

Naomi Martin

Assignment 06

Aug 17, 2022

GitHub Webpage: <https://naomixmartin.github.io/IntroToProg-Python-Mod06/>

A Modified To-Do List Program

Introduction

In this assignment, we were tasked with modifying the To-Do List Program from assignment 5 by incorporating functions to clean up and simplify the main body of the code. This assignment allowed us to build off of and strengthen our previous knowledge of loops, lists, and dictionaries, and become familiar with the basic mechanics, syntax, and utility of functions. In this paper, I will describe my approach to this assignment, and explain the logic of certain lines of code.

Starting Out

To start this assignment, I first read through the code to see what had to be done. It was helpful for me to collapse the lines of code for each function, to see the general idea of what Professor Root had intended. I saw that he first defined the functions corresponding to data processing and presentation, and then executed all of these functions in the very short main body of the script. Collapsing the lines of code allowed me to obtain an organized and intuitive understanding of the program. Next, I looked at the lines of code for each function, and identified which areas I needed to modify, and which functions were able to be left alone. I created a table with this information (table 1).

Table 1. Functions I Need To Change, and Functions I Do Not Need To Change

Functions to change	Functions to not change
add_data_to_list()	read_data_from_file()
remove_data_from_list()	output_menu_tasks()
write_data_to_file()	input_menu_choice()
input_new_task_and_priority()	output_current_tasks_in_list()
input_task_to_remove()	

Once I obtained a general idea of the program, I updated the change log with my name and date, and began the assignment.

Step 1: Adding Data to List

The approach I took to completing this assignment was to adjust the code for each menu option in descending order; hence, I started by updating the code that allowed the user to add data to the list. The two functions involved in adding data to the list are

input_new_task_and_priority() and add_data_to_list(). I tackled the code for the former function first, following the logical progression of data input then processing.

The input_new_task_and_priority() function is intended to receive user input for a new task and its priority, so this function was easy enough to write with simple input() function (figure 1). I used local variables taskChoice_str and priorityChoice_str to hold the user-inputted information. I created these variables so as to not confuse myself with the parameters “task” and “priority”. These variables would eventually be unpacked from a tuple and assigned the variables “task” and “priority” in the main body of the text.

```
@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
    """
    print("\n2) Add a new Task\n")
    taskChoice_str = input("Enter a task: ")
    priorityChoice_str = input("Enter your task's priority level: ")
    pass
```

Figure 1. Code to fill in the input_new_task_and_priority() function.

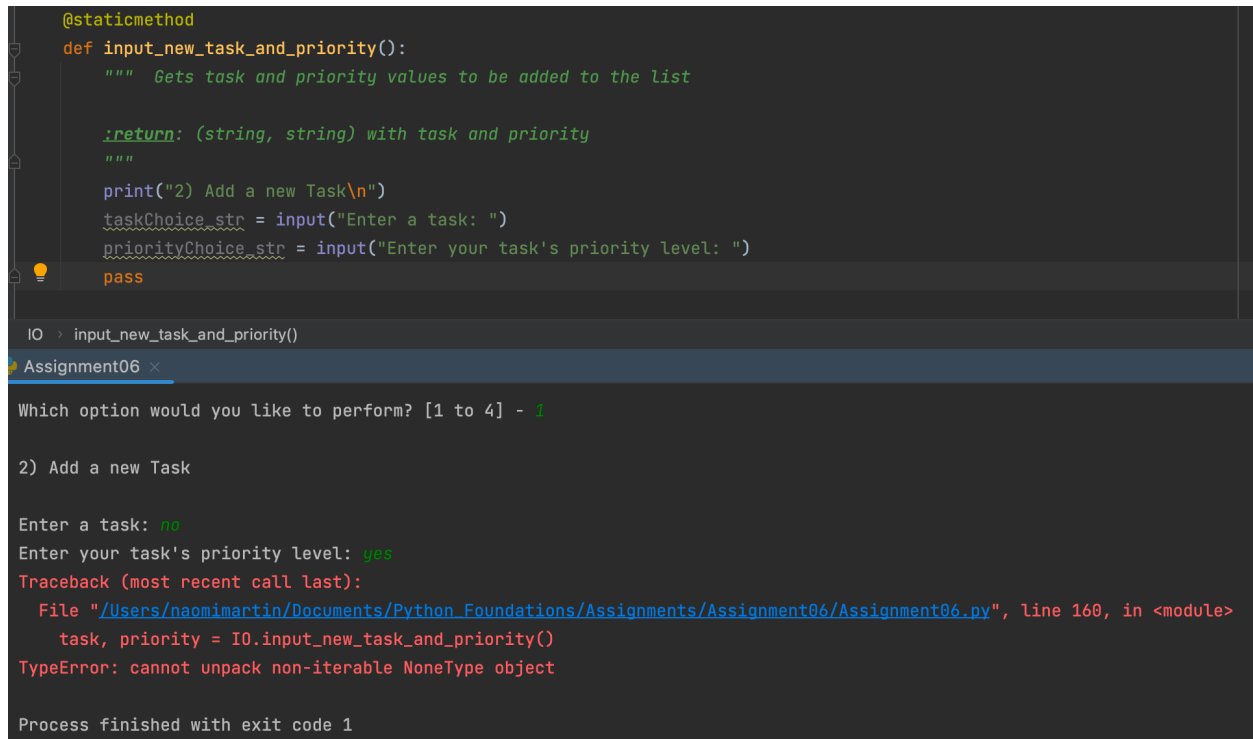
The add_data_to_list() function is intended to take the user input for task and priority and add them to the to-do list. This is done by taking in the parameters task, priority, and list_of_rows, creating a dictionary row containing the user provided task and priority arguments, and appending this dictionary row to the specified list_of_rows (figure 2).

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(taskChoice_str.title()).strip(), "Priority": str(priorityChoice_str.title()).strip()}
    list_of_rows.append(row)
    print("\nUser input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in orde
    return list_of_rows
```

Figure 2. Code to fill in the add_data_to_list() function

Once I had written the basic code for adding data to list, I decided to run the program by selecting menu option 1. Of course, I had run into an error (figure 3).



The image shows a code editor window with a Python function `input_new_task_and_priority()` and a terminal window below it. The function is a static method that prompts the user to enter a task and its priority level. The terminal shows the program running, with the user entering 'no' for the task and 'yes' for the priority level. A `TypeError: cannot unpack non-iterable NoneType object` is shown, indicating that the function did not return a tuple as expected.

```
@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
    """
    print("2) Add a new Task\n")
    taskChoice_str = input("Enter a task: ")
    priorityChoice_str = input("Enter your task's priority level: ")
    pass

IO > input_new_task_and_priority()

Assignment06 x
Which option would you like to perform? [1 to 4] - 1

2) Add a new Task

Enter a task: no
Enter your task's priority level: yes
Traceback (most recent call last):
  File "/Users/naomimartin/Documents/Python_Foundations/Assignments/Assignment06/Assignment06.py", line 160, in <module>
    task, priority = IO.input_new_task_and_priority()
TypeError: cannot unpack non-iterable NoneType object

Process finished with exit code 1
```

Figure 3. Error when attempting to run menu option 1 to add a new task.

When I went back to line 160, where the program ran into the error, I realized that the program was not able to unpack the tuple (task, priority) that I created in `input_new_task_and_priority()`. I went back to that function to see where I went wrong, and I realized that I had not even returned the variables that contained the user-inputted task and priority! Since I had not returned the variables in the form of a tuple, the tuple did not exist and the program was not able to unpack the nonexistent tuple. Once I had made this change, the user input was recorded, as indicated by a print statement that I added (figure 4).

```
@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
    """
    print("2) Add a new Task\n")
    taskChoice_str = input("Enter a task: ")
    priorityChoice_str = input("Enter your task's priority level: ")
    return (taskChoice_str, priorityChoice_str)

IO > input_new_task_and_priority()
Assignment06 x
2) Add a new Task

Enter a task: no
Enter your task's priority level: 456

User input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in order to update your To-Do List.
***** The current tasks ToDo are: *****
()
*****
```

Figure 4. User input was successfully recorded, as indicated by the print statement. However, user input was not appended to the list.

Although the input was successfully recorded, the input was not successfully added to the to-do list. I looked at the `output_current_tasks()` function to see if that was the issue, but saw nothing to change. So, I went back to look at the `add_data_to_list()` function, and noticed that I had used the local variables `taskChoice_str` and `priorityChoice_str` contained in the `input_new_task_and_priority()` function (figure 5).

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(taskChoice_str.title()).strip(), "Priority": str(priorityChoice_str.title()).strip()}
    list_of_rows.append(row)
    print("\nUser input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in orde
    return list_of_rows
```

Figure 5. Highlights my incorrect use of local variables from the `input_new_task_and_priority()` function in the `add_data_to_list` function. This is not readable by Python.

The local variables `taskChoice_str` and `priorityChoice_str` defined in the `input_new_task_and_priority()` function were unpacked from a tuple into the variables “task” and “priority”. Since the function parameters are “task” and “priority”, and we want to append the user-inputted arguments for “task” and “priority” into the to-do list, the dictionary row

here should specify the “task” and “priority” parameters rather than local variables. I had executed this concept correctly in the next line: `list_of_rows.append(row)`, where I append the variable “row” to the function parameter `list_of_rows`, but likely had forgotten in the previous line. Once I made this change, the program correctly appended my input to the list (figure 6).

```
@staticmethod
def add_data_to_list(task, priority, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param task: (string) with name of task:
    :param priority: (string) with name of priority:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    row = {"Task": str(task.title()).strip(), "Priority": str(priority.title()).strip()}
    list_of_rows.append(row)
    print("\nUser input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in order to update your To-Do List.")
    return list_of_rows

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

2) Add a new Task

Enter a task: laundry
Enter your task's priority level: high

User input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in order to update your To-Do List.

***** The current tasks ToDo are: *****
Laundry (High)
*****
```

Figure 6. The modified code to append the dictionary row containing the task and priority parameters to the to-do list, and the result of this code which showed it to be working.

Once this was done, I proofread my code and realized that the print statements I wrote to allow the user to see what menu option they were in had displayed “2) Add a new Task”. This is incorrect as the add a new task option is actually menu option 1). I went back and changed this for logical consistency.

That concluded the code that I wrote for menu option 1) Add a new task. Although the code itself was simple, I struggled a lot on using the correct syntax for functions, but armed with my newfound mistake-induced knowledge, I was ready to conquer the next part of code.

Step 2: Removing Data from List

Similarly to my approach for adding data to list, I first tackled the code for receiving user input. Again, this part was relatively simple, as I just needed to get user input (figure 7).

```

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

    :return: (string) with task
    """
    print("2) Remove an existing Task \n")
    remove = input("Choose a Task name to remove: ")
    return remove

```

Figure 7. Code for receiving user input on which task name to remove.

Next, I went to the `remove_data_from_list()` function. In the previous assignment, I had accomplished this part of data processing by assigning each row a number, and asking the user to input a number to remove a specific row. I figured it was not the most efficient or intuitive way for the user to remove a specific task, so I went to Professor Root's assignment 05 answer key for help on how to better accomplish this. The way that he accomplished this is as follows. User input was taken from the user, and a False Boolean value was defined to later verify that the data was found and removed. For each row in the to-do list, the row was converted into a dictionary so that its individual values could be extracted and put into a list with the `.values()` method. Each value in that list is then assigned to the task and priority variables. Then, an *if* statement is used to search the to-do list to see if any task names match the user input. If so, the row is removed, and the Boolean is switched to True, to indicate that the task had been found and removed. If the task had not been found in the to-do list, the Boolean remains false and a message conveying this information is displayed.

I incorporated Professor Root's assignment 05 code into my program, and changed the names of the variables to match the parameters of the function (figure 8).

```

@staticmethod
def remove_data_from_list(task, list_of_rows):
    """ Removes data from a list of dictionary rows

    :param task: (string) with name of task:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """

    itemRemoved_bln = False # naomimartin: used to verify that the data was found and removed
    for row in list_of_rows:
        task_in_list, priority_in_list = dict(row).values()
        if task.title().strip() == task_in_list:
            list_of_rows.remove(row)
            itemRemoved_bln = True

    if itemRemoved_bln == True:
        print("The task was removed. ")
    else:
        print("I'm sorry, but I could not find that task.")
    return list_of_rows

```

Figure 8. Code to remove an item from the to-do list.

When I ran this code to test the functionality, I discovered that it had worked (figure 9).

```

***** The current tasks ToDo are: *****
Laundry (High)
Dinner (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] - 2

3) Remove an existing Task

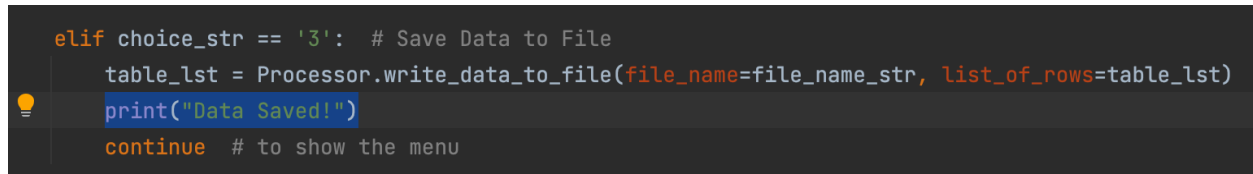
Choose a Task name to remove: laundry
The task was removed.
***** The current tasks ToDo are: *****
Dinner (High)
*****

```

Figure 9. The “Remove a Task” menu option worked.

Step 3: Saving Data to File

I first looked at the `write_data_to_file()` function and found that it was empty. However, I noticed that previously when I ran the program and selected menu option 3), a print statement displayed the message “Data Saved!”. I realized that this was because this print statement was included in the main body of the script (figure 10).



```
elif choice_str == '3': # Save Data to File
    table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
    print("Data Saved!")
    continue # to show the menu
```

Figure 10. The print statement displays the message “Data Saved!” in the main body of the script.

However, to maintain consistency with the rest of the program, which follows the principles of abstraction and encapsulation, I decided to move this print statement into the `write_data_to_file()` function.

Just like in last week’s assignment, I wrote code to extract each task and priority from the to do list, and put them into a dictionary by specifying their keys. In this way, I was able to simply save a row of data into the to-do list file by specifying the dictionary keys for task and priority. I wanted to write my code so that the program asks if the user wants to save, and awaits a “y”es or “n”o character string answer. If the user inputs “y”, the data is saved by appending rows to the file. If the user inputs “n”, a message is displayed saying that user input was not saved. If the user inputs anything other than “y” or “n”, the program says that they user has inputted an invalid option, and asks the user again if they want to save data (figure 11).


```

@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """

    while True:
        saveChoice_str = str(input("Would you like to save your data? (y)es or (n)o: ")).strip().lower()
        if saveChoice_str == "y":
            file_obj = open(file_name, "a")
            for row in list_of_rows:
                savedRow = {"Task": row["Task"], "Priority": row["Priority"]} # naomimartin: Each row in t
                file_obj.write(savedRow["Task"] + ", " + savedRow["Priority"] + "\n")
            print("User input saved.")
            file_obj.close() # naomimartin: close connection to file. Best practices.
            break
        elif saveChoice_str == "n":
            print("User input not saved.")
            break
        else:
            print("Please input a valid option.\n")
            continue
    return list_of_rows

```

Figure 11. Code that asks user if they want to save data, and either saves the data, does not save the data, or asks the user again.

I tested this code and the program executed successfully, updating ToDoFile.txt as expected. When I re-ran the program from the beginning after this test, since I “appended” the information to ToDoFile.txt rather than “writing” the information, the items I had saved in the to-do list showed back up on the screen. I also tested the functionality of the *if* statements within the *while* loop, for if the user selects “n” or an invalid option (figure 12).

```

Which option would you like to perform? [1 to 4] - 5
Would you like to save your data? (y)es or (n)o: n
User input not saved.
***** The current tasks ToDo are: *****
Dinner (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] - 5
Would you like to save your data? (y)es or (n)o: n
Please input a valid option.

Would you like to save your data? (y)es or (n)o: n
User input not saved.
***** The current tasks ToDo are: *****
Dinner (High)
*****

```

Figure 12. The functionality of the *if* statements nested within the *while* loop, to continue to ask for user input if an invalid option is selected.

Everything looked good, and this concluded the required updates to the program. The assignment was complete.

Testing in the MacOS Terminal Command Line

To completely test the functionality of the Modified To-Do List Program in the MacOS Terminal command line, I cleared the contents of the file `ToDoFile.txt` so I could start everything over. The program seemed to execute successfully, updating the `ToDoFile.txt` appropriately (figure 13).

```

Last login: Wed Aug 17 17:17:19 on ttys000
(base) naomimartin@Naomis-MacBook-Pro ~ % python3 /Users/naomimartin/Documents/Python_Foundations/Assignments/Assignment06/Assignment06.py
***** The current tasks ToDo are: *****
*****

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

1) Add a new Task

Enter a task: laundry
Enter your task's priority level: high

User input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in order to update your To-Do List.

***** The current tasks ToDo are: *****
Laundry (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

1) Add a new Task

Enter a task: dinner
Enter your task's priority level: high

User input recorded. Ensure that you select '4) Save Data to File' in the Menu of Options in order to update your To-Do List.

***** The current tasks ToDo are: *****
Laundry (High)
Dinner (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] - 2

2) Remove an existing Task

Choose a Task name to remove: laundry
The task was removed.

***** The current tasks ToDo are: *****
Dinner (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] - 3

Would you like to save your data? (y)es or (n)o: y
User input saved.
***** The current tasks ToDo are: *****
Dinner (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] - 4

Goodbye!
(base) naomimartin@Naomis-MacBook-Pro ~ % python3 /Users/naomimartin/Documents/Python_Foundations/Assignments/Assignment06/Assignment06.py
***** The current tasks ToDo are: *****
Dinner (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] - █

```

Figure 13. a) A test run through the Updated To-Do List Program showing successful processes of adding a new task, removing an existing task, and saving data to file.

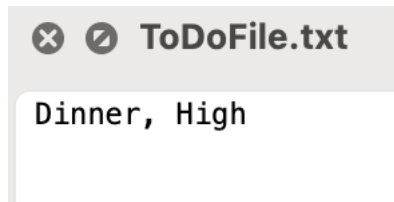


Figure 13. b) Data successfully saved to the file.

The program executed successfully in the command line.

Conclusion

This assignment again tested my understanding of proper Python syntax, in the context of differentiating the parameters of functions from the local variables of different functions. In addition, it challenged me to make sense of working with someone else's code. I seem to find it difficult to read through and make sense of a script that I did not write. While I've found it very beneficial to work with Professor Root's code, both in terms of learning and in terms of time consumption, I would like to practice writing my own pseudocode and planning out complex programs on my own. Armed with newfound knowledge about functions, I am excited at the possibilities that I can create.