**Name :** Leonardo Espinoza

**Date :** August 9, 2023

**Course :** Foundations of Programming, Python

**Assignment 05**

**Github : https://github.com/lesping/IntroToProg-Python**

The Household Items Inventory Program

# Introduction

In this module 5, we learned how to create scripts that work with collections of data in the form of lists to works with sequences of information and dictionaries to works with pairs of data. We also review some basic tools needed to improve code as it gets longer and more complex, such as functions, separation concerns, and script templates. Additionally, we started using the source control software called GitHub that serves to back up our code files and make them available to work collaboratively with other developers. This week the challenge was to complete and modify a provided script that manages a To Do List. In the next section I will indicate the steps carried out to obtain the result requested in assignment task 5.

# Improving the script that manages a “Daily To Do List”

***I.1 Defining Variables***

I started the code by using the *print()* function to add a welcome phrase, along with the *input()* function to ask the user if he or she wants to start the “To Do List”. Then, I reviewed the provided script and noted that I was going to need some additional variables, specifically to add new tasks and their priority level, so I declared two additional string variables at the beginning of the code leaving the string blank for the user to add the data (*task\_str and priority\_str*) (Figure 1).

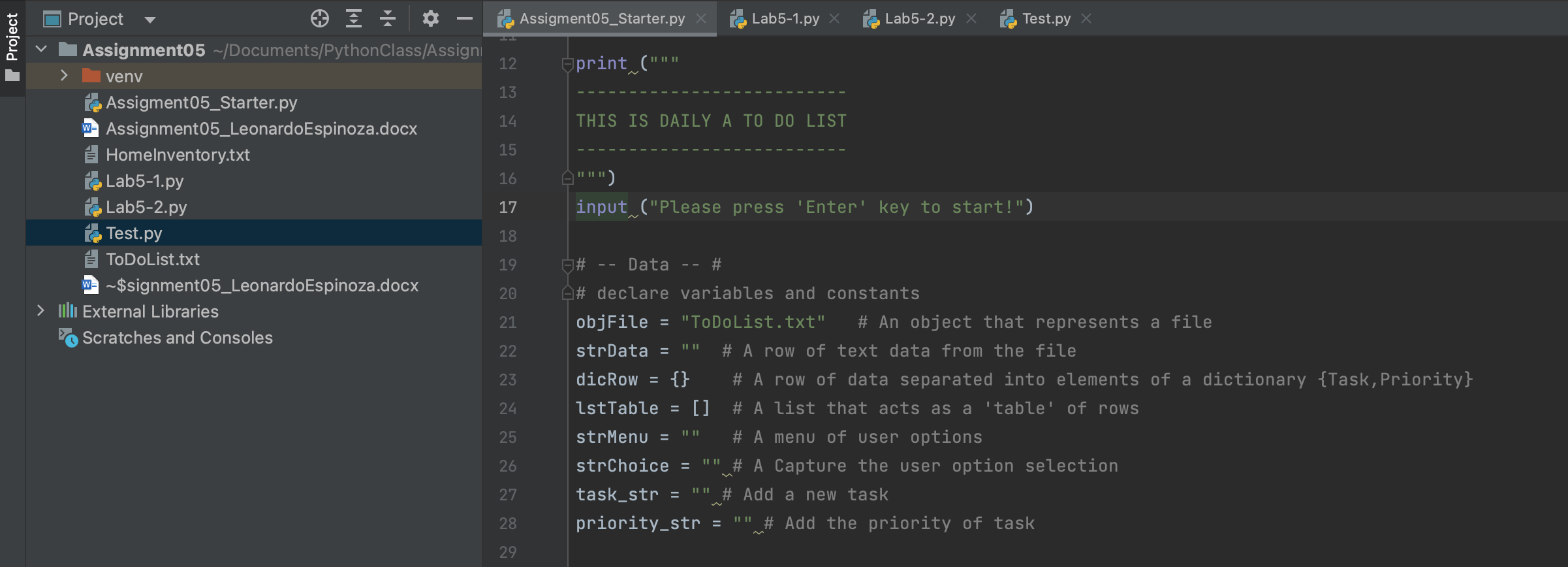


Figure 1: Welcome phrase & declaring variables

***I.2 Completing the code by steps***

The first part of the code requested to load the data that was already stored in the .txt file, so I used the *open()* function with the *"r"* option to read the information. I then manipulated the data to add it in the *dicRow{}* dictionary variable using the Task and Priority key elements and displayed the data in the screen with the *print()* function. Since I was going to continue working with this information, according to the different options in the program's menu, I used *append()* methods to add the data in the form of dictionary to the *lstTable[]* variable. Additionally, to improve the user experience, I used the try & except function to display the message *“File not found”* in case the ToDoList.txt file is not found.

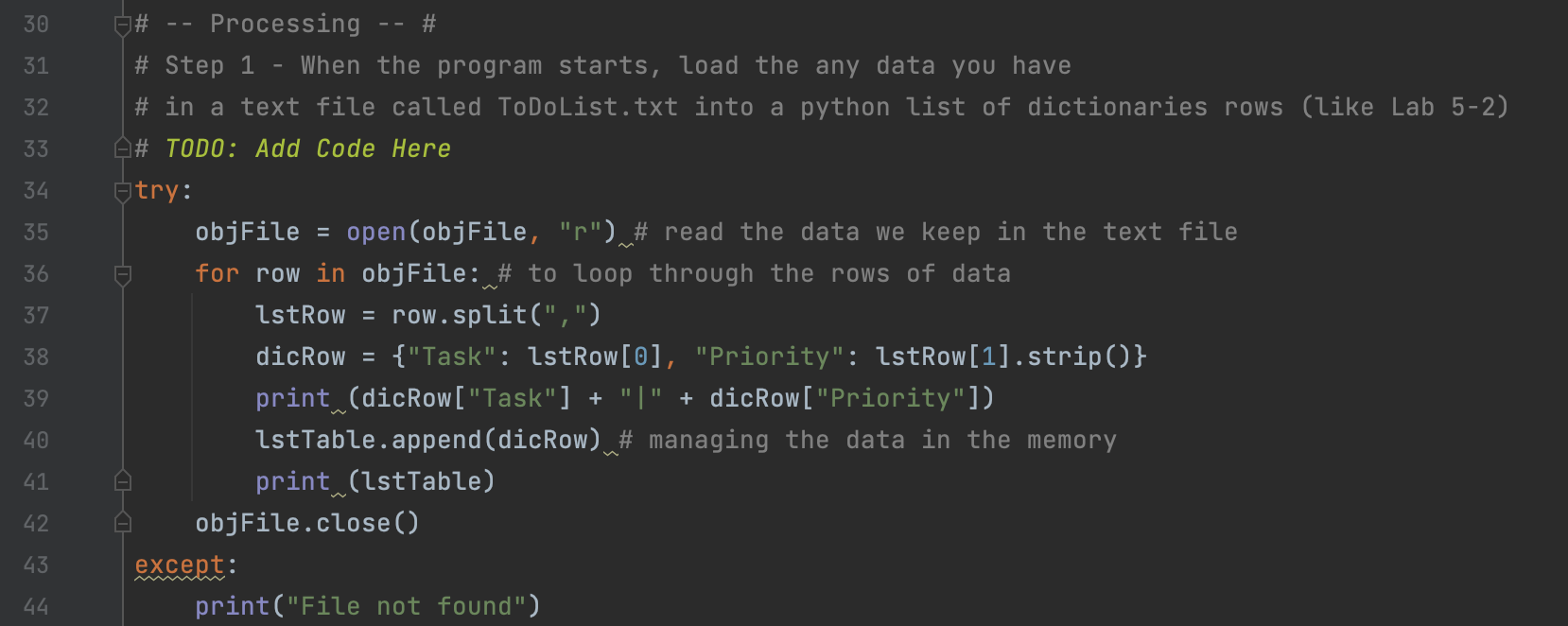


Figure 2: Code to load the data from the ToDoList.txt

Step 2 displays the menu options with the *print()* function, which was provided in the original script. To complete step 3, corresponding to *"1) Show current data"*, I used the *For loop* to iterate through the different "rows" of data from the dictionary that are part of the list table. I then used the *print()* function to display the information to the user (I displayed the data in two different ways) (Figure 3).

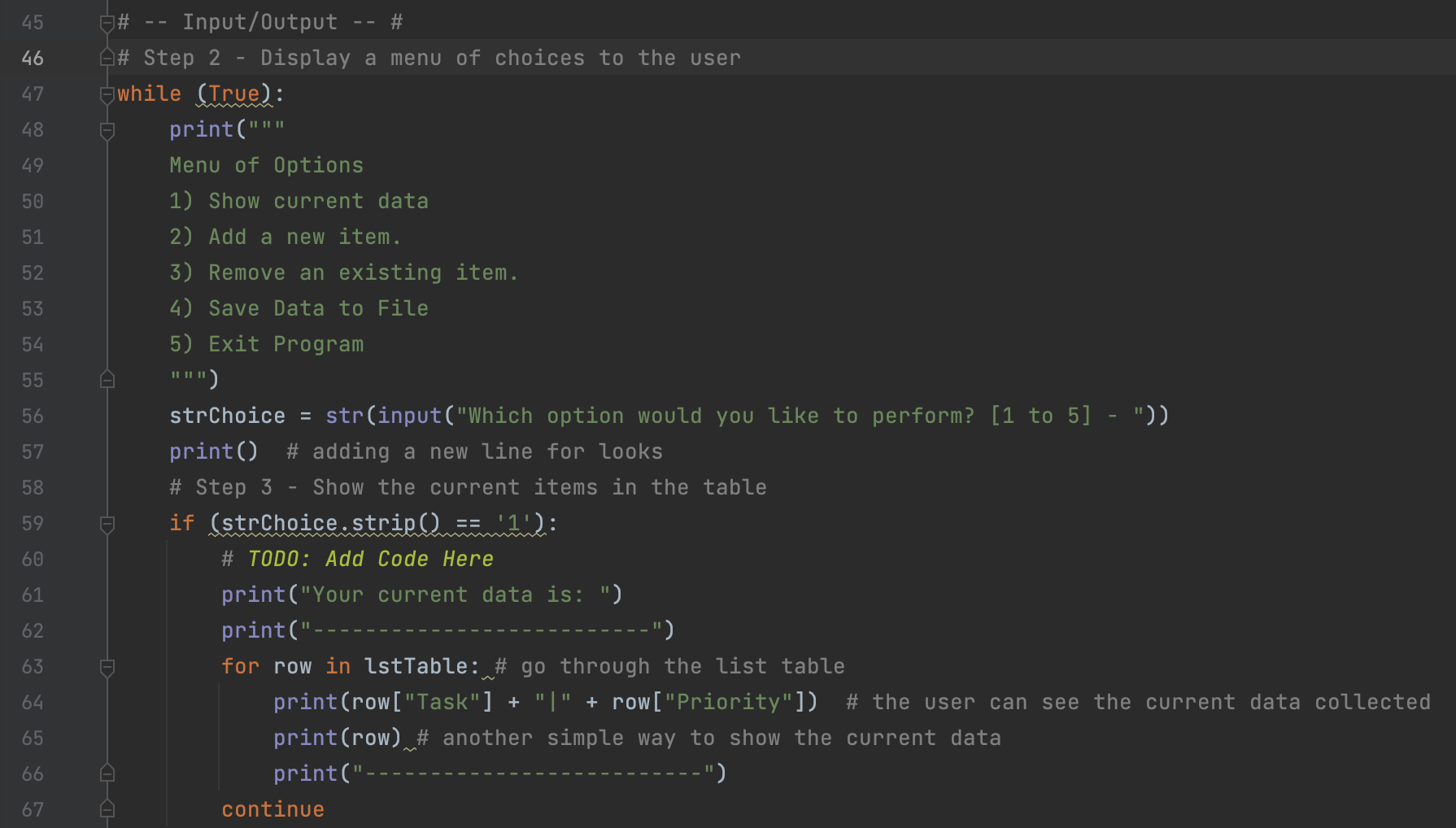


Figure 3: Code to show the current data to the user (step 3)

The following steps (3 and 4) correspond to adding new tasks or deleting existing tasks respectively. To add information (Menu option 2), I used the variables *task\_str* and *priority\_str* to receive the information from the user with the *input()* function and then I incorporated the data in the form of dictionary as part of the list table, for which I used again the *append()* methods.

On the other hand, to delete information from the table (Menu option 3), the first thing I used was the string variable *“strData”* and with the *input()* function I asked the user to enter the value of the task wanted to delete. Once the task was typed, I incorporated the *For loop* with the purpose of going through the information in the list table so once the values matched, the row would be removed from the table, using the *remove()* methods. If the value was not within the list table, the program continues by displaying the menu options to the user again.

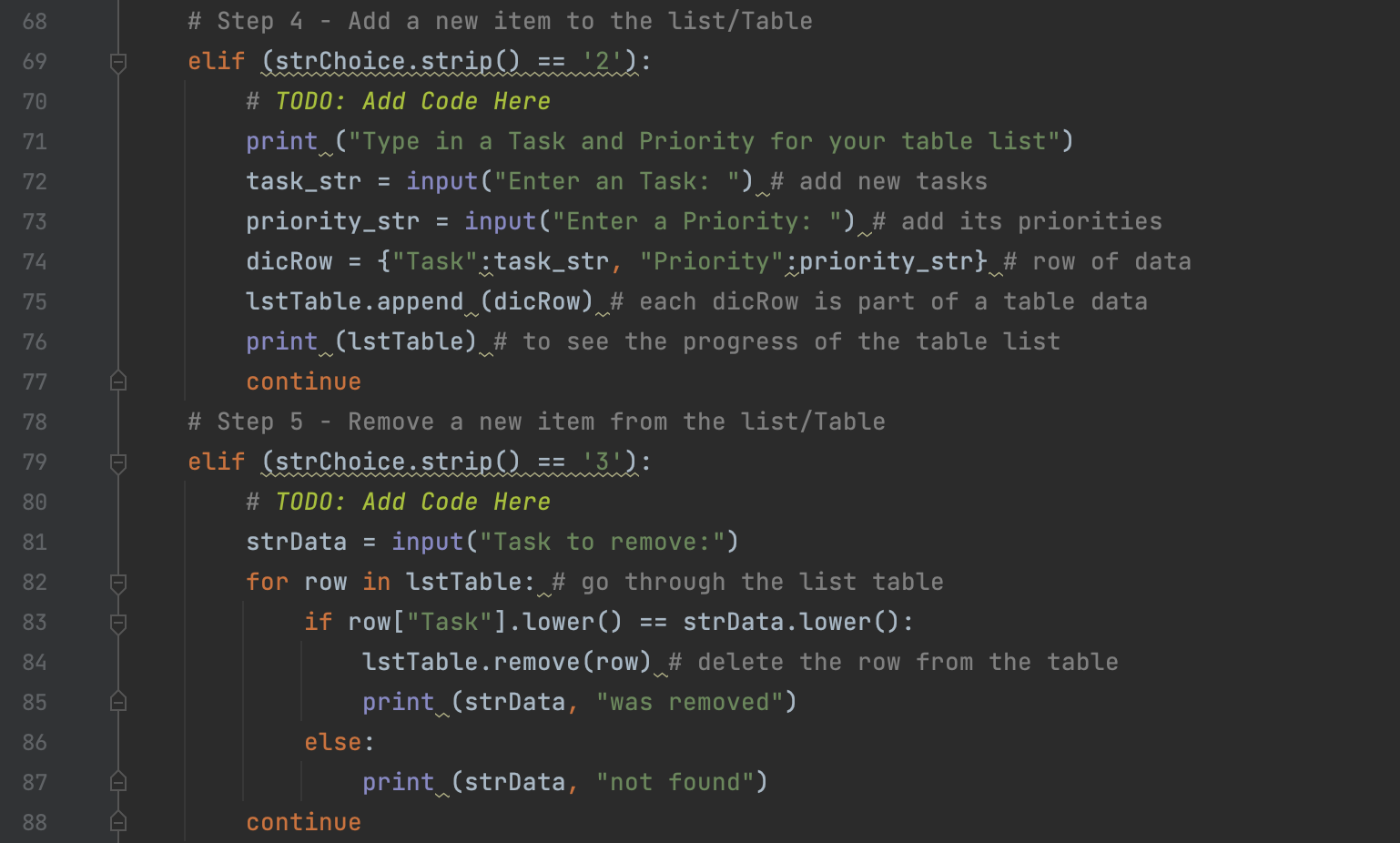


Figure 4: Code to add or remove data (step 4 & 5)

To write the block of code corresponding to option 4 of the menu, where the user must decide whether to save the information or not (I used the *strChoice* variable to receive input from the user), I used the *while*() function with the *if()* conditional. So, if the user decides to save the data, I used the *open()* function with the option *“w”* to add the new data in the plain text file where the information will be stored (I created a folder called 'Assignment05' where I also saved my script from PyCharm), then added the *For Loops* to be able to collect the data in the form of two-dimensional list (dictionary row) and leave it saved in the form of table in the .txt file, prior to close the program. I also concatenated the information using the operator “+”. In case the user does not decide to save the information, I only used the *print()* function to notify the user that the information was not saved and continue the program.

The block of code corresponding to option 5 of the menu was simpler, where the user must decide whether to exit the program or not (I used the *strChoice* variable to receive input from the user). In this case I only used the *if()* and *elif()* conditionals, therefore once the user makes the decision to quit “y”, the program ends (*break*).

Finally, from the step 2 until 7, it was used the *while()* function with the *if(), elif()* conditionals, allowing the user to repeat blocks as he or she progresses in the execution of the program and according to the decisions he or she makes. To condition the user to only choose between options 1 to 5, in the end I used the *“else”* statement with the *print()* function to deliver a message to him or her (Figure 5 and 6).

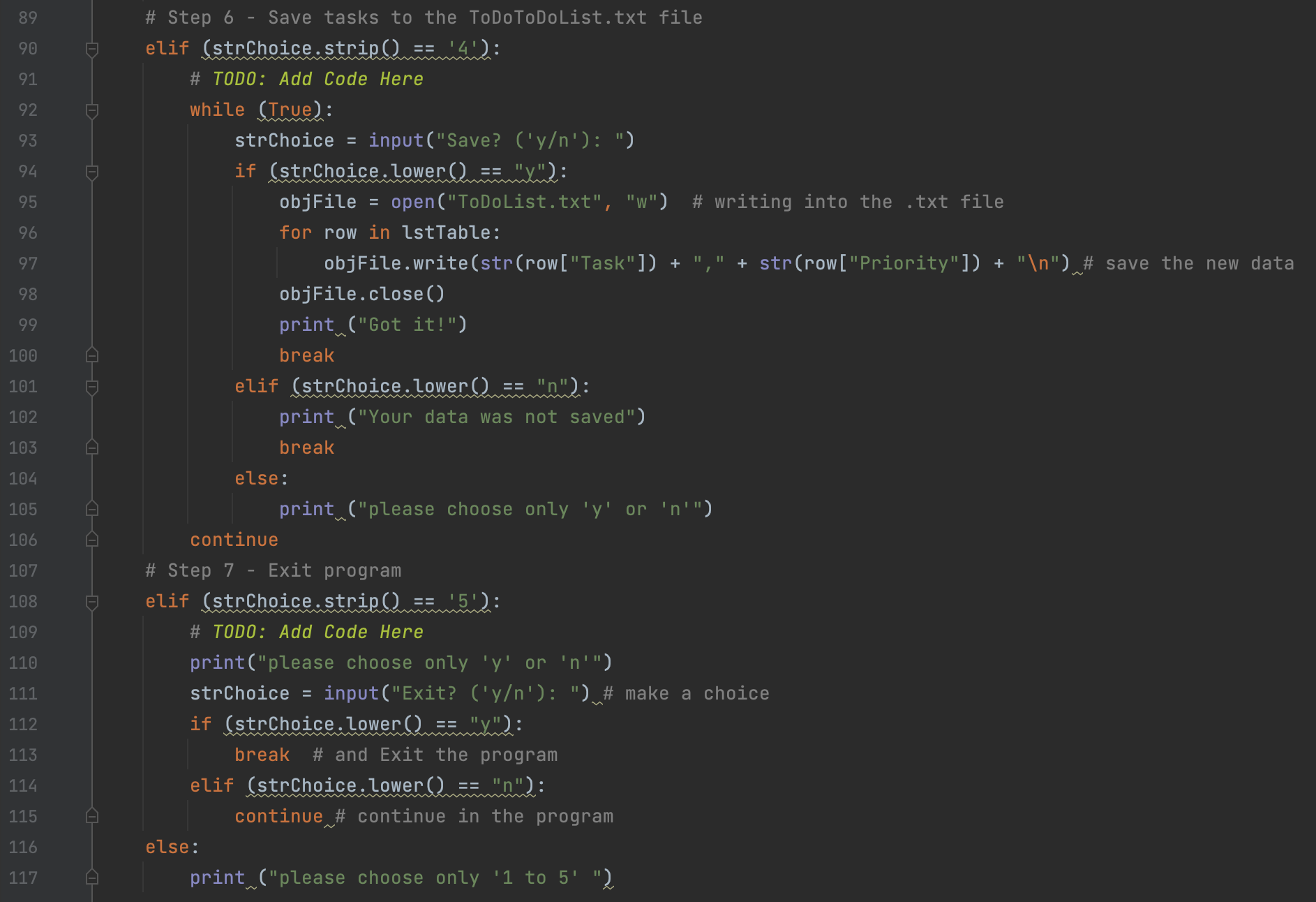


Figure 5:

***I.3 Running the code in PyCharm & Command Terminal***

Once I finished the code, I validated it in PyCharm and the Command Terminal. This ran as expected and the user can go through the menu options of the program as many times as he or she wants (Figure 6 and 7).

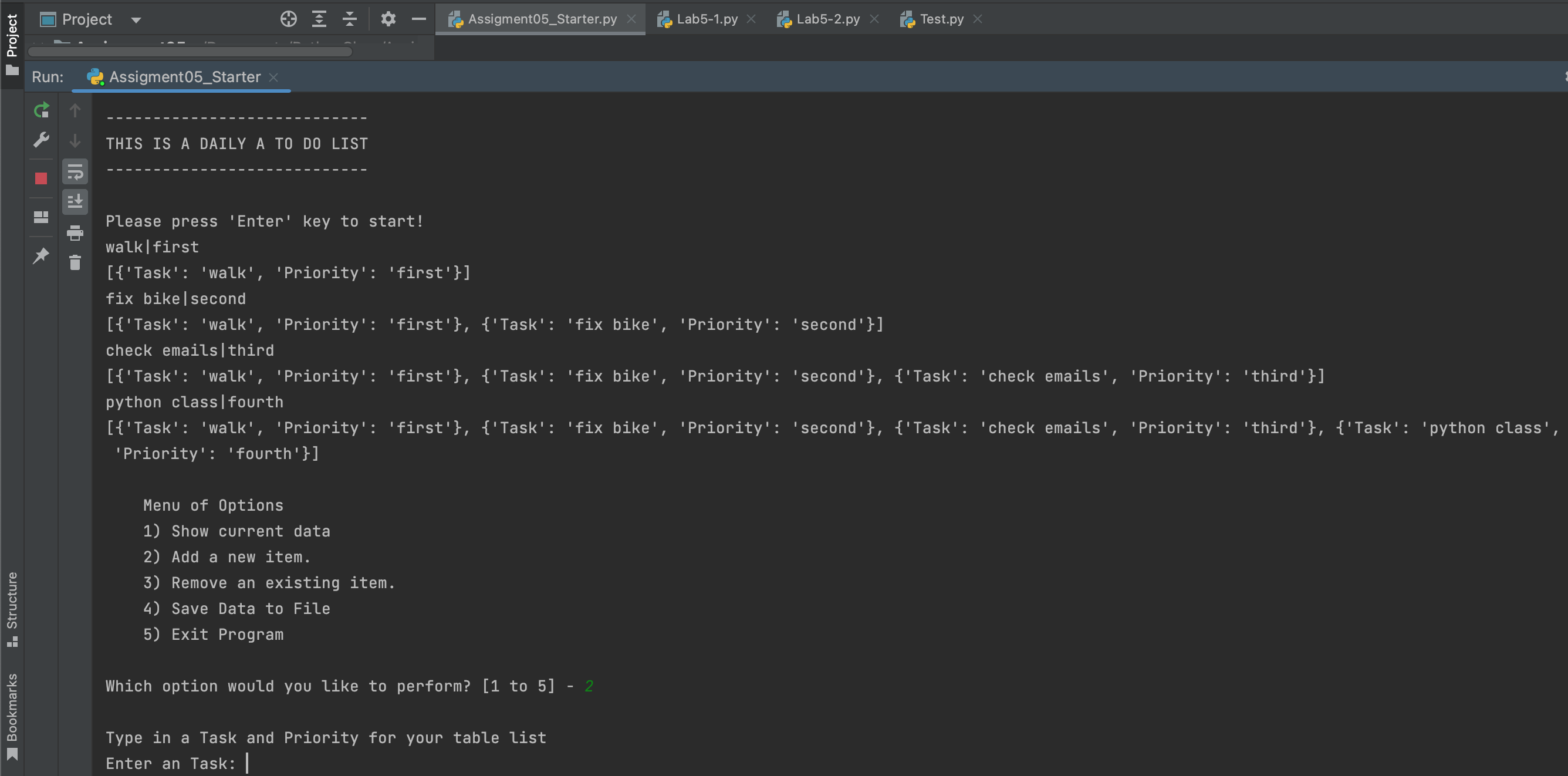


Figure 6: The code running in PyCharm

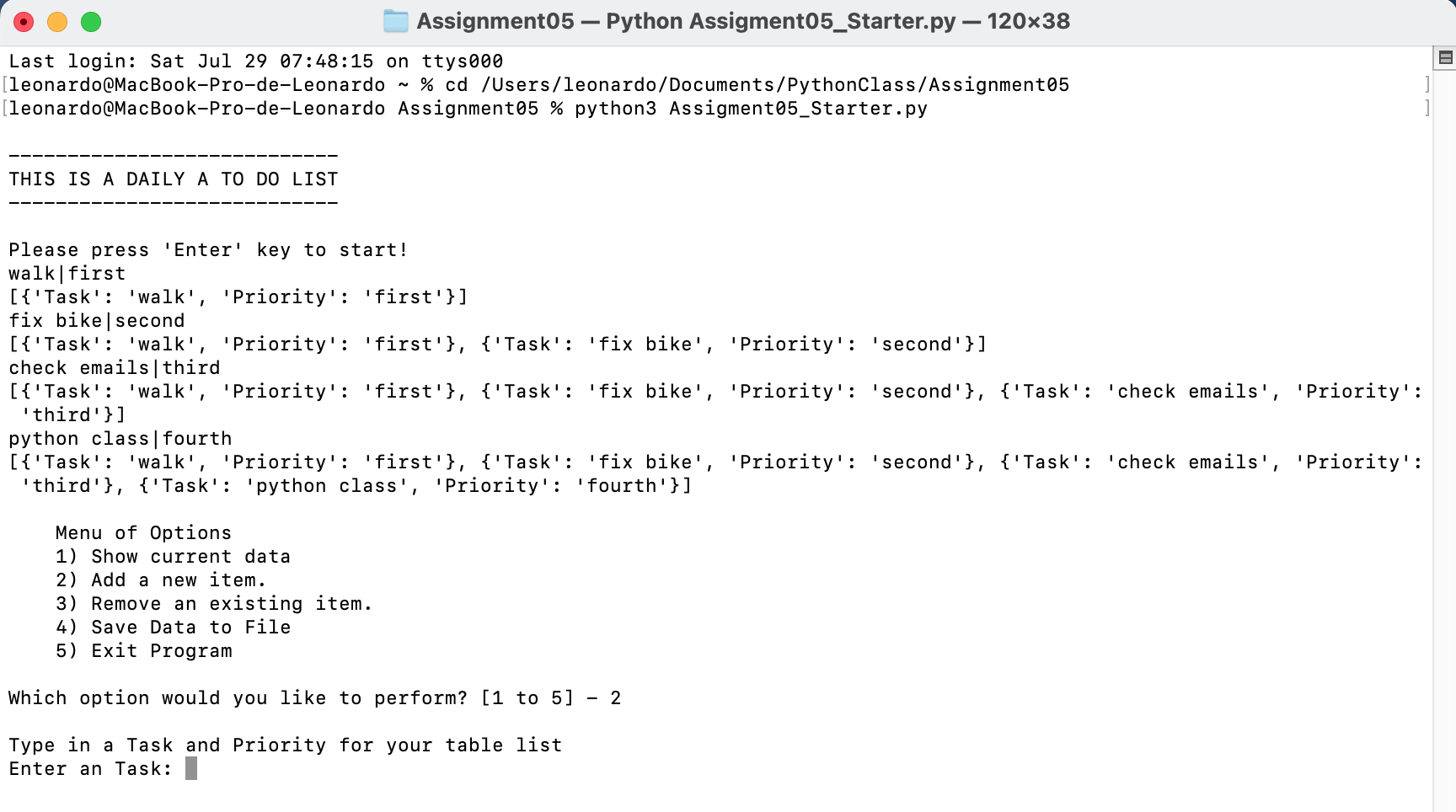


Figure 7: The code running in the Command Terminal

Finally, I opened the plain text file ToDoList.txt and it can be demonstrated that the code works by saving the information correctly (Figure 8).

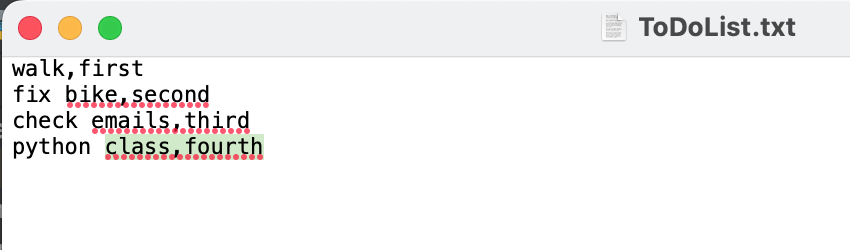


Figure 8: Checking the data in .txt file

# Summary

To achieve the goal of the assignment of this module, as I did in the previous ones, I followed the instructions given by the professor in the last class, reviewed the supplemental resources and read the chapter 5 of the textbook. In this occasion I completed the code by dividing it into different parts following the order given in the script, then I tested its correct execution, to finally run the full program. I used the list and dictionaries to store the collection of data in memory and then save it into the text file called ToDo list, to finally share the code in the GitHub repository. The program demonstrates my understanding of the tools reviewed in this module. However, despite the requested result is obtained with the program, I believe it can be further improved.