

To-Do List Using Functions

1.1 Introduction:

This paper describes the To-Do list script, in which functions are used for each section of the script. These functions detail all the necessary steps including displaying a user's To-Do list, allowing them to input and save their current and/or new tasks, and exiting the script. The main body of the script later calls these functions and assigns arguments to each function's parameters to be properly executed.

2.1 Creating Variables and Initial Processing:

Since this assignment begins from starter code written by Professor Root, the global variables for this script had already been given names and initial values. I did some minor adjustments to these variables and to the first function `read_data_from_file`. In the `row_dic` variable, I added the keys 'Task' and 'Priority' just to help me keep track of what was needed for each row of data that would be input. I also renamed the `file` variable, whose default value is set to None, to `file_obj`, so as to keep track of the difference with the `file_name_str` variable, which kept the name of the .txt file that is called. These changes can be seen in **Figure 1**:

```
13  # Data ----- #
14  # Declare variables and constants
15  file_name_str = "ToDoFile.txt" # The name of the data file
16  file_obj = None # An object that represents a file
17  row_dic = {"Task", "Priority"} # A row of data separated into elements of a dictionary {Task,Priority}
18  table_lst = [] # A list that acts as a 'table' of rows
19  choice_str = "" # Captures the user option selection
20
21  # Processing ----- #
22  class Processor:
23      """ Performs Processing tasks """
24
25      @staticmethod
26      def read_data_from_file(file_name, list_of_rows):
27          """ Reads data from a file into a list of dictionary rows
28
29          :param file_name: (string) with name of file:
30          :param list_of_rows: (list) you want filled with file data:
31          :return: (list) of dictionary rows
32          """
33          list_of_rows.clear() # clear current data
34          file_obj = open(file_name_str, "r")
35          for line in file_obj:
36              task, priority = line.split(",")
37              row = {"Task": task.strip(), "Priority": priority.strip()}
38              list_of_rows.append(row)
39          file_obj.close()
40          return list_of_rows
```

Figure 1 —Slight modifications to the existing variables for clarity, specifically the `file` and `row_dic`.

In **Figure 1**, I renamed all instances of the variable `file` to `file_obj` so that it was clear to me that this variable would be the one to open and read the existing data of the `.txt` file.

2.2 Adding Data:

The first function which needed code was the `add_data_to_list` function. As per the function's name and its docstring, it takes three parameters: the task, the task's priority, and the list of rows of which these items will be added. Once the function is called, each parameter will have data passed into it.

The starter code had already defined and included the first two parameters (task and priority) in a variable called `row`. **Row** is set to equal a dictionary which contains the keys 'task' and 'priority'; the dictionary's values are set to two strings 'task' and 'priority' followed by the `.strip()` function. The `.strip()` function will remove any invisible spaces that the user may have typed when adding data to the original `.txt` file.

`List_of_rows` is a parameter for this function, meaning that once the function is called, a variable containing data is expected to fill it. Further down in the body of the script (**Figure 2**), we can see that `list_of_rows` has been set to `table_lst`, the empty list from **Figure 1**, showing that an argument, specifically `table_lst`, will be passed into this parameter. The variable `table_lst` serves as the list described in the docstring.

```
165 # Step 4 - Process user's menu choice
166 if choice_str.strip() == '1': # Add a new Task
167     task, priority = IO.input_new_task_and_priority()
168     table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
169     continue # to show the menu
```

Figure 2 — Step 4 from the main body of the starter code, showing that the parameter `list_of_rows` will be filled by the variable `table_lst`.

```
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 filled with file data:
49     :return: (list) of dictionary rows
50     """
51     # TODO: Add Code Here!
52     row = {"Task": str(task).strip(), "Priority": str(priority).strip()}
53     list_of_rows.append(row)
54     return list_of_rows
```

Figure 3 — Code to add data to existing list. The parameter `list_of_rows` is used in the function until the function is called, after which it is replaced by `table_lst`.

Figure 3 displays my code for this function. After the `row` variable from the starter code, I called the parameter `list_of_rows` and added the `.append()` function to it. Within the `.append()` function,

I included the variable `row`. When run as part of this function, this line of code will add the existing user tasks and their priorities to the existing empty list, which serves as the table to display the To-Do list data.

2.3 Removing Data:

Removing the user's data in this assignment relied on two functions: `input_task_to_remove` and `remove_data_from_list`. The first function serves as the input, in which it prompts the user to input the task name they'd like to remove and stores that value for it to later be accessed.

To accomplish this first step, I added a local variable in the `input_task_to_remove()` function. This variable, **`removeTask`**, took in a string name that the user had input. Since the docstring only requires returning the task name, I only added a return statement that returned the variable. This code can be seen in **Figure 4**:

```
144     @staticmethod
145     def input_task_to_remove():
146         """ Gets the task name to be removed from the list
147
148         :return: (string) with task
149         """
150         # TODO: Add Code Here!
151         removeTask = str(input("Write the name of the task you'd like to remove: "))
152         return removeTask
```

Figure 4 — Code that takes in user input and stores the name of the task they'd like removed from the To-Do list.

The second function for this section in the script to work involves checking the name of the user-input task with the existing To-Do tasks in the table of data. To accomplish this, I created a for loop that looped through table.

Within this for-loop I included an if statement that checked if that row containing a task was equal to the parameter task. The task parameter comes from assigning the `input_task_to_remove()` function to this parameter in the main body of the script. By doing this, we can access the `removeTask` from the `input_task` function, despite the variable and its data being a local variable and solely pertaining to that function.

Once the script calls each function and checks to see if the task names match, the `.remove()` function appended to the table of data removes the relevant row. This can be seen in **Figure 5**:

```

57     @staticmethod
58     def remove_data_from_list(task, list_of_rows):
59         """ Removes data from a list of dictionary rows
60
61         :param task: (string) with name of task:
62         :param list_of_rows: (list) you want filled with file data:
63         :return: (list) of dictionary rows
64         """
65         # TODO: Add Code Here!
66         for row in list_of_rows:
67             if row["Task"].lower() == task: # task is string that is in returnTask
68                 #_once_input_task_function_is_called
69                 #_the_user_input_(string)_replaces_the_task_parameter
70                 list_of_rows.remove(row)
71         return list_of_rows # list_of_rows is the parameter of which table_lst is later assigned to
72                             # when the function is called

```

Figure 5 — Code that takes the task parameter from the previous function and checks it against the available tasks in the To-Do list.

2.4 Writing Data to File:

The function that saves a user's data to the text file takes in two parameters: the file name and the list of rows which contains the data. The file's name is "ToDoList.txt", which was previously defined in the variable `file_name_str` from section 1.1.

Following the docstring for this function, the text file needs to be accessed and the function needs to add each individual task and its priority, regardless of how many tasks the user input. To do this, I re-assigned the variable **file_obj** to re-open the text file in write mode, which would allow the user's data to be written to the text file, as it had previously been in read mode.

```

72     @staticmethod
73     def write_data_to_file(file_name, list_of_rows):
74         """ Writes data from a list of dictionary rows to a File
75
76         :param file_name: (string) with name of file:
77         :param list_of_rows: (list) you want filled with file data:
78         :return: (list) of dictionary rows
79         """
80         # TODO: Add Code Here!
81         file_obj = open(file_name_str, 'w')
82         for row in list_of_rows:
83             file_obj.write(row['Task'] + ',' + row['Priority'] + '\n')
84         file_obj.close()
85         return list_of_rows

```

Figure 6 — Code that writes the To-Do list data to the .txt file

The for loop in **Figure 6** loops through the list parameter, checking each row for a task and its priority. For each pair that it finds, the task and its priority are written to `file_obj` with the `.write()` function. Once the script has looped through all the user data, it exits the loop and reads the next line of code, which closes the text file through the `.close()` function.

2.5 Taking in New User Data:

To take in user data I created two local variables for this function: `new_Task` and `task_Priority`. Both variables take strings and prompt the user to input the task they want to complete and one of three priorities (high, medium, or low). As local variables they only work within the function. These values are then passed into the global variable `row_dic`, which holds each string in its respective key:value pair. That is, 'Task' with `new_Task` and 'Priority' for the `task_Priority` variable.

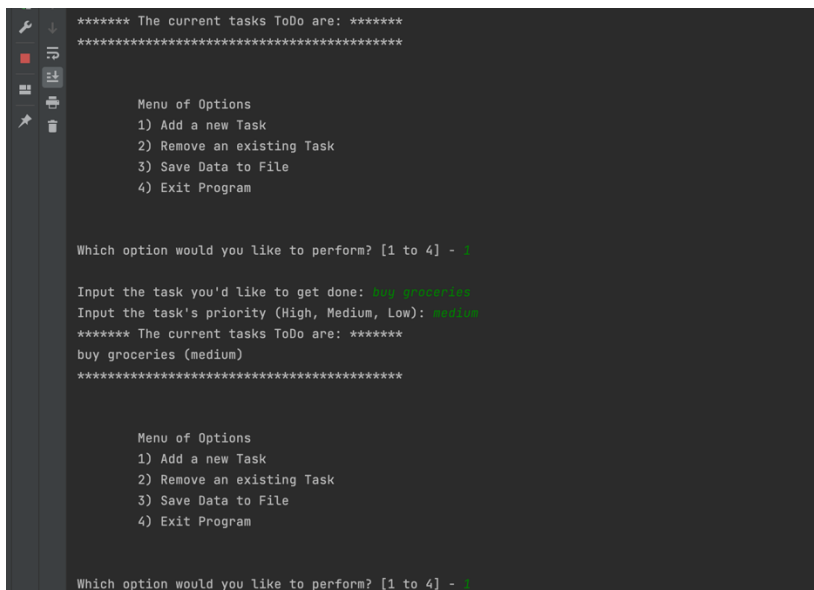
```
130     @staticmethod
131     def input_new_task_and_priority():
132         """ Gets task and priority values to be added to the list
133
134         :return: (string, string) with task and priority
135         """
136         # TODO: Add Code Here!
137         new_Task = str(input("Input the task you'd like to get done: "))
138         task_Priority = str(input("Input the task's priority (High, Medium, Low): "))
139         row_dic = {'Task':new_Task, 'Priority':task_Priority}
140         return (new_Task, task_Priority)
```

Figure 7 — Code that prompts user input for a new task and its priority. Those values are then assigned to a dictionary

3.1 Running the Script:

The following figures show the script running in PyCharm and in the MacOS Command Shell.

Figures 8 – 10 show the script running in PyCharm. It adds two tasks, saves them to the file, and exits the program. **Figure 11 — 13** show the script running in the command shell, in which two new tasks are added and saved.



```
***** 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

Input the task you'd like to get done: buy groceries
Input the task's priority (High, Medium, Low): medium
***** The current tasks ToDo are: *****
buy groceries (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] - 1
```

Figure 8 — Script running in PyCharm adding the new task buy groceries to the To-Do list.

```
Run: Assignment06 x
Input the task you'd like to get done: write article chapter
Input the task's priority (High, Medium, Low): high
***** The current tasks ToDo are: *****
buy groceries (medium)
write article chapter (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

Data Saved!
***** The current tasks ToDo are: *****
buy groceries (medium)
write article chapter (high)
*****

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

Figure 9 — Script in PyCharm adding a new task to the existing list and saving the current data.



ToDoFile.txt

```
buy groceries, medium
write article chapter, high
```

Figure 10 — Text file showing the saved tasks

```

Last login: Tue Nov 22 15:08:21 on ttys000
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.
Sebs-MacBook-Air:~ milaclendenning$ cd ~/Documents/_PythonClass/Assignment06
Sebs-MacBook-Air:Assignment06 milaclendenning$ python3 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

Input the task you'd like to get done: wash dishes
Input the task's priority (High, Medium, Low): high
***** The current tasks ToDo are: *****
wash dishes (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

Input the task you'd like to get done: take out trash
Input the task's priority (High, Medium, Low): medium
***** The current tasks ToDo are: *****
wash dishes (high)
take out trash (medium)
*****
```

Figure 11 — Running the script in the MacOS command shell and adding two new tasks

```

***** The current tasks ToDo are: *****
wash dishes (high)
take out trash (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] - 3

Data Saved!
***** The current tasks ToDo are: *****
wash dishes (high)
take out trash (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 12 — Saving the new tasks to the text file

```

wash dishes, high
take out trash, medium
```

 **ToDoFile.txt**

Figure 13 — The saved tasks displayed in the text file

4.1 Summary:

This paper has described writing the To-Do list script using functions. These functions allow the user to input new data, save data to a file, and remove data from the To-Do list. By setting parameters from function headers to call other functions, this allows the entire script to access

local variables and the data stored within them, in which local variables are usually restricted to only being read and used within the functions they are created for.