
Mariia Kaminskaia

February 21, 2023

IT FDN 110 A Wi 23: Foundations Of Programming: Python

Assignment_06

Link to GitHub: <https://github.com/moriia/IntroToProg-Python-Mod06>

Create a ToDo list using functions


Introduction

In this assignment, I had to modify a script that manages a ToDo list, using a dictionary object and custom functions. Add, remove, display entered data, and save entered data to the file. This document summarizes the steps I performed to complete the assignment.

Processing data

Step 1. When the program starts, Load data from ToDoFile.txt.

There was a template that I had to modify and use for my program called "Assignment06_Starter.py". In the beginning of Main Body of the script, the code loads data from a file into a Python List of Dictionary objects. It calls a function "read_data_from_file" located in class "Processor" and function includes the name of the parameters filled explicitly with the arguments (**Figure 1**).



```
9
0  # Main Body of Script ----- #
1
2
3  # Step 1 - When the program starts, Load data from ToDoFile.txt.
4  Processor.read_data_from_file( file_name=file_name_str, list_of_rows=table_lst) # read file data
5
```

Figure 1. Main Body of the script which calls a read_data_from_file

Function contains the next logic: it opens a file and reads data from a file into a list of dictionary rows (**Figure 2**).

```

@staticmethod
def read_data_from_file(file_name, list_of_rows):
    """ Reads data from a file into a list of dictionary rows

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

    # noinspection PyBroadException
    try: #Try statement for case the file is not exist
        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
    except:
        pass

```

Figure 2. Read_data_from_file function.

To prevent an error in case file does not exist, I included the “try-except” (**Figure 2**) block which will handle an exception. This prevents abrupt exits of the program on error.

So In case file exists, it will be opened and data will be read from the file into a list of dictionary rows. For the case file does not exist I added pass statement, so if the program runs and file does not exist nothing happens, it avoids getting an error.

Step 2. Display a menu of choices to the user

The second block of the main script body is to preset to user menu of options, actions user can accomplish using the program. For this purpose firstly I am calling a method “output_current_tasks_in_the_list” in class “IO” (aka Input/Output) which shows to user the tasks existing in the list. After it, the main menu is shown to user by calling “output_menu_tasks” from the same IO class. As soon as menu displayed to the user, user will be asked to select one of the option from the menu by calling “input_menu_choice” and the result will be assigned to “choice_str” variable (**Figure 3**).

```

# Step 2 - Display a menu of choices to the user and handle user choice
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

```

Figure 3. Block of code 2 - to show current data and show menu to the user.

Step 2.1.Show current data in the table list.

To show user a data currently existing in the list/table the method “output_current tasks_in_the list” is called. The argument “table_list” is passed to the function using keyword “list_of_rows”, which is a name of parameter function should receive (**Figure 4**).

```

@staticmethod
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
    """
    print() # Line for looks
    print("***** The current tasks ToDo are: *****")
    if len(list_of_rows) == 0:
        print("There is no data in the list, yet")
    else:
        for row in list_of_rows:
            print(row["Task"] + " (" + row["Priority"] + ")")
        print("*****")
    print() # Line for looks

```

Figure 4. Function to show user tasks which are existing the list of rows.

The function gets as a parameter list_of_rows, in case length or number of rows is 0 user will get the an message that there is no items in the list, otherwise the list will be presented to him. Each row will be printed to user in the format ‘task (priority)’. The function does not return anything to the main script. After function finished the user will sees a main menu (**Figure 3**).

Step 2.2. Show menu of options to user.

To show user a menu the void method “output_menu_tasks()” is called (**Figure 5**). As a result of calling the method user sees the menu from which he is required to select at least one option.

```
class IO:
    """ Performs Input and Output tasks """

    @staticmethod
    def output_menu_tasks():
        """ Display a menu of choices to the user

        :return: nothing
        """
        print(''
Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program
'')
        print() # Line for looks
```

Figure 5. Menu of options which will be printed to the user.

Step 2.3. Get user's choice of option.

After the menu displayed to user, he is asked to select one of for option. The method input_menu_choice() (**Figure 6**) is a simple method which return string as a user's choice. The method strip() is used here to remove any spaces at the beginning and spaces at the end. As soon as user entered his choice the spaces removed from the end and the beginning. The adjusted string of user's choice send back to the caller (main strip body) by using return statement.

```

def input_menu_choice():
    """ Gets the menu choice from a user

    :return: string
    """
    print() # Line for looks
    return input("Which option would you like to perform? [1 to 4] - ").strip()

```

Figure 6. Function which receives value entered by user and sending it back using return statement.

Step 3. Processing the data

Next step in the main script body is to process data according user's choice (**Figure 7**). In case user chooses 1st option '1' - the new data will be added to the list. In case user's choice is '2' - the data will be removed from the list. If user selects option '3' data will be saved to the file. After selecting option '4' program will be finished (the option menu previously shown in **Figure 5**).

```

193 # Step 4 - Process user's menu choice
194 if choice_str == '1': # Add a new Task
195     task, priority = IO.input_new_task_and_priority()
196     table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
197     continue # to show the menu
198
199 elif choice_str == '2': # Remove an existing Task
200     task = IO.input_task_to_remove()
201     table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
202     continue # to show the menu
203
204 elif choice_str == '3': # Save Data to File
205     table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
206     continue # to show the menu
207
208 elif choice_str == '4': # Exit Program
209     print("Goodbye!")
210     break # by exiting loop
211
212 else:
213     print("Please choose one of valid options: '1' or '2' or '3' or '4'")
214     continue # to show the menu

```

Figure 7. Block of code used to process data according to user's choice.

Step 3.1. Add new task to the list

After user entered the option he wants to perform, his choice is stored in variable choice_str. If user's choice '1' the two methods will be called. First of them is input_new_task_and_priority() located in the class 'IO'.

Step 3.1.1. Input new task and its priority.

“Input_new_task_and_priority” method (line 195 - **Figure 7**) asks user to enter task and its priority and returns them as a string (**Figure 8**). I did not cast task and priority values as a string, since input() automatically converts the entered user value to a string.

```
157     @staticmethod
158     def input_new_task_and_priority():
159         """ Gets task and priority values to be added to the list
160
161         :return: (string, string) with task and priority
162         """
163         task = input("Please enter the task: ").strip()
164         priority = input("Please enter the task priority: ").strip()
165         print() # Line for looks
166         return task,priority
```

Figure 8. Function used to return user’s entered values of task and priority.

Step 3.1.2. Add entered task and priority to the list.

To add new rows to the list the “add_data_to_list” method located in “Processor” class is being used. To make it work the function needs to receive 3 parameters: task, priority and list of rows (**Figure 9**). Parameters task and priority will be used to build a dictionary, and after it this dictionary row will be added to the list_of_rows using append method, to add a new row at the end. The function returns updated list_of_rows.

```
46     @staticmethod
47     def add_data_to_list(task, priority, list_of_rows):
48         """ Adds data to a list of dictionary rows
49
50         :param task: (string) with name of task:
51         :param priority: (string) with name of priority:
52         :param list_of_rows: (list) you want to add more data to:
53         :return: (list) of dictionary rows
54         """
55         row = {"Task": task, "Priority": priority}
56         list_of_rows.append(row)
57         return list_of_rows
```

Figure 9. Function used to add data to the list.

To make it work in the main scrip body we passes arguments task, priority and table_lst (**Figure 10** - line 196).

```
193     # Step 4 - Process user's menu choice
194     if choice_str == '1': # Add a new Task
195         task, priority = IO.input_new_task_and_priority()
196         table_lst = Processor.add_data_to_list(task=task, priority=priority, list_of_rows=table_lst)
197         continue # to show the menu
```

Figure 10. The block of code responsible to get user's task and priority and add it to the list.

After newly updated table_list is returned after calling the method, continue statement returns the control to the beginning of the while loop and user will see the main menu again.

Step 3.2. Remove an existing Task

If user selects option '2' from the main menu, the block of code responsible to delete data runs. This block includes two methods. First method is "input_tasks_to remove" from the "IO" class, which returns string of task user wants to remove. Variable "task" is assigned to value which will be returned from the call of "input_tasks_to remove" method. Second method is "remove_data_from_the_list" from the "Processor" class, value returned from the call assigned to variable table_lst (**Figure 11**).

```
199     elif choice_str == '2': # Remove an existing Task
200         task = IO.input_task_to_remove()
201         table_lst = Processor.remove_data_from_list(task=task, list_of_rows=table_lst)
202         continue # to show the menu
```

Figure 11. The block of code responsible to aks user what item he wants to remove and delete from the list afterwards.

Step 3.2.1. Ask user for a task he wants to remove.

In the method "input_task_to_remove" user is asked to input the name of task he wants to remove. The received value is adjusted by removing spaces before or after (if the will exist) using strip() method. And received value returned to as a result of calling (**Figure 12**). There is no need to display user the task list again, since it was displayed together with the main menu options.

```

168     @staticmethod
169     def input_task_to_remove():
170         """ Gets the task name to be removed from the list
171
172         :return: (string) with task
173         """
174         print() # Line for looks
175         return input("What is the name of task you wish to remove? - ").strip()

```

Figure 12. Function which receives the user input regarding task he wants to remove.

Step 3.2.2. Remove data from the list.

Task name received from the “input_task_to_remove” method is passed as an argument to the “remove_data_from_list” function together with “table_lst”. There 3 possibilities how deletion can go. First, there is no data in the table. Second, user entered a task name which is not matching any existing task. Third, task name entered by user found in the list and can be removed. So the function “remove_data_from_list” should handle all of those scenarios (**Figure 13**).

```

72     @staticmethod
73     def remove_data_from_list(task, list_of_rows):
74         """ Removes data from a list of dictionary rows
75
76         :param task: (string) with name of task:
77         :param list_of_rows: (list) you want filled with file data:
78         :return: (list) of dictionary rows
79         """
80         if len(list_of_rows)==0:
81             print() # Line for looks
82             print("There is no task to remove, task list is empty\n")
83         else:
84             row = Processor.search_data_in_the_list(task, list_of_rows)
85             if not row:
86                 print() # Line for looks
87                 print("The task does not exist in the list\n")
88             else:
89                 list_of_rows.remove(row)
90                 print() # Line for looks
91                 print("The task removed successfully!\n")
92         return list_of_rows

```

Figure 13. Function used to remove data from the list.

First part of the function (lines 80-82- **Figure 13**) handles scenario for the case if there is no data in the list. For this case there is a check if at least 1 line in the list of task exists. To do this check i use len() method, If the length of the list is 0, the message that the table is empty and there is nothing to delete is shown to the user.

Second part handle the case if there is at least 1 task exists in the list of the tasks. In this case function “search_data_in_the_list” is called. To run in it needs to pass to it two parameters, 1st is a task - the one user entered to delete and the second list of rows - list of exiting items (line 84 - **Figure 13**).

The “search_data_in_the_list” method perform the search of task through the list of existing task (**Figure 14**).

```
59      @staticmethod
60      def search_data_in_the_list(task, list_of_rows):
61          """ Search for a task user wants to remove in list of dictionary rows
62
63              :param task: (string) with name of task:
64              :param list_of_rows: (list) list of data:
65              :return: (row) of dictionary rows if data is found
66          """
67          for row in list_of_rows:
68              if row["Task"].lower() == task.lower(): return row
69              else: return False
```

Figure 14. Function used to search for an item in the list of existing tasks.

In case there is a match the row where the match exists will be returned, in case there is no match the function returns “False”. The value returned from the function run is assigned to variable row in the “remove_data_from_list” (line 84 - **Figure 13**).

If the match is not found user will be notified that item he wants to remove from the list does not exist there (lines 85-87 - **Figure 13**).

If there is a match the item will be deleted from the list using remove method, which takes a single element as an argument and removes it from the list (lines 88-91 - **Figure 13**).

After all the “remove_data_from_list” returns an updated list of tasks.

Step 3.3. Save updated data

If user selects option '3' from the main menu, the block of code responsible to save data to the file will be run. This block includes 1 method - "write_data_to_file" from the "Processor" class, which returns list of task. To call the method arguments file_name_str and table_lst are passed to the function, using keywords file_name and list_of_rows accordingly (**Figure 15**).

```
205 elif choice_str == '3': # Save Data to File
206     table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
207     continue # to show the menu
```

Figure 15. The block of code responsible to save data to the file.

Step 3.3.1. Write data to the file.

Function write_data_to_file opens a file for the writing. file_name variable is provided at the beginning of the scrip and store the real file name. After the file is opened for writing, each row will be written to the file. After the last item existing in the list of tasks is written to the file. User will be informed that the data is saved successfully (**Figure 16**). And the function will return list of the items.

```
95 @staticmethod
96 def write_data_to_file(file_name, list_of_rows):
97     """ Writes data from a list of dictionary rows to a File
98
99     :param file_name: (string) with name of file:
100     :param list_of_rows: (list) you want filled with file data:
101     :return: (list) of dictionary rows
102     """
103     file = open(file_name, "w")
104     for row in list_of_rows:
105         file.write(row["Task"] + "," + row["Priority"] + "\n")
106     file.close()
107     print("Data saved successfully!")
108     return list_of_rows
```

Figure 16. Function used to write data to the file.

Step 3.4. Exit program

If user selects option '4' from the main menu, the block of code responsible to exit the program will be run (**Figure 17**). It just calls the method `exit()` and program will be terminated.

```
209 elif choice_str == '4': # Exit Program
210     print("Goodbye!")
211     exit() # by exiting loop
```

Figure 17. The block of code responsible to exit the program.

Step 3.5. Handle the invalid user's input

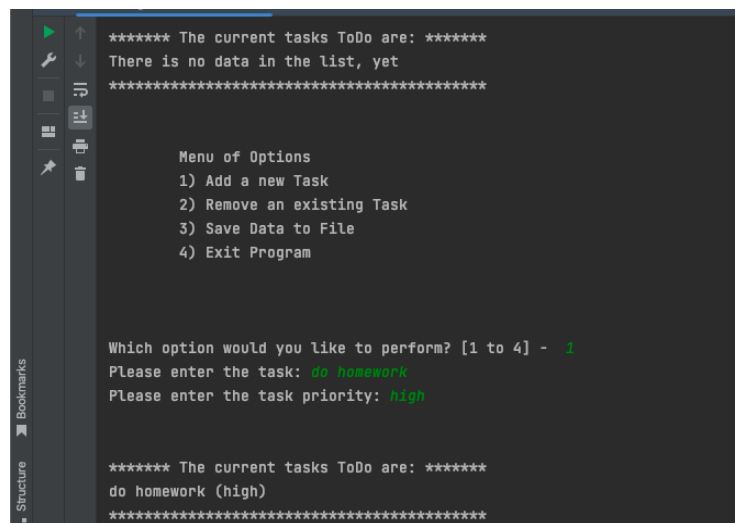
If user selects not existing option, the block of code responsible to handle it will be executed (**Figure 18**). In case user enter any invalid value, he will be asked to enter a valid option and will be returned to the main menu.

```
213 else:
214     print("Please choose one of valid options: '1' or '2' or '3' or '4'")
215     continue # to show the menu
216
```

Figure 18. The block of code responsible to handle the invalid user input.

Step 4. Run the code in the PyCharm and Terminal

After script was finished it was run in the pycharm (**Figure 19**) .



```
***** The current tasks ToDo are: *****
There is no data in the list, yet
*****

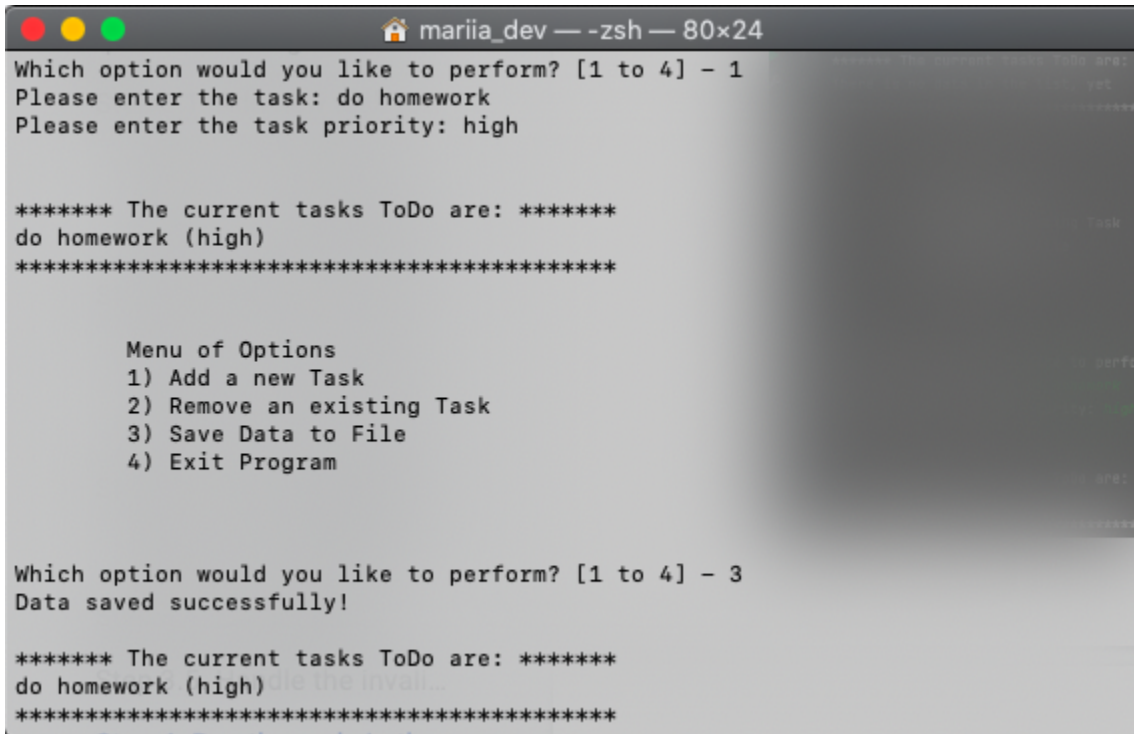
Menu of Options
1) Add a new Task
2) Remove an existing Task
3) Save Data to File
4) Exit Program

Which option would you like to perform? [1 to 4] - 1
Please enter the task: do homework
Please enter the task priority: high

***** The current tasks ToDo are: *****
do homework (high)
*****
```

Figure 19. Result of running script in the PyCharm.

After I run the program in terminal (**Figure 20**).



```
mariia_dev — -zsh — 80x24
Which option would you like to perform? [1 to 4] - 1
Please enter the task: do homework
Please enter the task priority: high

***** The current tasks ToDo are: *****
do homework (high)
*****

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

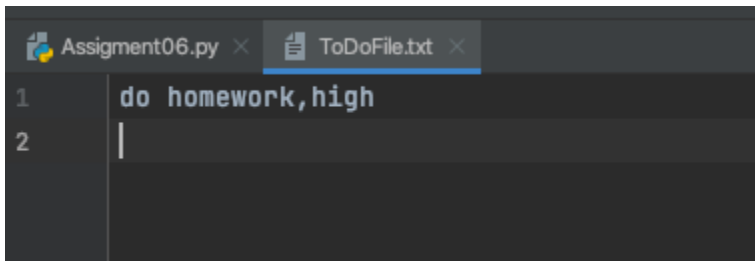
Which option would you like to perform? [1 to 4] - 3
Data saved successfully!

***** The current tasks ToDo are: *****
do homework (high)
*****
```

Figure 20. Result of running scrip in Terminal

Step 5. Check data is saved

After the script was run I checked the file, where the data was supposed to be saved (**Figure 21**).



```
Assignment06.py x ToDoFile.txt x
1 do homework,high
2 |
```

Figure 21. Result of running script, the data saved in the file.

Add a GitHub Web Page

To add a GitHub web page I entered a repository I created for the assignment 06. I created a file index.md under the docs folder in "IntroToProg-Python-Mod06" repository. I added the code provided in the assignment task (**Figure 22**):

```
# Module06 Website
---
[Google Homepage](https://www.google.com "Google's Homepage")
[GitHub Webpage Code
CheatSheet](https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet)
```

Figure 22. Example of text added to index.md file for creating a GitHub webpage.

After it I opened settings and set that my GitHub Pages site is being built from the /docs folder in the main branch. So now I will be able to access the GitHub webpage (**Figure 23**). The github site can be accessed using link: <https://moriia.github.io/IntroToProg-Python-Mod06/>.

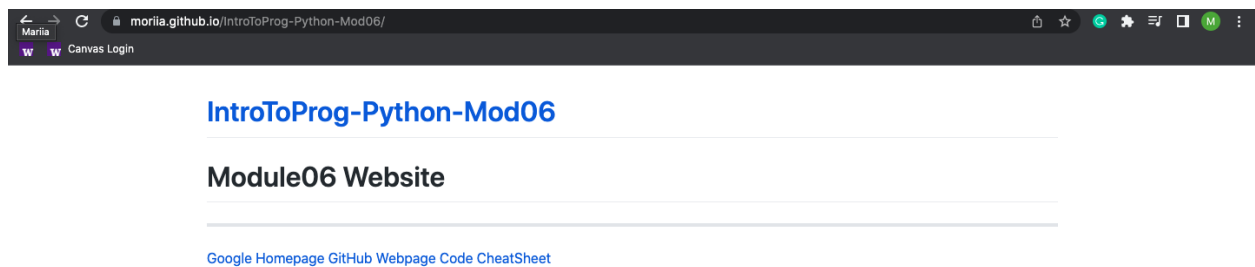


Figure 23. GitHub webpage created in the repository.

Summary

While completing the assignment I learned about: how to use custom function, how to organize code using “Separations of Concerns” pattern. What is the difference between arguments passing to the function and parameter function will use. I learned how return values from the function, and that they can be assigned to the variables which can be used later. While working with variables during assignment I understood the difference between a global and a local variables. And as a cherry on top I learned how to creat a simple webpage in GitHub.