

Name : Leonardo Espinoza

Date : August 9, 2023

Course : Foundations of Programming, Python

Assignment 05

Github : <https://github.com/lesping/IntroToProg-Python>

The To Do List Program

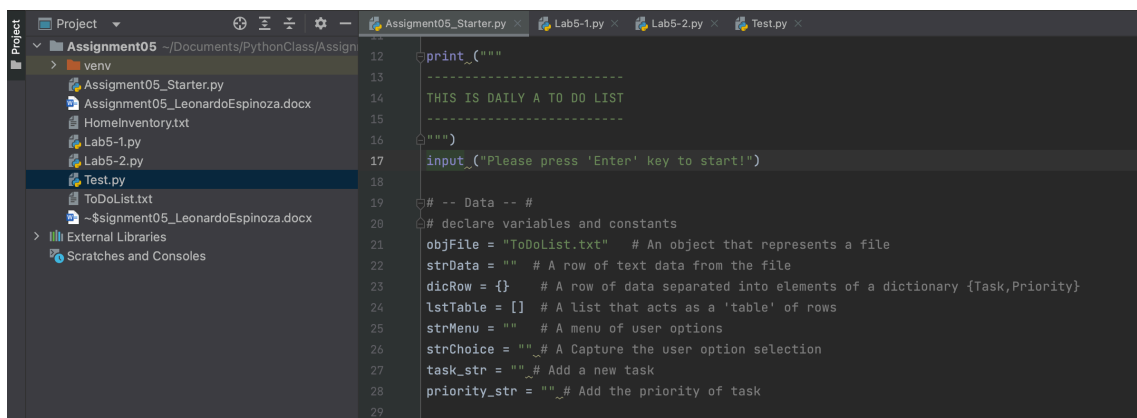
Introduction

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.

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

1.1 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).



```
12 print("""
13 -----
14 THIS IS DAILY A TO DO LIST
15 -----
16 """)
17 input('Please press \'Enter\' key to start!')
18
19 ## -- Data -- ##
20 # declare variables and constants
21 objFile = "ToDoList.txt" # An object that represents a file
22 strData = "" # A row of text data from the file
23 dicRow = {} # A row of data separated into elements of a dictionary {Task,Priority}
24 lstTable = [] # A list that acts as a 'table' of rows
25 strMenu = "" # A menu of user options
26 strChoice = "" # A Capture the user option selection
27 task_str = "" # Add a new task
28 priority_str = "" # Add the priority of task
29
```

Figure 1: Welcome phrase & declaring variables

1.2 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.

```
30 # -- Processing -- #
31 # Step 1 - When the program starts, load the any data you have
32 # in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
33 # TODO: Add Code Here
34 try:
35     objFile = open(objFile, "r")_# read the data we keep in the text file
36     for row in objFile:_# to loop through the rows of data
37         lstRow = row.split(",")
38         dicRow = {"Task": lstRow[0], "Priority": lstRow[1].strip()}
39         print(dicRow["Task"] + "|" + dicRow["Priority"])
40         lstTable.append(dicRow)_# managing the data in the memory
41         print(lstTable)
42     objFile.close()
43 except:
44     print("File 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).

```
45 # -- Input/Output -- #
46 # Step 2 - Display a menu of choices to the user
47 while (True):
48     print("""
49     Menu of Options
50     1) Show current data
51     2) Add a new item.
52     3) Remove an existing item.
53     4) Save Data to File
54     5) Exit Program
55     """)
56     strChoice = str(input("Which option would you like to perform? [1 to 5] - "))
57     print() # adding a new line for looks
58     # Step 3 - Show the current items in the table
59     if (strChoice.strip() == '1'):
60         # TODO: Add Code Here
61         print("Your current data is: ")
62         print("-----")
63         for row in lstTable:_# go through the list table
64             print(row["Task"] + "|" + row["Priority"]) # the user can see the current data collected
65             print(row)_# another simple way to show the current data
66             print("-----")
67         continue
```

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.

```
68      # Step 4 - Add a new item to the list/Table
69      elif (strChoice.strip() == '2'):
70          # TODO: Add Code Here
71          print_("Type in a Task and Priority for your table list")
72          task_str = input("Enter an Task: ")_# add new tasks
73          priority_str = input("Enter a Priority: ")_# add its priorities
74          dicRow = {"Task":task_str, "Priority":priority_str}_# row of data
75          lstTable.append(dicRow)_# each dicRow is part of a table data
76          print_(lstTable)_# to see the progress of the table list
77          continue
78      # Step 5 - Remove a new item from the list/Table
79      elif (strChoice.strip() == '3'):
80          # TODO: Add Code Here
81          strData = input("Task to remove:")
82          for row in lstTable:_# go through the list table
83              if row["Task"].lower() == strData.lower():
84                  lstTable.remove(row)_# delete the row from the table
85                  print_(strData, "was removed")
86              else:
87                  print_(strData, "not found")
88          continue
```

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*).

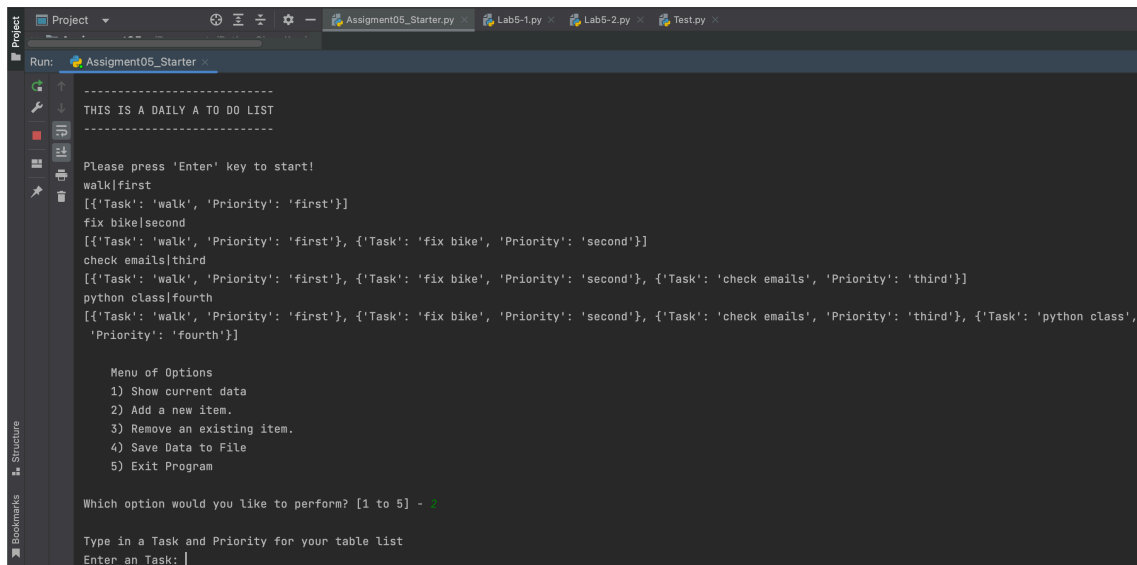
Finally, from the step 2 until 7, it was used the *while()* function with the *if()*, *elif()* conditionals, allowing the user to repeat blocks as he or she progresses in the execution of the program and according to the decisions he or she makes. To condition the user to only choose between options 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).

```
89 # Step 6 - Save tasks to the ToDoToDoList.txt file
90 elif (strChoice.strip() == '4'):
91     # TODO: Add Code Here
92     while (True):
93         strChoice = input("Save? ('y/n'): ")
94         if (strChoice.lower() == "y"):
95             objFile = open("ToDoList.txt", "w") # writing into the .txt file
96             for row in lstTable:
97                 objFile.write(str(row["Task"]) + "," + str(row["Priority"]) + "\n") # save the new data
98             objFile.close()
99             print("Got it!")
100             break
101         elif (strChoice.lower() == "n"):
102             print("Your data was not saved")
103             break
104         else:
105             print("please choose only 'y' or 'n'")
106             continue
107 # Step 7 - Exit program
108 elif (strChoice.strip() == '5'):
109     # TODO: Add Code Here
110     print("please choose only 'y' or 'n'")
111     strChoice = input("Exit? ('y/n'): ") # make a choice
112     if (strChoice.lower() == "y"):
113         break # and Exit the program
114     elif (strChoice.lower() == "n"):
115         continue # continue in the program
116     else:
117         print("please choose only '1 to 5' ")
```

Figure 5:

1.3 Running the code in PyCharm & Command Terminal

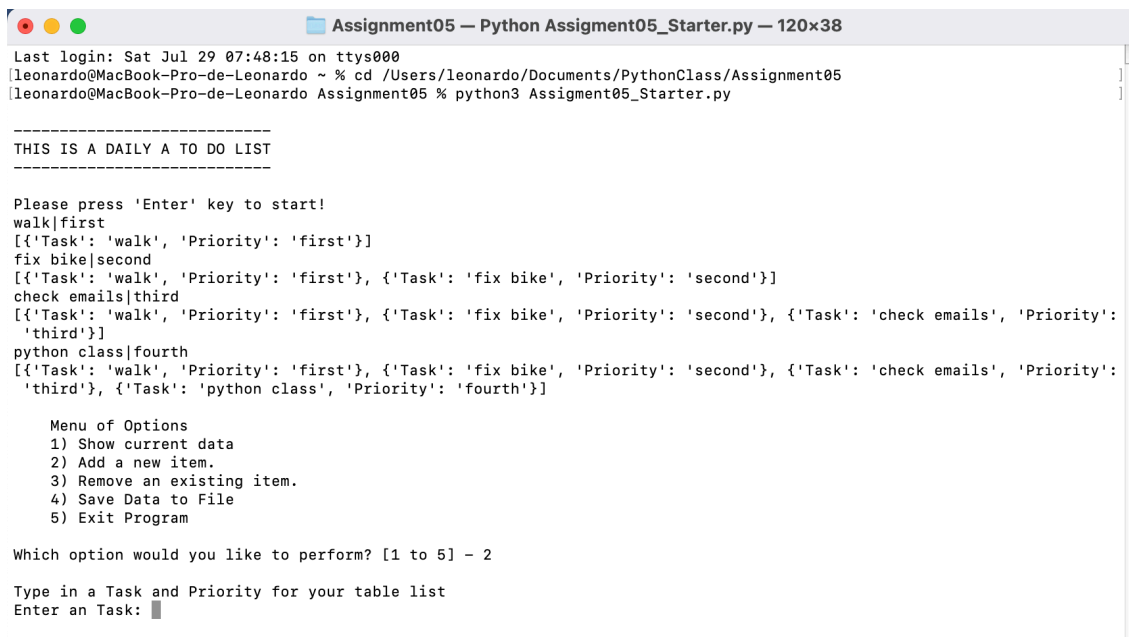
Once I finished the code, I validated it in PyCharm and the Command Terminal. This ran as expected and the user can go through the menu options of the program as many times as he or she wants (Figure 6 and 7).



The image shows the PyCharm IDE interface. The top toolbar includes icons for running and debugging. The 'Run' tab is active, displaying the output of the program. The code in the background is a Python script for a to-do list application. The output shows the program's execution flow, including task additions and a menu of options.

```
-----  
THIS IS A DAILY A TO DO LIST  
-----  
  
Please press 'Enter' key to start!  
walk|first  
[{'Task': 'walk', 'Priority': 'first'}]  
fix bike|second  
[{'Task': 'walk', 'Priority': 'first'}, {'Task': 'fix bike', 'Priority': 'second'}]  
check emails|third  
[{'Task': 'walk', 'Priority': 'first'}, {'Task': 'fix bike', 'Priority': 'second'}, {'Task': 'check emails', 'Priority': 'third'}]  
python class|fourth  
[{'Task': 'walk', 'Priority': 'first'}, {'Task': 'fix bike', 'Priority': 'second'}, {'Task': 'check emails', 'Priority': 'third'}, {'Task': 'python class', 'Priority': 'fourth'}]  
  
Menu of Options  
1) Show current data  
2) Add a new item.  
3) Remove an existing item.  
4) Save Data to File  
5) Exit Program  
  
Which option would you like to perform? [1 to 5] - 2  
  
Type in a Task and Priority for your table list  
Enter an Task: |
```

Figure 6: The code running in PyCharm



The image shows a terminal window titled 'Assignment05 — Python Assignment05_Starter.py — 120x38'. The terminal output is identical to the one in Figure 6, showing the program's execution flow and the user's input for option 2.

```
Last login: Sat Jul 29 07:48:15 on ttys000  
leonardo@MacBook-Pro-de-Leonardo ~ % cd /Users/leonardo/Documents/PythonClass/Assignment05  
leonardo@MacBook-Pro-de-Leonardo Assignment05 % python3 Assignment05_Starter.py  
  
-----  
THIS IS A DAILY A TO DO LIST  
-----  
  
Please press 'Enter' key to start!  
walk|first  
[{'Task': 'walk', 'Priority': 'first'}]  
fix bike|second  
[{'Task': 'walk', 'Priority': 'first'}, {'Task': 'fix bike', 'Priority': 'second'}]  
check emails|third  
[{'Task': 'walk', 'Priority': 'first'}, {'Task': 'fix bike', 'Priority': 'second'}, {'Task': 'check emails', 'Priority':  
'third'}]  
python class|fourth  
[{'Task': 'walk', 'Priority': 'first'}, {'Task': 'fix bike', 'Priority': 'second'}, {'Task': 'check emails', 'Priority':  
'third'}, {'Task': 'python class', 'Priority': 'fourth'}]  
  
Menu of Options  
1) Show current data  
2) Add a new item.  
3) Remove an existing item.  
4) Save Data to File  
5) Exit Program  
  
Which option would you like to perform? [1 to 5] - 2  
  
Type in a Task and Priority for your table list  
Enter an Task: |
```

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).



The image shows a text file named 'ToDoList.txt'. The file contains the following text:

```
walk,first  
fix bike,second  
check emails,third  
python class,fourth
```

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.