Luis Valderrama

08/7/2021

IT FDN 110 A

Module 05

<https://github.com/lvalderr/IntroToProg-Python>

Lists and Dictionaries

# Intro

During the fifth module, I learned the concepts of working with Lists and Dictionaries. This document covers the steps taken to create a script named, <Assignment05_Starter.py> (Python script), designed to prompt the user to select from a five-option menu and execute the program based on the selection. The three options are 1) Show Current Data, 2) Add a New Item, 3) Remove an Existing Item 4) Save Data to File (.txt file) and 5) Exit Program. The script is designed to run in PyCharm and Command Prompt, applying concepts learned throughout this module.

# 1.0. Creating the Assignment05\_Starter.py Code Script

The following steps were taken to create the Python Script using PyCharm as the Integrated Development Environment (IDE).

## Script Header and Objective

As mentioned in prior documents, the first part of the script outlines not only the title, description, date, and change log. But it also provides the reader with a high-level objective of the script and what it is intended to do. (Figure 1.1)

# ---------------------------------------------------------------------------------------------------------------------#  
# Title: Assignment05\_Starter.py  
# Description: Working with Dictionaries and Files  
# When the program starts, load each "row" of data  
# in "ToDoToDoList.txt" into a python Dictionary.  
# Add the each dictionary "row" to a python list "table"  
# Change Log: (Who, When, What)  
# LValderrama, 2021-07-22, Created File  
# LValderrama, 2021-07-25, Modified the script to capture user input  
# LValderrama, 2021-08-01, Modified the script to enable removing tasks from list and corrected saving to txt file "bug"  
# ---------------------------------------------------------------------------------------------------------------------#  
  
# Objective:  
# The Assignment05.py script is designed to:  
# 1. Manage a .txt file named "*ToDo.txt" that contains two columns of data, "Task" and "Priority."*# 2. Load the columns into a Python Dictionary object. Each dictionary object represents one row of data,  
# and these rows must be added to a Python List object to create a table of data.

***Figure 1.1: Script Header and Objective***

## Pseudo-Code

Before developing the script, the steps are outlined in the form of Pseudo-Code (Figure 1.2) to help translate the objective into the programing code. In this example, there are seven core steps that may expand into sub-steps as the script develops.

# -- Data -- #  
# declare variables and constants

# -- Processing -- #  
# Step 1 - When the program starts, load any data you have  
# in a text file called ToDoList.txt into a python list of dictionaries rows   
# Create the table

# -- Input/Output -- #  
# Step 2 - Display a menu of choices to the user

# Step 3 - Show the current items in the table

# Process the data

# Step 4 - Add a new item to the list/Table

# Get user input

# Step 5 - Remove a new item from the list/Table

# Step 6 - Save tasks to the ToDoToDoList.txt file

# Process the data into file

# Step 7 - Exit program

***Figure 1.2: Pseudo-Code***

## Declaring Variables and Constants

As best practice, the variables and constants are declared before inserting the codes. (Figure 1.3)

# -- Data -- #  
# declare variables and constants  
strChoice = "" # A Capture the user option selection  
dicRow = {} # A row of data separated into elements of a dictionary {Task, Priority}  
lstTable = [] # A list that acts as a 'table' of rows  
strFile = "ToDoList.txt" # A row of text data from the file  
objFile = None # An object that represents a file  
strMenu = "" # A menu of user options

***Figure 1.3: Declaring Variables and Constants***

## Loading Data from Text File to the Table

When the program starts it opens the ToDoList.txt and loads the two columns content into a list held in a table named lstTable.

# -- Processing -- #  
# Step 1 - When the program starts, load any data you have  
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)  
lstTable = []  
objFile = open(strFile, "r") # open the text file and read the data  
for row in objFile: # when the data returns  
 lstRow = row.split(",") # split the data into a list  
 dicRow = {"Task": lstRow[0], "Priority": lstRow[1].strip()} # put in the keys and extract values  
 lstTable.append(dicRow) # append to the table itself  
objFile.close() # closing the file is best practice

***Figure 1.4: Loading the data from text file to a table***

## 1.5 Menu of Options

In this assignment, the script initiates with a menu that offers five choices. 1) Show current tasks, 2) Add a new task, 3) Remove an existing task, 4) Save data to file, and 5) Exit program. To create this menu: (Figure 1.5)

1. Insert a block of **print()** function displaying the menu title, and menu-option.
2. A variable, **strChoice**, followed by **input()** function is added to prompt the user to type the numerical values 1 – 5, and depending on the choice the program performs a specific task.
3. The Menu of Options is contained within a **While Loop** to cycle the selection process until the user selects the break option that stops the loop.

while (True):  
 print("""  
 Menu of Options  
 1) Show current tasks  
 2) Add a new task

3) Remove an existing task  
 4) Save data to file  
 5) Exit program  
 """)  
 strChoice = str(input("Which option would you like to perform? [1 to 5] - "))  
 print() # adding a new line for looks

***Figure 1.5: Program Menu***

## Show Current Tasks (Menu Option 1)

In this section we examine the programing to execute option 1 from the Menu of Options (Figure 1.6)

1. The conditional function **if** works together with the **while(True)** loop. In this example, if the user types “1” in the **strChoice** variable, the program processes the data by opening the ToDoList.txt file and read the file as indicated by **“r”.**
2. When the data returns, the program splits the data into a list.
3. The dictionary keys, “Task” and “Priority” in the variable, **dicRow** extract the values.
4. The data table in the variable, **lstTable.append(dicRow)** is appended with new data.
5. The print has been formatted to **“un-pack”** the list into individual variables with the code {**print(lstRow[0], lstRow[1].strip()}**.

# Step 3 - Show the current items in the table  
if (strChoice.strip() == '1'):  
 for objRow in lstTable: # Displaying rows below in the form of a vertical list  
 print(objRow["Task"] + ',' + objRow["Priority"]) # Unpacking  
 continue

***Figure 1.6: Show Current Data (Menu Option 1)***

## Add a New Task (Menu Option 2)

In this section we examine the programing to execute option 2 from the Menu of Options (Figure 1.7)

1. The conditional function **if** works together with **while(True)** loop. In this example, if the user types “2” in the **strChoice** variable, the program **print()** a message prompting the user to enter data in the dictionary key, **Task** by way of an **input()** function. The program pauses until the user enters data and presses Enter.
2. Then the program prompts the user to enter the second **input()** to the second dictionary key, **Priority** by way of an **input()** function and pauses until the user enters data and presses Enter.
3. The program holds the dictionary items in the keys, **Task** and **Priority**, within the variable, **dicRow**
4. The **lstTable** is appended with the **append** function to incorporate the new data. The lstTable continues to add data as an **append** so the data already in the file is not overwritten and instead the new data will be added.

# Step 4 - Add a new item to the list/Table  
elif (strChoice.strip() == '2'):  
 print("\nType a Task and Priority")  
  
 # Get user input  
 Task = str(input("Enter a New Task: ")).strip() # User enters task by way of input function  
 Priority = str(input("Enter Priority (Top, Med, Low): ")).strip() # User enters task by way of input function  
 dicRow = {"Task": Task, "Priority": Priority} # Build new dictionary row  
 lstTable.append(dicRow) # the lstTable is appended with data added by the user  
 print('\nYour Current Tasks are:')  
 for objRow in lstTable:  
 print(objRow["Task"] + ',' + objRow["Priority"]) # Unpacking the data entered by user  
 continue

***Figure 1.7: Displaying Data (Menu Option 2)***

## Remove an Existing Task (Menu Option 3)

In this section we examine the programing to execute option 3 from the Menu of Options. (Figure 1.8)

1. The conditional function **elif** works together with the **while(True)** loop. In this example, if the user types “4” in the **strChoice** variable, the program prompts a secondary question or condition by way of a variable, **strSavetoFileInput** = (“Enter ‘y’ or ‘n’: “).
2. The program engages a secondary loop. In this case **While (len)** loop.
3. The program displays a list of the current items in the **lstTable**.
4. An **input()** function allows the user to enter the name of the “Task” to remove.
5. Once the item matches input against the lstTable then the program removes the item from the list through lstTable.**remove**(task) function and displays a message confirming the removal. However, this does not the item has been deleted from the tax file. The user would need to proceed to option 4 rom the Menu of Options to save the updated **lstTable** to the text file.
6. If the item is not found in the list, then the program displays a message indicating the items does not exist in the **lstTable**.

**Note:** The program is designed to give the option to the user to break the loop and return to the primary loop to access the Menu of Options and move on to the next option.

# Step 5 - Remove a new item from the list/Table  
elif (strChoice.strip() == '3'):  
 while len(lstTable) > 0:  
 print("Current Task list:\n", "\n".join([f"\t{item['Task']}" for item in lstTable])) # Display current data  
 term = input("Type the Task you want to delete or type 'exit' to return to Menu of Options: ")  
 if term.lower() == "exit": # Option for the user to exit the while len loop  
 break  
 for task in lstTable:  
 if term in task["Task"]: # If the task entered by the user matches the name in the list then proceed  
 print(f"Removing {task['Task']}...") # Print a message "item found" and processing removal  
 lstTable.remove(task) # The item is removed from the table  
 break  
 else:  
 print(f"Task not found in list: {term}") # Displays message indicating the task was not found!  
 continue

***Figure 1.8: Displaying Data (Menu Option 3)***

## 1.9 Saving Data in .txt File (Menu Option 4)

In this section we examine the programing to execute option 4 from the Menu of Options (Figure 1.9.a)

1. The conditional function **elif** works together with the **while(True)** loop. In this example, if the user types “4” in the **strChoice** variable, the program prompts a secondary question or condition by way of a variable, **strSavetoFileInput** = (“Enter ‘y’ or ‘n’: “).
2. **If** the user enters ‘n’, the conditional function activates the loop **break,** the program stops and displays a message indicating “Data not Saved!”.
3. **If** the user enters ‘y’, the conditional function:
   1. With the user of the **objFile** variablethe program **open()** the ToDoList.txt file from its location.
   2. The program **write()** the **dicRow**.
   3. The program closes the .txt file.

# Step 6 - Save tasks to the ToDoToDoList.txt file  
elif (strChoice.strip() == '4'):  
 print('\nWould you like to save your data?')  
 strSaveToFileInput = input("Enter 'y' or 'n': ")  
 if (strSaveToFileInput == 'n'): # Conditional number 1 which will not save data to txt file  
 print('Data not saved!')  
 if (strSaveToFileInput == 'y'): # Conditional number 2 which will save data to txt file  
 # Process the data into file  
 objFile = open(strFile, "w") # "a" to append the file and not overwrite the exiting data  
 for row in lstTable:  
 objFile.write(row["Task"] + "," + row["Priority"] + "\n") # Unpacking  
 objFile.close() # File close, best practice  
 print('\nYour data is saved to:', strFile)  
 continue

***Figure 1.9.a: Saving Data in ToDoList.txt File***

1. To view the saved data simply open the .txt file. (Figure 1.9.b)

A screenshot of a computer

Description automatically generated

***Figure 1.9.b: View of the Assignment05\_Sstarter.py***

## 1.9.1 Exit the Program (Option 5)

In this section we examine the programing to execute option 4 from the Menu of Options (Figure 1.9.1)

1. The conditional function **elif** works together with the **while(True)** loop. In this example, if the user types “5” in the **strChoice** variable, the program prints a “Thank you!” message.
2. The script includes one last **input()** function displaying the message “Press Enter to End Program”. This is primarily done so the program does not close “disappears from the screen” in Command Prompt immediately after the user enters option “5”.
3. At this point the loop **break** and the program ends.

# Step 7 - Exit program  
elif (strChoice.strip() == '5'):  
 print('Thank you!')  
 EndProgram = input('\n(Press Enter to End Program)')  
 break # and Exit the program

***Figure 1.9.1: View of the Assignment05\_Sstarter.py***

# 2.0. Running the Python Script

The script can be executed in PyCharm and Command Prompt.

## 2.1. PyCharm

1. Open the file in PyCharm.
2. PyCharm displays the Menu of Option.
3. Enter one option from 1 - 5 and press “Enter”.
4. In this example, select “1”.
5. View the items in the .txt file. (Figure 2.1.1)

Text

Description automatically generated

***Figure 2.1.1: View of program running in PyCharm while executing Option 1***

1. Select option “2”
2. Enter the task and press “Enter”.
3. Enter the priority and press “Enter”. (Figure 2.1.2)

Text

Description automatically generated

***Figure 2.1.2: View of program running in PyCharm while executing Option 2***

1. Select option “3”.
2. The program displays the list of tasks to choose from.
3. Enter the name (or part of the name) to delete, and press “Enter”.
4. Then type ‘exit’ to exit the remove data program.

Text

Description automatically generated

***Figure 2.1.3: View of program running in PyCharm while executing Option 3***

1. Select option “4”.
2. To test the program, I enter ‘n’.
3. The program does not save the data to the .txt file and displays a message indicating “Data not Saved! (Figure 2.1.4)

Text

Description automatically generated

***Figure 2.1.4: View of program running in PyCharm while executing Option 4***

1. Alternatively, the user enters ‘y’ and view the confirmation message.
2. Press Enter to continue. (Figure 2.1.5)

Text

Description automatically generated

***Figure 2.1.5: View of program running in PyCharm while executing Option 4***

1. Select option #5
2. The program responds by displaying a “Thank You!” message.
3. And has one more prompt to press “Enter” to end. (Figure 2.1.6)

Text

Description automatically generated

***Figure 2.1.6: View of program running in PyCharm while executing Option 5***

## 2.2 Command Prompt

The script runs similarly in Command Prompt as it does in PyCharm. (Figure 2.2.1)

1. Open the file in Command Prompt by:
   1. Search/open Command Prompt, type “Python”.
   2. Then, go to the Assignment05\_Starter.py file location, select the file, and from the top Menu/Home, select “Copy path”, paste the path after “Python” in Command Prompt, and press “Enter”.
   3. Or simply open the file from its directory and should default open in Command Prompt.
2. Command Prompt displays the Menu of Options.
3. Enter one option from 1 - 5 and press “Enter”.
4. In this case, enter “1” to view the existing data in the .txt file. (Figure 2.2.1)

Text

Description automatically generated

***Figure 2.2.1: View of program running in Command Prompt while executing Option 1***

1. Select option “2”
2. Enter Task and press Enter.
3. Enter Priority and press Enter. (Figure 2.2.2)

Text

Description automatically generated

***Figure 2.2.2: View of program running in Command Prompt while executing Option 2***

1. Select option “3”
2. Press Enter to continue. (Figure 2.2.3)

Text

Description automatically generated

***Figure 2.2.3: View of program running in Command Prompt while executing Option 3***

1. In this case, to test the program I first enter ‘n’ and the program does not save the data in the .txt file and displays a message indicating “Data not Saved! (Figure 2.2.4)

Text

Description automatically generated

***Figure 2.2.4: View of program running in Command Prompt while executing Option 4 and testing to not save the new or removed data***

1. Alternatively, I enter ‘y’ which saved the data in the .txt file and displayed a confirmation message and the location of the file. (Figure 2.2.5)

Text

Description automatically generated

***Figure 2.2.5: View of program running in Command Prompt while executing Option 4 and testing to save the new or removed data***

1. Enter option #5
2. The program responds by displaying a “Thank You!” message.
3. And has one more prompt to press “Enter” to end. (Figure 2.2.5.)

Text

Description automatically generated

***Figure 2.2.5: View of program running in Command Prompt while executing Option 5***

# Summary

To recap, the fifth module introduced me to working with Lists and Dictionaries. The example created for this assignment, Assignment05\_Starter.py is the result of steps taken to develop a script designed to prompt the user to select from a five-option menu and execute the program based on the selection. The program successfully allows the user to enter, display, delete, and save the data in a .txt file. The program runs in PyCharm and Command Prompt and considers the concepts and best practices learned in this module.