Haley Barsa

August 9, 2022

IT FDN 110 B Foundations of Programming: Python

Assignment05

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

Modifying a New Script that Manages a To-Do List

**Introduction**

This document presents the steps taken to modify a script that manages a to-do list. The script allows a user to select from five options: show current data, add a new item, remove an existing item, save data to a file, and exit the program. The data being collected is a task and its priority. If data is inputted and saved, the data is stored in a .txt file. PyCharm is used to edit and test the script. IDLE is also used to test the script. There are seven steps within the script: load data, display menu, show current items, add new item, remove new item, save to file, and exit program. I will explain the steps in order.

**Load Data**

The first step is to load any existing data within the file to memory since the script will use the stored data. The file is opened in “read only”. For every row within the file, the data is split between commas using the split() function. This separates the rows into columns. The list can now be turned into a dictionary. Dictionaries are written with curly brackets, and have keys and values. A key is a subscript, like an index, but uses character data instead of numeric data. An index is used to select the column within the list row and store it as the value within the dictionary. The value can be a character, list, tuple, or any data type. A key is added to identify the value, example: “task” or “priority”. The strip() function removes unwanted spaces. The dictionary rows are then added to a table so that it can be viewed and appended later. Error handling with “try-except” is included to create a new text file if a file does not exist yet. See Figure 1 below for the commands to accomplish loading data from a file.

*# -- 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** objFile:  
 strData = row.split(**","**) *# Returns a list!* dicRow = {**"Task"**: strData[0], **"Priority"**: strData[1].strip()}  
 lstTable += [dicRow] *# Make a table* objFile.close()  
**except**:  
 objFile = open(**"ToDoList.txt"**, **"w"**)  
 objFile.close()

***Figure 1: Load the Data***

**Display Menu**

The second step is to display a menu of choices to the user. A while loop is used to continue the script until the user exits the program. First within the While loop, a menu is displayed to the user using print commands. The input command is used next to solicit a menu option from the user. The script below in Figure #2 can be used to display the menu options and gather input from the user.

*# -- 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*

***Figure 2: Display the Menu***

**Show Current Items**

The third step is to display the current items in the table. The data was stored in a table in step 1. A for loop is used to show the table with a single row on each line and a comma between columns. The keys are used to pull data from the columns. Error handling is included with “try-except” because data needs to exist in the file before it can be displayed. The script below in Figure #3 can be used to display current items stored as a table in memory.

*# Step 3 - Show the current items in the table***if** (strChoice.strip() == **'1'**):  
 **try**:  
 **for** row **in** lstTable:  
 print(row[**"Task"**] + **','** + row[**"Priority"**])  
 **except**:  
 print(**"File is empty. Please add a new item (Option 2) before selecting this option."**)  
 **continue**

***Figure 3: Show Current Items***

**Add New Item**

The fourth step is to add a new item to the table. Data is gathered from the user using an input command. The new string values are turned into a dictionary row, described in step 1. The new row is appended to the existing table. It is important to note that the change is made only to the memory and not to the text file. The data must be saved for the change to appear in the file. See Figure 4 below to add new items to the existing table.

*# Step 4 - Add a new item to the list/Table***elif** (strChoice.strip() == **'2'**):  
 strTask = str(input(**"Insert Task:"**))  
 strPriority = str(input(**"Insert Priority:"**))  
 dicRow = {**"Task"**: strTask, **"Priority"**: strPriority}  
 lstTable += [dicRow] *# Add new entry to table*

**continue**

***Figure 4: Add New Item***

**Remove New Item**

The fifth step is to remove a new item from the table. The .remove() function is used to delete the latest entry from the table. A confirmation is sent to the user with the print command. Same as step 4, the change only appears in the text file if the data is saved. Figure 5 below provides the script to delete a row from the table.

*# Step 5 - Remove a new item from the list/Table***elif** (strChoice.strip() == **'3'**):  
 **try**:  
 lstTable.remove(dicRow) *# Delete latest entry from table* print(**"You deleted the latest entry."**)  
 **except**:  
 print(**"Unable to delete. Please Save, Exit, and restart program to delete more entries."**)  
 **continue**

***Figure 5: Add New Item***

**Save to File**

The sixth step is to save tasks to the text file. If additions or deletions were made, this step will save the changes to the text file. The file is opened in “write” mode. Each row of the table is written to the file on an individual line. The tasks and priorities are separated by a comma. See Figure #6 below for how to save data to the text file.

*# Step 6 - Save tasks to the ToDoToDoList.txt file***elif** (strChoice.strip() == **'4'**):  
 objFile = open(**"ToDoList.txt"**, **"w"**)  
 **for** row **in** lstTable:  
 objFile.write(row[**"Task"**] + **','** + row[**"Priority"**] + **'\n'**) *# Save table to text file* objFile.close()  
 **continue**

***Figure 6: Save to File***

**Exit Program**

The last step is exiting the program. A break is initiated if this option is selected. Figure 7 below shows the break command.

*# Step 7 - Exit program***elif** (strChoice.strip() == **'5'**):  
 print(**"Program was exited!"**)  
 **break** *# and Exit the program*

***Figure 7: Exit Program***

**Testing the Script**

Figure 8 below shows options 1 and 2 working in Pycharm.

Graphical user interface, text, application

Description automatically generated

***Figure 8: Options 1 and 2 Running From PyCharm***

Figure 9 below shows options 2 through 5 working in Pycharm.

***Graphical user interface, text, application

Description automatically generated***

***Figure 9: Options 3-5 Running From PyCharm***

Figure 10 below shows the script working in a shell window.

***Graphical user interface, text

Description automatically generated with medium confidence***

***Figure 10: Script Running From a Shell Window***

Figure 11 below shows the data saved to a text file.

***Graphical user interface, text, application

Description automatically generated***

***Figure 11: Verifying that the File has Data***

**Summary**

It is a unique challenge to modify someone else’s script. It requires interpretation of the previous writer’s intent. It is helpful to modify small sections of the code at a time. That way errors can be easily located and corrected. This assignment offered practice making dictionaries, transforming between lists and dictionaries, and appending to and removing from a table.