56      @staticmethod
57      def remove_data_from_list(row_id, list_of_rows):
58          """ Removes data from a list of dictionary rows
59
60          :param row_id: (int) index of row to be deleted
61          :param list_of_rows: (list) table list containing row to be deleted
62          :return: (list) of dictionary rows
63          """
64          # Check if the selected index is valid before deleting to avoid errors
65          if row_id in range(0, len(list_of_rows)):
66              list_of_rows.pop(row_id)
67          return list_of_rows
```

Figure 4 - "Processor.remove_data_from_list" function

The last function in the “Processor” class is the “write_data_to_file” function. The code I added opens the file identified in the “file_name” parameter in ‘write’ mode, writes the key and value of each dictionary row to the file in CSV format, then closes the file again before returning the “list_of_rows” (Figure 5).

```
70      def write_data_to_file(file_name, list_of_rows):
71          """ Writes data from a list of dictionary rows to a File
72
73          :param file_name: (string) with name of file
74          :param list_of_rows: (list) to write to file
75          :return: (list) of dictionary rows
76          """
77          # Open the file and write each row of data in CSV format
78          file = open(file_name, "w")
79          for row in list_of_rows:
80              file.write(row["Task"] + "," + row["Priority"] + "\n")
81          file.close()
82          return list_of_rows
```

Figure 5 - “Processor.write_data_to_file" function

The IO Class

The “IO” class contains function directed at input and output to the user—i.e. what the user sees while the program is running. Accordingly, the first function in this class (“output_menu_tasks”) simply displays a pre-formatted menu in a triple-quoted string. The second function (“input_menu_choice”) prompts the user to make a selection from the menu and returns that “choice.” Since there’s no “TODO” or other changes to either of these functions, I won’t bother with a separate screenshot.

I re-used code from assignment 5 to enhance the third function (Figure 6 - “IO. output_current_tasks_in_list”) to provide a dynamically-formatted list with an index column on the left. This code scans the length of all the task names to determine the column spacing, and provides the user with an easy means to select a record for deletion if they choose that option from the menu.

```python
def output_current_tasks_in_list(list_of_rows):
    """ Shows the current Tasks in the list of dictionaries rows

    :param list_of_rows: (list) of rows you want to display
    :return: nothing
    """
    max_task_chars_int = 0  # the length of the task name
    # Find the longest task name
    for row in list_of_rows:
        if len(row['Task']) > max_task_chars_int:
            max_task_chars_int = len(row['Task'])
    # Display indexed list of records
    row_count_int = 0
    print("ID:", "Task", " " * (max_task_chars_int - 4), "|", "Priority")
    for row in list_of_rows:
        print(row_count_int, ":", row['Task'], " " * (max_task_chars_int - len(row['Task'])), "|", row['Priority'])
        row_count_int += 1
```

Figure 6 - "IO. output_current_tasks_in_list" function

The "IO. input_new_task_and_priority" function asks the user to input a task and priority, and returns these strings. If an empty string is allowed to be stored and saved to the output file, it could make the file invalid for future use, so I chose to pre-declare the argument defaults so that I could use a 'while' loop to check that the user entered a string for both before continuing (Figure 7).

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

        :return: (string, string) with task and priority
        """
        task = None
        priority = None
        # If either task or priority is missing, ask again...
        while not task or not priority:
            task = str(input("What task do you want to add?: ").strip())
            priority = str(input("What priority is this task?: ").strip())  # store as a string
        return task, priority
```

Figure 7 - "IO.input_new_task_and_priority" function

The final "IO" class function prompts the user for which task to delete. While the template assumes this will be a string, I chose to change it to an integer (Figure 8) in keeping with the changes I mentioned earlier.

```python
    @staticmethod
    def input_task_to_remove():
        """ Gets the task name to be removed from the list

        :return: (int) with task
        """
        task_id_int = int(input("Enter the ID to remove: ").strip())
        return task_id_int
```

Figure 8 - "IO.input_task_to_remove" function

Main Body

The "Main Body of Script" doesn't contain any "TODO" sections, so I left it (mostly) untouched. The only edits I made in this section were to variable names. I did this for two reasons: to reflect changes to how a variable is used (e.g. on row 176 I changed "task" to "row_id"). I also changed some variables to clarify purpose and avoid "shadowing" issues that I mentioned previously. The screenshot below (Figure 9) contains a couple examples of where I made these replacements:

```python
# Step 2 - Display a menu of choices to the user
while True:
    # Step 3 Show current data
    IO.output_current_tasks_in_list(list_of_rows=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, priority_str = IO.input_new_task_and_priority()
        table_lst = Processor.add_data_to_list(task_name=task_str, task_priority=priority_str, list_of_rows=table_lst)
        continue  # to show the menu

    elif choice_str == '2':  # Remove an existing task based on row index
        task_id = IO.input_task_to_remove()
        table_lst = Processor.remove_data_from_list(row_id=task_id, list_of_rows=table_lst)
        continue  # to show the menu
```

Figure 9 - "Main Body" sample with edited variable names

Running the code

Below are screenshots of the program working within PowerShell. To start, Figure 10 shows the initial menu and display of current data, as well as entering a new entry:

```
PS C:\Users\jtlarson\OneDrive - UW\_PythonClass\Module06- Functions Updated\Assignment> python .\A06-jtlarson.py
ID: Task          | Priority
0 : read chapter  | 1
1 : watch video   | 2
2 : write code    | 3
3 : upload        | 4

        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

What task do you want to add?: git
What priority is this task?: 5
ID: Task          | Priority
0 : read chapter  | 1
1 : watch video   | 2
2 : write code    | 3
3 : upload        | 4
4 : git           | 5
```

Figure 10 - Running and adding data in PowerShell

I then choose to delete the entry identified by index ID 4, and confirm it is removed (Figure 11):

```
        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 ID to remove: 4
ID: Task          | Priority
0 : read chapter  | 1
1 : watch video   | 2
2 : write code    | 3
3 : upload        | 4
```

Figure 11 - Deleting a row by ID

Finally I save the file, exit the program, and confirm the text is written to the file (Figure 12):

```
Which option would you like to perform? [1 to 4] - 3

Data Saved!
ID: Task          | Priority
0 : read chapter  | 1
1 : watch video   | 2
2 : write code    | 3
3 : upload        | 4

        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!
PS C:\Users\jtlarson\OneDrive - UW\_PythonClass\Module06- Functions Updated\Assignment> cat .\ToDoFile.txt
read chapter,1
watch video,2
write code,3
upload,4
```

Figure 12 - Saving, exiting, and confirming file contents in PowerShell

Running in PyCharm

The output is identical in PyCharm, but to avoid too much duplication, I'll use PyCharm to show how the "IO. input_new_task_and_priority" function keeps asking for input until both task and priority are filled in (Figure 13):

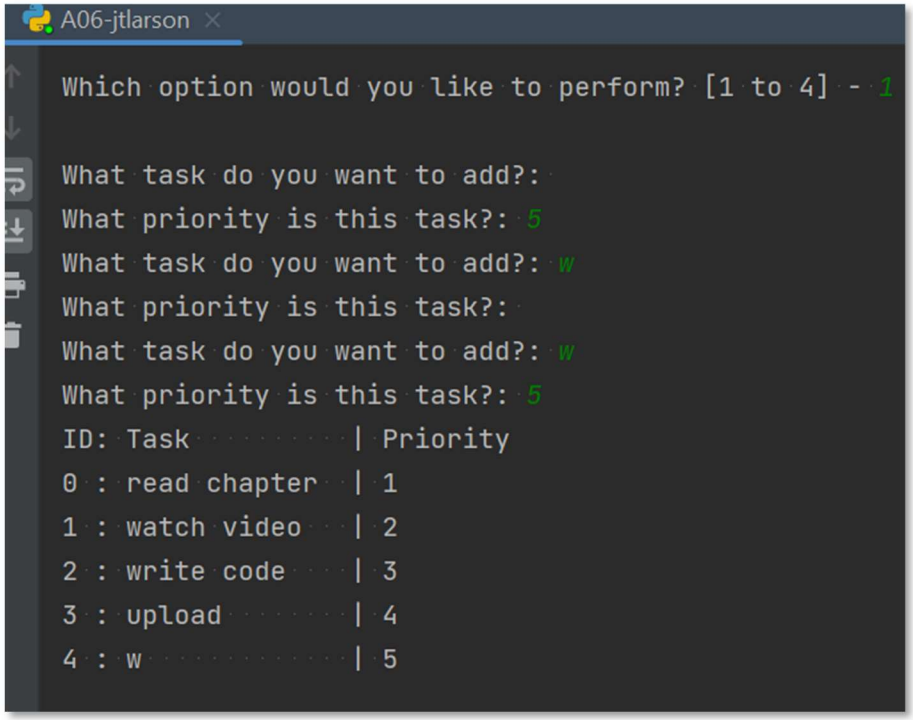

Figure 13 - Input requires both task and priority - running in PyCharm

Summary

In this document I described the structure of the pre-existing code and how it reflects the "separation of concerns." I described some changes I chose to make (and the reasoning behind those changes) to the template code. I also described the code sections with a focus on the functions containing custom code used to complete the assignment. I mentioned a couple areas where I used conditionals to avoid invalid inputs and demonstrated the program operation (as well as proper output to the text file) in PowerShell and PyCharm.