Rockey Aye

Instructor: Randal Root

IT FDN 100 B Su 18

August 20, 2018

**Assignment 6 Knowledge Document**

**Introduction**

This document shows the steps that were taken to perform Assignment 6. In addition, it includes the discoveries and lesson learned items relating to the concept of defining functions. Python script, “test.py”, is designed to start with loading the initial list of tasks and priorities in the ToDo.txt by defining the class and functions. Then, the program is intended to read the data from ToDo.txt and convert it into the dictionary table format by using the class and functions. After the conversion, the program executes the selection of 5 different options, which are independently and specifically designed to be performed and executed the specific sets of functions within the class. The key concept of Assignment 6 is to understand the use of the class and the functions along with the Python script layout format of data, processing, and presentation. (Root – Module 6) In addition, “test.py” is designed to load each row of additional data, load those data in the dictionary “row”, and displays the data while saving the data back to ToDo.txt, as needed. (Root – Module 6) Appendix section shows “test.py” Python script for Assignment 6.

**Topic**

After completing the module, it is important on the presentation of Python script layout formatting. In order to achieve the proper level of presentation, Assignment 6, “test.py”, is using to the data, processing, and presentation of Python formation. The Python script, “test.py”, also explores and demonstrates the concept of building, editing, and modifying the dictionary and list based on the user input under the function. Using “.open”, “.write”, and “.close” function, a user can store the new data back into the file. (Root – Module 6)

**Requirements and Understandings**

The basic requirements of the program are similar to Assignment 5; however, requirements are captured in the functions, which are grouped under a class. (Root – Assignment 6 Module)

1. Make a function for the code that loads the each rows of data you have in the ToDo.txt text file into a python Dictionary and adds it to a Python List.
2. Make a function for the code that displays the contents of the List to the user.
3. Make a function for the code that allows the user to Add or Remove tasks from the list, plus save the tasks in the List tasks-priorities using numbered choices.
4. Make a function for the code that saves the data from the table into the Todo.txt file when the program exits.
5. Make a Class to hold the functions.

My understanding on the requirements on Assignment 6 is similar to Assignment 5. In Assignment 5, “test.py” is designed by loading the existing data from ToDo.txt file. While loading, the data is transformed to the “dictionary” table format. The executions of the program are based on the available selections to perform the operation. Currently, those options are: show current data, add a new item, remove an existing item, save data to file, and exit program. Depending upon a user’s decision, the modified data is either saved or disregarded. Furthermore, as part of the requirement, I have introduced while not function to keep looping the entry until the user provides the response to prevent the blank response. In Assignment 6, all of these executions are defined under the multiple “functions”, which are grouped together under a “class”. In addition, the Python script is divided into data, processing, and presentation section.

**Summary of Python Script – Data Section**

First, I created a “test.py” for Assignment 6 under C:\\_PythonClass folder. The Python script was then divided into data, processing, and presentation section. In data section, I defined all the necessary variables for executing the program. Figure 1 shows the definition of the data that for the entire program.

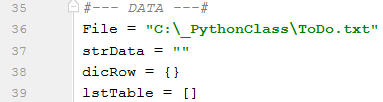


Figure 1. Data definition of “test.py”.

**Summary of Python Script – Processing Section**

The class structure with “@staticmethod”, which is shown in Figure 2, was created prior to laying out the logical structure of the pseudo code to outline the flow of the logical steps for “test.py”. “ToDoList(object)” was used to define the class of the entire “test.py” to control the flow and execution of the Python program in the data, processing, and presentation format. This allows and ensures the script abstraction and encapsulation to properly working on the script.



Figure 2. Processing – “class” method using “@staticmethod” to control the program.

In order to run the program logically and properly, following pseudo code was laid out in the processing section with each step of the pseudo code was defined underneath of the fucntion.

# Loading data from ToDo.txt in Python dictionary

# Display a menu of choices to the user

# Display current task ToDo list to the user

# Add a new item to the list/Table

# Remove a item from the list/Table

# Save tasks to the ToDo.txt file

# Exit program

First step is to load the existing data from ToDo.txt file in the dictionary table format. “readFile( )” was used to define as the function in the processing section of the Python script. Figure 2 shows the Python script setup using “readFile( )” function in the processing section. Within this function in the processing section, “task, priority” were created to split the existing data into the dictionary. Task was designed to represent the key of the data while priority was designed to represent the value of the data. The “disRow” was created for the row of dictionary data, which I used to represent under “lstTable” to form the dictionary table.

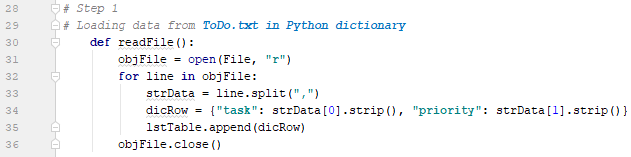


Figure 2. Processing – Script Loading ToDo.txt file into Python Dictionary.

For displaying the current tasks and each of their priority, “showList(message):” was assigned to define the complete function to show each row of the data existing in ToDo.txt. Thus, “print” function was used to print out and show the current condition of data. After the execution is completed, due to “continue”, it will execute “while” loop to a user for the next user’s decision. Figure 3 shows the script and 3b shows the result of the script.

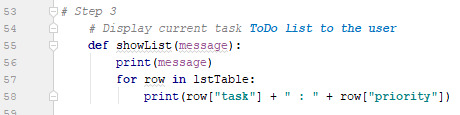


Figure 3. Processing – Python Script of Displaying Current Data.

For adding the new task and its priority to the current list, “addItem(message2):” was assigned to execute the function in the processing section. Using input function, “strTask” and “strPriority” were assigned to request the inputs from a user. Since “while not” function was used, the script is preventing any user blank response. In addition, “.strip( ).title( )” was used to strip out extra space and capitalize the first letter of the word when it prints out and save the data. Once both entries were entered, “dicRow” was assigned immediately to convert those new entries in the dictionary form by assigning with the following: {"task":strTask,"priority":strPriority}. After that, “.append” function was used to append into the original list in “lstTable”. Figure 4 shows the setup of the script.

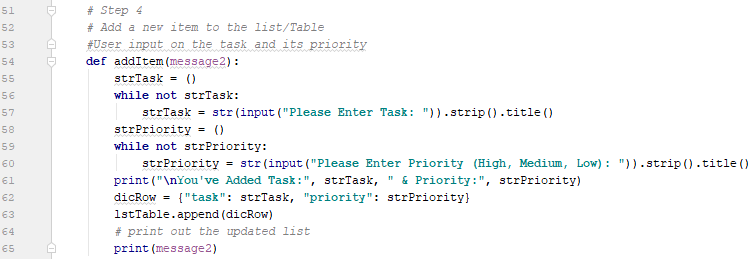


Figure 4. Processing – Python Script of Adding New Task and Priority.

For removing any existing task from the list, “removeItem(message3):” was created in the processing section of the script to perform the user’s desired task to be removed from the list. A Boolean function, “BlankRemoved = False”, and the input function, “strDelete”, with “while not” were set up. Since “while not” function attached to the input function of “strDelete” was used, the script is preventing any user blank response. I also created “.strip( ).title( )” to strip out extra space and capitalize the first letter of the word when it prints out and save the data. “BlankRemoved”, as the Boolean function, I designed a script to delete if and when a user enters a task that is matched in the list. In order to check the entire dictionary, using “while (intRow < len(lstTable)):” function, I designed to check every row in the data table. When the entry matches with one of the tasks in the list, it will delete the task from the list. Otherwise, the Program is designed to provide the “can’t find” message under the “if / else” and print function. Figure 5 shows the Python script setup.

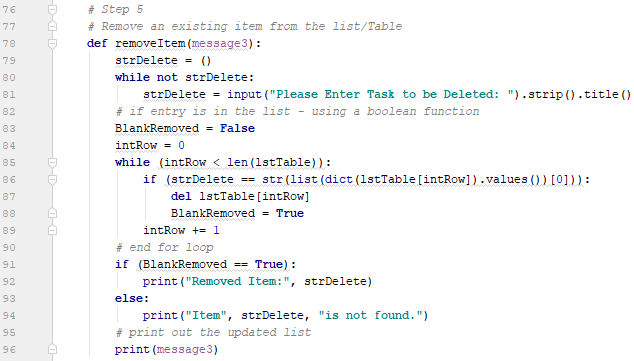


Figure 5. Processing – Python Script of Removing Task and Priority.

For saving the updated tasks and priorities into ToDo.txt, “saveFile( )” was created to define the function to save the data in the processing section of the Python file. I created “strSave” as the input function so that if a user decides to save the script is designed to execute “open”, “.write with ‘w’”, and “.close” operation to create a file, update, and save the data. Figure 6 shows the script of “if / else” statement under “saveFile( )” function for saving or disregarding the data.

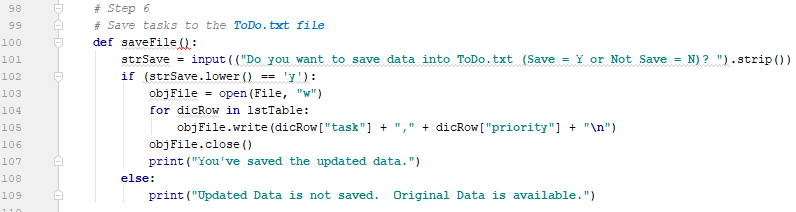


Figure 6. Processing – Python Script of Saving Task and Priority into ToDo.txt.

**Summary of Python Script – Presentation Section**

In order to extract data from ToDo.txt in the form of the list, I called the “ToDoList” that holds the function, “read.File( )”. Figure 7 shows the script. This generates the abstraction of the existing data from the file, which is tied to the function of displaying the current list.



Figure 7. Presentation – Data Extraction from ToDo.txt.

Immediately after, the data extraction, I put “while” loop under the presentation section because its main function is to present the option selections of 5 different functionalities that a user can operate. The “while” loop is designated to show the menu of choices by incorporating with if/elif/else statement, which is intended to either keep running the program or end the program due to a user’s decision. Figure 8 shows the concept of “while” loop to control the entire program to keep executing various functions / operations of a user’s choice until a user ends the program. “strChoice” with input function was used to designate the desired option.

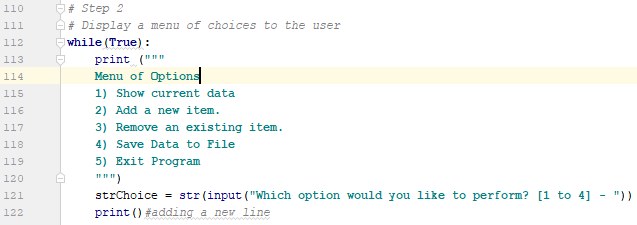


Figure8. Presentation – Concept of “while loop” for the entire “test.py”.

When a user selects Option 1 from the selection, the “while” loop was designed to execute when “strChoice” is equal to 1. Figure 9a shows the use of the class function, “ToDoList”, holding the function, “showList” with the print message to run the data extraction. Figure 9b shows the result of conversion of current data from ToDo.txt file by using the class and function.

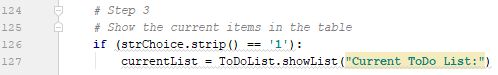


Figure 9a. Presentation – Option 1 Script of Data Extraction from ToDo.txt.

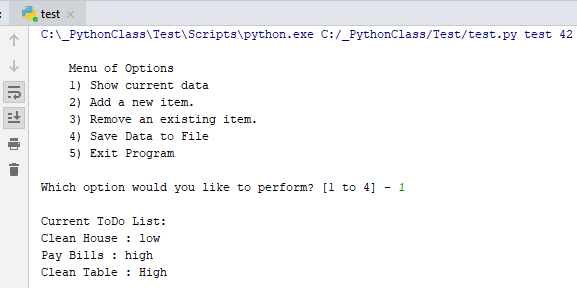


Figure 9b. Result of Option 1 Script.

For Option 2 on adding a new task and its priority, using “elif”, it was designed to equal to 2 in order to execute the following: newItem = ToDoList.addItem("\nUpdated ToDo List:"). This “newItem” is assigned to execute by calling the script defined in the class and its function, ToDoList.addItem shown in Figure 10a. The “for” loop will go through each row by row entry in terms of key – value combination. The script executes “while not” loop properly, which is part of “addItem” function defined in ToDoList class, since it keeps requesting a user’s input in order to prevent the empty response. Figure 10b shows the script result.

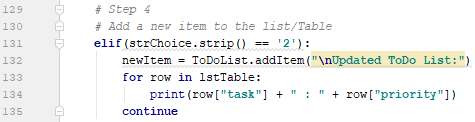


Figure 10a. Presentation – Option 2 Script of Adding Data.

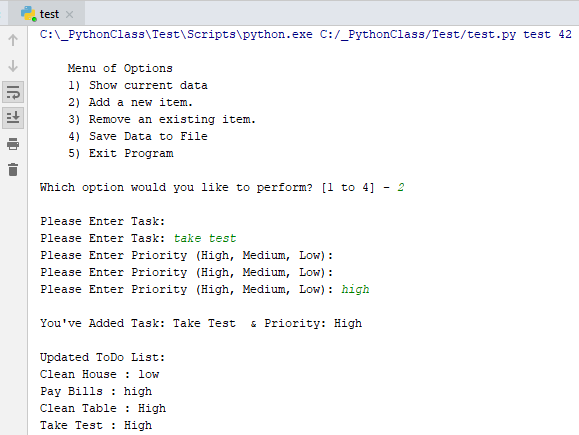


Figure 10b. Result of Option 2 Script.

For Option 3 of removing an existing task and its priority of a user’s choice, using “elif”, it was designed to equal to 3 in order to execute the following: delItem = ToDoList.removeItem("\nUpdated ToDo List AFTER Removed Task:"). This “delItem” is assigned to execute by calling the script defined in the class and its function, ToDoList.removeItem shown in Figure 11a. The “for” loop will go through each row by row entry in terms of key – value combination. The script executes “while not” loop properly, which is part of “delItem” function defined in ToDoList class, since it keeps requesting a user’s input in order to prevent the empty response. Figure 11b shows the script result.

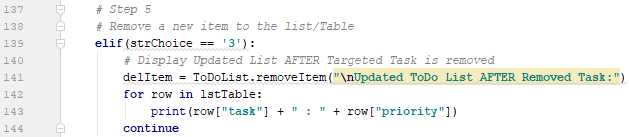


Figure 11a. Presentation – Option 3 Script of Removing Data.

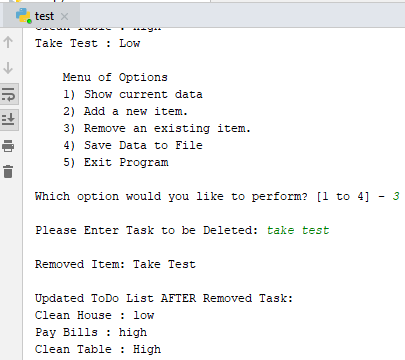


Figure 11b. Result of Option 3 Script.

For Option 4 to save the data into ToDo.txt file, using “elif”, it was designed to equal to 4 in order to execute the following: File = ToDoList.saveFile( ). This “File” is assigned to execute by calling the script defined in the class and its function, ToDoList.saveFile shown in Figure 12a. Figure 12b shows the script result.

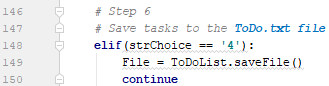


Figure 12a. Presentation – Option 4 Script of Removing Data.

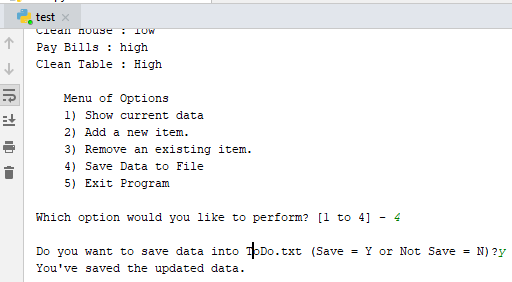


Figure 12b. Result of Option 4 Script.

For the final Option 4, it simply “break” the “while” loop when the user choice, which is assigned to “strChoice” is equal to 5. Figure 13a shows the script of “break” and Figure 13b shows its result.



Figure 13a. Presentation – Option 5 Script of “break” the “while” loop ending the Program.

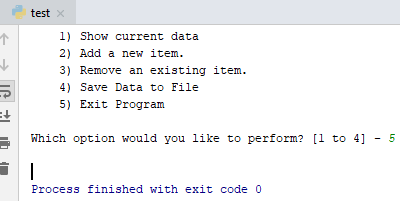


Figure 13b. Result of Option 5 Script.

**Test Case Scenario**

Following is the description of test case scenario for evaluating the Python Script for Assignment 5. The test case scenario was designed to align with the requirements of Assignment 5.

Step 0 is to load initially load the converted dictionary data from ToDo.txt.

Step 1 is to perform Option 2 to add “clean table” as the task and “medium” as its priority.

Step 2 is to perform Option 2 to add “buy grocery” as the task and “high” as its priority.

Step 3 is to perform Option 1 to see the latest task list.

Step 4 is to perform Option 3 to remove “buy grocery” and “high”.

Step 5 is to perform Option 4 to save the data.

Step 6 is to perform Option 5 to end the program.

Step 7 is to validate the updated list in ToDo.txt.

**Result of Test Case Scenario**

I have summarized the results and observations of the test case scenario in Table 1 with the screenshot. Validating the code improves the consistency and reliability of the code performance.

Table 1. Test Result and Validation of the Program.

|  |  |  |
| --- | --- | --- |
| **Step** | **Result Description** | **Result** |
| 0 | By selecting Option 1, the initial data was loaded successfully. |  |
| 1 | By selecting Option 2, the first entry of “clean table” and “medium” were made successfully. The current ToDo list was successfully updated with both entries. The first letter of each word was capitalized. |  |
| 2 | By selecting Option 2, the second entry of “buy grocery” and “high” were made successfully. The current ToDo list was successfully updated with both entries. The first letter of each word was capitalized. |  |
| 3 | By selecting Option 1, the latest ToDo list reflected all the updated entries. |  |
| 4 | By selecting Option 3, “buy grocery” is deleted from the updated ToDo dictionary list. It is by the latest updated ToDo list. |  |
| 5 | By selecting Option 4,The updated list was saved into ToDo.txt successfully when “y” was entered to save the data. |  |
| 6 | The program was ended successfully when Option 5 was selected. |  |
| 7 | ToDo.txt was updated successfully with the latest updated data. |  |

Work Cited

Dawson, Michael. *Python Programming for the Absolute Beginner*. Third Edition. Boston: Course Technology Cengage Learning, 2010. Print.

Root, Randal. "Assignment 6 Module." *IT FDN 100 B Su 18: Foundations Of Programming: Python.* UW PCE, Web. August 17, 2018. <https://canvas.uw.edu/courses/1243112>.

Root, Randal. "Python Module 05." *Python Module 06.* You Tube, Oct 1, 2016. Web. Aug 17, 2018. <https://www.youtube.com/watch?v=UigL0Cd4BRk&list=PLfycUyp06LG_Ui56yzznsHhfEosKZSahX>.

Appendix

*#-------------------------------------------------#  
# Title: Working with Dictionaries  
# Dev: Rockey Aye  
# Date: Aug 20, 2018  
# Change Log: New  
# Change Description: Rockey Aye, 8/19/2016, Create code to complete assignment 6  
# 1. Make a function for the code that loads the each rows of data  
# you have in the* ***ToDo.txt text file into a python Dictionary****# and adds it to a Python List.  
# 2. Make a function for the code that displays the contents of the List to the user.  
# 3. Make a function for the code that allows the user  
# to Add or Remove tasks from the list, plus save the tasks in the List  
# tasks-priorities using numbered choices.  
# 4. Make a function for the code that saves the data from the table  
# into the* ***Todo.txt file when the program exits.****# 5. Make a Class to hold the functions.  
#-------------------------------------------------#  
  
#--- DATA ---#*File = **"C:\\_PythonClass\ToDo.txt"**strData = **""**dicRow = {}  
lstTable = []  
  
*#--- PROCESSING ---#  
# Setting up class for the collection of functions***class** ToDoList(object):  
 @staticmethod  
 *# Step 1  
 # Loading data from* ***ToDo.txt in Python dictionary* def** readFile():  
 objFile = open(File, **"r"**)  
 **for** line **in** objFile:  
 strData = line.split(**","**)  
 dicRow = {**"task"**: strData[0].strip(), **"priority"**: strData[1].strip()}  
 lstTable.append(dicRow)  
 objFile.close()  
  
 *# Step 2  
 # Display a menu of choices to the user  
 # Part of Presentation Section  
  
  
 # Step 3  
 # Display current task* ***ToDo list to the user* def** showList(message):  
 print(message)  
 **for** row **in** lstTable:  
 print(row[**"task"**] + **" : "** + row[**"priority"**])  
  
 *# Step 4  
 # Add a new item to the list/Table  
 #User input on the task and its priority* **def** addItem(message2):  
 strTask = ()  
 **while not** strTask:  
 strTask = str(input(**"Please Enter Task: "**)).strip().title()  
 strPriority = ()  
 **while not** strPriority:  
 strPriority = str(input(**"Please Enter Priority (High, Medium, Low): "**)).strip().title()  
 print(**"\nYou've Added Task:"**, strTask, **" & Priority:"**, strPriority)  
 dicRow = {**"task"**: strTask, **"priority"**: strPriority}  
 lstTable.append(dicRow)  
 *# print out the updated list* print(message2)  
  
 *# Step 5  
 # Remove an existing item from the list/Table* **def** removeItem(message3):  
 strDelete = ()  
 **while not** strDelete:  
 strDelete = input(**"Please Enter Task to be Deleted: "**).strip().title()  
 *# if entry is in the list - using a boolean function* BlankRemoved = **False** intRow = 0  
 **while** (intRow < len(lstTable)):  
 **if** (strDelete == str(list(dict(lstTable[intRow]).values())[0])):  
 **del** lstTable[intRow]  
 BlankRemoved = **True** intRow += 1  
 *# end for loop* **if** (BlankRemoved == **True**):  
 print(**"\nRemoved Item:"**, strDelete)  
 **else**:  
 print(**"Item"**, strDelete, **"is not found."**)  
 *# print out the updated list* print(message3)  
  
 *# Step 6  
 # Save tasks to the* ***ToDo.txt file* def** saveFile():  
 strSave = input((**"Do you want to save data into ToDo.txt (Save = Y or Not Save = N)? "**))  
 **if** (strSave.lower() == **'y'**):  
 objFile = open(File, **"w"**)  
 **for** dicRow **in** lstTable:  
 objFile.write(dicRow[**"task"**] + **","** + dicRow[**"priority"**] + **"\n"**)  
 objFile.close()  
 print(**"You've saved the updated data."**)  
 **else**:  
 print(**"Updated Data is not saved. Original Data is available."**)  
  
 *# Step 7  
 # Exit Program - Part of Presentation Section  
#-------------------------------  
  
#--- PRESENTATION ---#  
# Step 1 - Extract / Open File using open, append, and close*ToDoList.readFile()  
  
*# Step 2  
# Display a menu of choices to the user***while**(**True**):  
 print (**"""  
 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  
 """**)  
 strChoice = str(input(**"Which option would you like to perform? [1 to 4] - "**))  
 print()*#adding a new line  
  
 # Step 3  
 # Show the current items in the table* **if** (strChoice.strip() == **'1'**):  
 currentList = ToDoList.showList(**"Current ToDo List:"**)  
  
 *# Step 4  
 # Add a new item to the list/Table* **elif**(strChoice.strip() == **'2'**):  
 newItem = ToDoList.addItem(**"\nUpdated ToDo List:"**)  
 **for** row **in** lstTable:  
 print(row[**"task"**] + **" : "** + row[**"priority"**])  
 **continue** *# Step 5  
 # Remove a new item to the list/Table* **elif**(strChoice == **'3'**):  
 *# Display Updated List AFTER Targeted Task is removed* delItem = ToDoList.removeItem(**"\nUpdated ToDo List AFTER Removed Task:"**)  
 **for** row **in** lstTable:  
 print(row[**"task"**] + **" : "** + row[**"priority"**])  
 **continue** *# Step 6  
 # Save tasks to the* ***ToDo.txt file* elif**(strChoice == **'4'**):  
 File = ToDoList.saveFile()  
 **continue  
  
 elif** (strChoice == **'5'**):  
 **break** *#and Exit the program*