

Steven Simpers

May 23, 2023

Foundations of Programming: Python

Assignment 06

<https://github.com/ssimpers/IntroToProg-Python-Mod06.git> (External Site)

To-Do List Script

Introduction

In this week's assignment, we started working with functions organized within classes. The lesson taught about how parameters and arguments are used by functions. It also explained the differences between global and local variables.

Problem Statement

The assignment's problem statement is to update the provided starter code to allow the user to add to, remove from, and save the to-do list tasks. The starter code addresses separations of concerns by organizing the code into general actions such as data, processing, input/output, main body statements. Processing and input/output functions are both defined within their own class.

Processor Class: "Add Data to List" Function

The "add_data_to_list" function in Listing 1 is the first function that required updating. This function is defined within the processor class and uses three parameters: task, priority, and list_of_rows. A description of the function is provided in the document header (lines 44 through 49) along with helpful information about the parameters used and the data returned out of the function. The first statement in the function uses the task and priority parameters to define a dictionary in a local variable. This local variable is only accessible within the function. The dictionary row is then appended to list_of_rows and it is returned out of the function; allowing it to be captured in a variable when the function is used.

Listing 1: "Add Data to List" Function

```
42     @staticmethod
43     def add_data_to_list(task, priority, list_of_rows):
44         """ Adds data to a list of dictionary rows
45
46         :param task: (string) with name of task:
47         :param priority: (string) with name of priority:
48         :param list_of_rows: (list) you want to add more data to:
49         :return: (list) of dictionary rows
50         """
51         row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
52         list_of_rows.append(row)
53         return list_of_rows
```

Processor Class: “Remove Data From List” Function

The “remove_data_from_list” function in Listing 2 is also in the processor class. Similar to the previous function, it requires task and list_of_rows parameters however this information is used to loop through each row of the list and remove any row that contains a value matching the task parameter. A Boolean flag on lines 63 and 68 is tested in an “if” statement on line 69 to assign a two possible strings to message_status local variable depending on if the Boolean flag is true or false. The updated list_of_rows and message_status are returned. The message_status was added so that the main body of the script can capture this status in a string variable and print it to the user.

Listing 2: “Remove Data From List” Function

```
55     @staticmethod
56     def remove_data_from_list(task, list_of_rows):
57         """ Removes data from a list of dictionary rows
58
59         :param task: (string) with name of task:
60         :param list_of_rows: (list) you want filled with file data:
61         :return: (list) of dictionary rows
62         """
63         status = False
64         for row in list_of_rows:
65             row_task, row_priority = dict(row).values()
66             if row_task.lower() == task.lower():
67                 list_of_rows.remove(row)
68                 status = True
69         if status is True:
70             message_status = "\n\tTask " + task + " was removed from the to-do list!"
71         else:
72             message_status = "\n\tTask " + task + " was not found on the to-do list!"
73         return list_of_rows, message_status
```

Processor Class: “Write Data to File” Function

The “write_data_to_file” function in Listing 3 is in the processor class and uses a filename parameter in addition to the previously used list_of_rows parameter. This function opens the file in write mode and loops through each line of the list, writing only the values from the task and priority keys separated by a comma. The function returns the list_of_rows however I’m realizing now that I don’t think this is necessary as long as the statement in the main body of the script (line 178) doesn’t assign returned data to a variable.

Listing 3: “Write Data to File” Function

```
75     @staticmethod
76     def write_data_to_file(file_name, list_of_rows):
77         """ Writes data from a list of dictionary rows to a File
78
79         :param file_name: (string) with name of file:
80         :param list_of_rows: (list) you want filled with file data:
81         :return: (list) of dictionary rows
82         """
83         objFile = open(file_name, "w") # open file in write mode
84         for row in list_of_rows:
85             objFile.write(row["Task"] + "," + row["Priority"] + "\n")
86         objFile.close()
87         return list_of_rows
```

IO Class: “Input New Task and Priority” Function

The “input_new_task_and_priority” function in Listing 4 is in the input/output class and has no parameters to accept information. However, after prompting the user for a task name and priority, the function returns the task and priority strings.

Listing 4: “Input New Task and Priority” Function

```
131     @staticmethod
132     def input_new_task_and_priority():
133         """ Gets task and priority values to be added to the list
134
135         :return: (string, string) with task and priority
136         """
137         task = str(input(" Enter task name: ")).strip()
138         priority = str(input(" Enter priority: ")).strip()
139         return task, priority
```

IO Class: “Input Task to Remove” Function

The “input_task_to_remove” function in Listing 5 is in the IO class. It doesn’t accept information but it prompts the user for a task to remove and returns that string.

Listing 5: “Input Task to Remove” Function

```
141     @staticmethod
142     def input_task_to_remove():
143         """ Gets the task name to be removed from the list
144
145         :return: (string) with task
146         """
147         task = str(input(" Enter a task name to remove: ")).strip()
148         return task
```

Main Body of Script

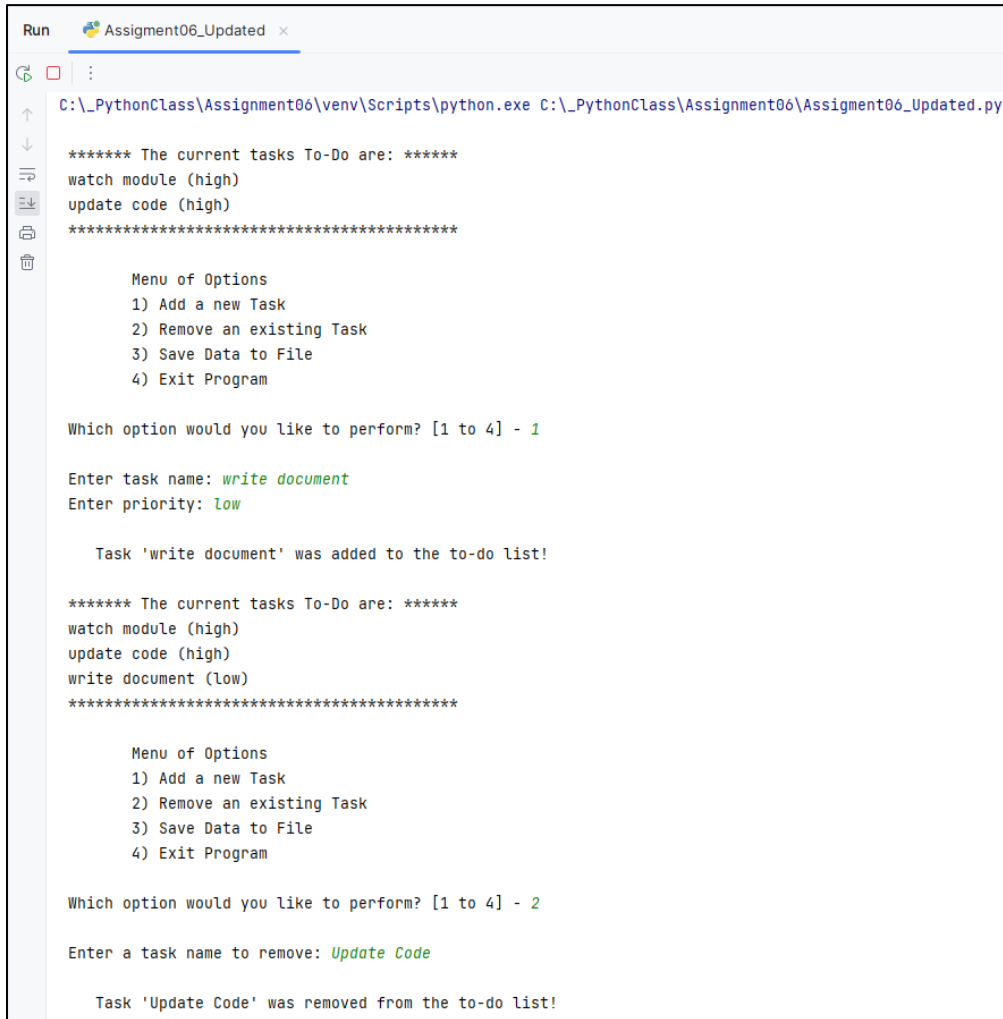
Listing 6 shows the main body of the script which applies abstraction by simplifying the code in which most statements only execute functions. The function statements are written in the form of: `class.function(parameter=argument)`. The class name is defined, followed by the function name. Arguments are passed through the parameter to the function. Most of the main body statements were included in the starter code, however, lines 168, 173, and 174 were added or updated to provide confirmations to the user regarding the status of actions.

Listing 6: Main Body of Script

```
154 # Step 1 - When the program starts, Load data from ToDoFile.txt.
155 Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst) # read file data
156
157 # Step 2 - Display a menu of choices to the user
158 while (True):
159     # Step 3 Show current data
160     IO.output_current_tasks_in_list(list_of_rows=table_lst) # Show current data in the list/table
161     IO.output_menu_tasks() # Shows menu
162     choice_str = IO.input_menu_choice() # Get menu option
163
164     # Step 4 - Process user's menu choice
165     if choice_str.strip() == '1': # Add a new Task
166         task, priority = IO.input_new_task_and_priority()
167         table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
168         print("\n\tTask '" + task + "' was added to the to-do list!")
169         continue # to show the menu
170
171     elif choice_str == '2': # Remove an existing Task
172         task = IO.input_task_to_remove()
173         table_lst, status_str = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
174         print(status_str)
175         continue # to show the menu
176
177     elif choice_str == '3': # Save Data to File
178         table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
179         print("\tData Saved!")
180         continue # to show the menu
181
182     elif choice_str == '4': # Exit Program
183         print("\tGoodbye!")
184         break # by exiting loop
```

Results

An example of the assignment script running in PyCharm is provided below in Figure 1. The user submits menu option 1, is prompted to enter a task and priority to add to the list, and receives confirmation that their task was added to the list. The user chooses menu option 2 next, is prompted to enter a task to remove, and receives confirmation that the task was removed from the list.



```
Run Assignment06_Updated x
C:\_PythonClass\Assignment06\venv\Scripts\python.exe C:\_PythonClass\Assignment06\Assignment06_Updated.py

***** The current tasks To-Do are: *****
watch module (high)
update code (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 4] - 1

Enter task name: write document
Enter priority: low

Task 'write document' was added to the to-do list!

***** The current tasks To-Do are: *****
watch module (high)
update code (high)
write document (low)
*****

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

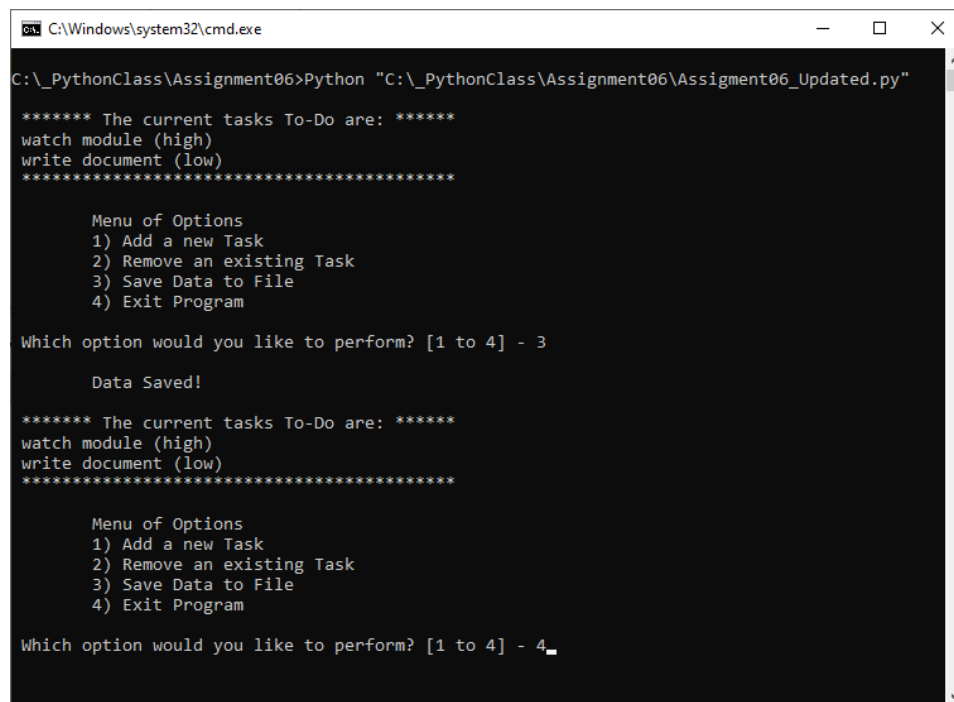
Which option would you like to perform? [1 to 4] - 2

Enter a task name to remove: Update Code

Task 'Update Code' was removed from the to-do list!
```

Figure 1: "Assignment06_Updated.py" Python script running in PyCharm

Figure 2 below shows the script running in the command window. The user submits menu option 3 and receives confirmation that the data was saved. Lastly, the user picks menu option 4 and the program exits right after the user presses “enter”.



```
C:\Windows\system32\cmd.exe
C:\_PythonClass\Assignment06>Python "C:\_PythonClass\Assignment06\Assignment06_Updated.py"

***** The current tasks To-Do are: *****
watch module (high)
write document (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] - 3

Data Saved!

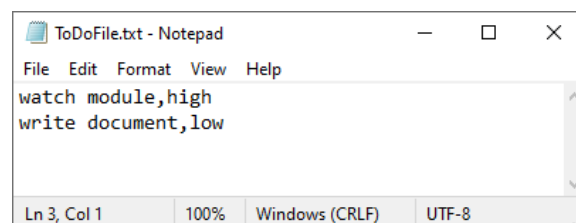
***** The current tasks To-Do are: *****
watch module (high)
write document (low)
*****

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] - 4_
```

Figure 2: “Assignment06_Updated.py” Python script running in Command Window

The data file “ToDoFile.txt” in Figure 3 shows the data stored as comma separated values.



```
ToDoFile.txt - Notepad
File Edit Format View Help
watch module,high
write document,low
Ln 3, Col 1 100% Windows (CRLF) UTF-8
```

Figure 3: Data Saved in “ToDoFile.txt” File

Summary

The assignment developed skills working with functions and classes while introducing more programming foundations such as debugging with breakpoints.