Yuting Lin

Nov. 18, 2020

Introduction to Python Scripts

Assignment05

# Storing the User Input Data as Dictionary data in a Table (List)

## Introduction

### Dictionary

Other than the List, Tuple, and String that was introduced for the type of collection of data, Dictionary allows the data to be stored with subscripts as “keys” (characters), in another word, the data is stored in pairs (a word and its definition). The dictionary pair is referred as key-value. A dictionary is indicated with curly brackets: {}, and can be declared as “key”:value.

#### Difference between List and Dictionary

A List has subscripts as indexes (numerical) and Dictionaries has subscripts as keys (charactors). Usually the dictionary are used as individual rows in a table, and can be collectively stored in a “Table,” which usually is declared as a list.

#### How to read data from a Dictionary (and a list)

The data stored as dictionary can be read by methods such as items () that unpack items in the dictionary to key-value pairs, values() that only pulls out the values, and keys() that only access the keys of the dictionaries. The data in list can be called out using indexes such as list[0], where the 0 is the index noting the first data stored in the list.

### Some operators and methods

#### The “in” operator

The in operator searches through the characters (data) and look for a match. It is often associated with a for loop as a conditional statement.

#### .split() attribute for Strings

The .split() attributes split the data in the list into individual values and can later be re-declared as dictionary (or other type of collection of data) and re-stored.

#### .remove() attribute for lists

To remove an entire row from the list, the.remove() attribute is used instead of the .pop(), which only allow the removal of an index.

### Improving the script

#### Separation of Concerns

Adding the “Data,” “Processing,” and “Presentation (Input/Output)” allows the datafile to be better organized for the other developers. While it is hard to separate the presentation with the processing, one can use the functions to better organize the script. The function can be load into the memory (included in the “processing” section), but not carried out until the statement is indicated (the “presentation” section).

#### Script Templates

Another way to organize the code is creating script templates that allows the formatting to be consistent across the different scripts. One place to use the script template is the heading/information section at the beginning of the code.

#### Error Handling with Try-Except

Error Handling allows the script to better handle errors that is input by the user. It can help provide more general error statements to the error and trap the errors without interrupting the flow of the program. It also allows the error messages to be customized by the developer so it can be more direct and understandable by the user.

#### GitHub

GitHub is a cloud backup that allows the developer to store their scripts in the cloud and share with others. It also allows the changes to be uploaded in portions so different versions can be stored.

## A Python Script Example that stores user input data as dictionary in a Table (List)

Here is an example of a python script. Note that there is a pseudo-code that is included at the beginning of the script to indicate the Developer, date, and logging the changes. This portion is not included in this document for it does not provide function that is executed by Python. The goal of this script is to provide the user with a list of options, and let the user decide which option to proceed with. The program will first look for text files that is already being created and contains information (Step #1), then provide the users with a menu of options as to what they would like to do with the file (Step #2). If there was no pre-existing file, the script will create one when the user starts to write information into the file (Step #4). The user can check what is in the file (Step #3) and remove selected data from the file (Step #5). The user can save the data (Step #6) and exit (Step #7) once they are ready.

*# -- Data -- #  
# declare variables and constants*objFile = **"ToDoList.txt"** *# An object that represents a file*strData = **""** *# A row of text data from the file*dicRow = {} *# A row of data separated into elements of a dictionary {Task,Priority}*lstTable = [] *# A list that acts as a 'table' of rows*strMenu = **""** *# A menu of user options*strChoice = **""** *# A Capture the user option selection  
  
  
# -- Processing -- #  
# Step 1 - When the program starts, load the any data you have  
# in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)*try:  
 objFile=open(**"ToDoList.txt"**, **"r"**)  
 for row in lstTable:  
 strData = row.split(**","**)  
 dicRow = {**"Task"**:strData[0].strip,**"Priority"**:strData[1].strip()}  
 lstTable.append(dicRow)  
  
except:  
 print(**"There is no data in this file yet. Please proceed with the menu to input data."**)  
  
*# -- Input/Output -- #  
# 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 5] - "**))  
 print() *# adding a new line for looks  
 # Step 3 - Show the current items in the table* if (strChoice.strip() == **'1'**):  
 if len(lstTable) == 0:  
 print(**"Currently, there is no Data in the ToDoList.txt File"**)  
 else:  
 for row in lstTable:  
 print (row[**"Task"**] + **'**\t**'** + row[**"Priority"**])  
  
 continue  
 *# Step 4 - Add a new item to the list/Table* elif (strChoice.strip() == **'2'**):  
 strTask = input(**"Type in a Task: "**)  
 strPriority = input(**"Indicate its Priority (High/Medium/Low): "**)  
 print(**"Adding"**, strTask, strPriority, **"to Table"**, sep=**" "**)  
 dicRow = {**"Task"**: strTask.strip(), **"Priority"** : strPriority.strip()}  
 lstTable.append(dicRow)  
  
 continue  
 *# Step 5 - Remove a new item from the list/Table* elif (strChoice.strip() == **'3'**):  
 strDelete = input(**"Which task would you like to delete from the To-Do list? "**)  
 for row in lstTable:  
 if strDelete.strip().lower() in row[**"Task"**].strip().lower():  
 strConfirmDelete = input(**"Are you sure you would like to delete the task "** + strDelete + **" from the file? [Y/N] "**)  
 if strConfirmDelete.lower() == **'y'**:  
 lstTable.remove(row)  
 print(**"The task"** + strDelete + **"has been removed from the file."**)  
 else:  
 print(**"The task"** + strDelete + **"was not removed from the file."**)  
 else:  
 print(**"The task entered does not exist in this row."**)  
 continue  
 *# Step 6 - Save tasks to the ToDoToDoList.txt file* elif (strChoice.strip() == **'4'**):  
 strResponse = input(**"Would you like to save your Data? [Y/N] "**)  
 if strResponse.lower().strip() == **'y'**:  
 objFile = open(**"ToDoList.txt"**, **"w"**)  
 for row in lstTable:  
 objFile.write(row[**"Task"**] + **","** + row[**"Priority"**] + **"**\n**"**)  
 objFile.close()  
 else:  
 print(**"The data entered was not saved."**)  
 *# Step 7 - Exit program* elif (strChoice.strip() == **'5'**):  
 strConfirmExit = input(**"Are you sure you want to exit the program? [Y/N] "**)  
 if strConfirmExit.lower().strip() == **'y'**:  
 break  
 else:  
 continue  
 *# and Exit the program* else:  
 print(**"Invalid entry, please enter a number between 1-5."**)

### Separation of Concerns

The script is organized and noted by the different comments. The data is first declared, and the beginning and the processing and presentation section is blended together. The two sections can later be separated with the functions.

### Main Body of the Script

#### Declaring Data and the Try-Except (Step #1)

At first, the different data and values are declared as string (“”), dictionaries ({}), and lists ([]). The script will first run a try-except looking for text files that already exists. If a text file was found, the rows in the text file will be split into values and re-declared as dictionaries based on its index subscript. If no file was found, it will print that there isn’t a file and asking the user to create on by choosing the menu.

#### Adding values and displaying the data (Step #2-4)

Proceeding with the menu of options, the while(true) loop allows the action to be carried out until the user choose to exit the program. The user can choose to display the data, and if no data is in the file, a message will be printed out indicated that. The user can also choose to add values to the file by inputting the Task and priorities separately, which will be stored as dictionaries into “Task” and “Priorities” after their formatting is stripped.

#### Removing values (Step #5)

With the data that is input by the user, the user can then decide if they would like to remove a row of data. They can do so by typing in a task that they would like to remove (for instance, “homework” removes all rows containing the task homework). The program will evaluate the list (the “table”) row by row and if it found a matching value, it will ask for confirmation of removal and remove it once confirmed by the user. The search is done by looking for the user input (format stripped and lowercased) that matches the task key in the dictionary (also stripped and lowercased). If the row does contain the value, it will print and indicate that it cannot be found.

#### Saving file and exiting the program (Step #6)

When the user is ready, it can choose to save the file as a .txt that is going to be next to the .py file in the local folder. The file will be storying the dictionaries as rows into a table (a list). The user can then exit the data to break the while loop.

Graphical user interface, text, application

Description automatically generated

**Figure 1.** A screenshot of running ToDoList.py in PyCharm with user input

A picture containing text

Description automatically generated

**Figure 2.** A screenshot of running ToDoLis.py in Terminal with user input

Graphical user interface, application, Word

Description automatically generated

(from PyCharm)

Graphical user interface, application, Word

Description automatically generated

(from Terminal)

**Figure 3.** A screenshot of the text file created using the ToDoLis.py script, ToDoLis.txt, with data written into the text file via the program (first one is the export file from the Pycharm run, the second is the exported file from the Terminal run).

## Summary

Collections of data can be stored as dictionaries to better organize the data that is input by the user. It can also closely resemble the table that is usually used to present the data. In general, one key will match one value, and the key is immutable. The values that is entered doesn’t have to be unique.

## Citation

The resources used to complete this document includes the IT FDN 110 A Module 5 course material provided by Prof. Randle Root and the textbook: Python Programming for the Absolute Beginner, Third Edition, By Michael Dawson.