Name: Kimberly Harms
Date: 8/2/2023
Course: IT FDN 110 A Su 23: Foundations of Programming: Python
Assignment 05
https://github.com/kimtara/IntroToProg-Python

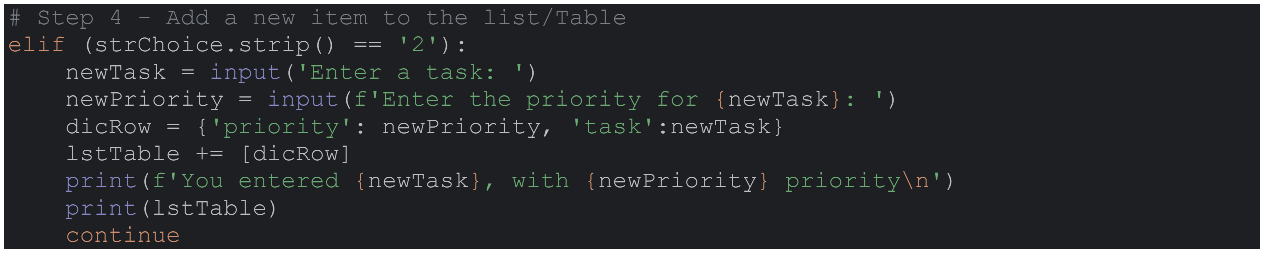
Lists and Dictionaries

Introduction

This week I worked with another type of collection, a dictionary, to organize information for the To Do List program. In this assignment, I was given starter code and a basic pseudocode outline of the pieces to tackle to read, add, and delete data, passing it between a two-dimensional list and a text file. The user utilizes a menu of options to choose any of these functions any number of times and in any order as they accomplish their goals.

Setting up a Dictionary and Table

My first step was to set up data entry into “rows” of a dictionary and add them to a list table. The table contains two “columns” – the task and the priority of the task. I set up my dictionary rows with keys “task” and “priority”. Then I ask for user input for the values newTask and newPriority and pair each key and value in a dictionary row ‘dicRow’ (figure 1). Each dicRow is added to the table lstTable. I included an extra statement print(lstTable) so I could monitor the successful addition of each row. This is commented out for the final code.

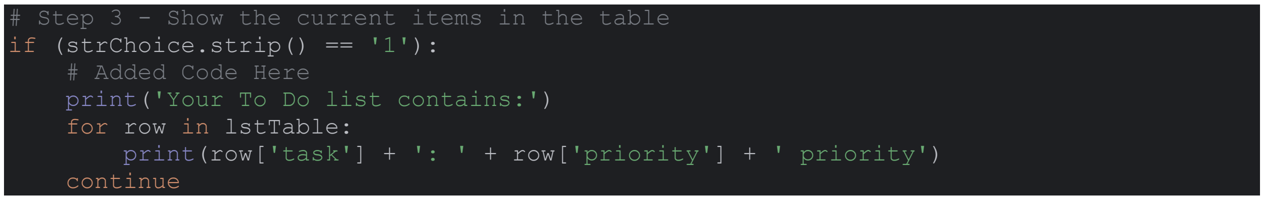


```
# Step 4 - Add a new item to the list/Table
elif (strChoice.strip() == '2'):
    newTask = input('Enter a task: ')
    newPriority = input(f'Enter the priority for {newTask}: ')
    dicRow = {'priority': newPriority, 'task':newTask}
    lstTable += [dicRow]
    print(f'You entered {newTask}, with {newPriority} priority\n')
    print(lstTable)
    continue
```

Figure 1. Pairing values to task and priority keys in a dictionary and adding each dictionary row to a two-dimensional table.

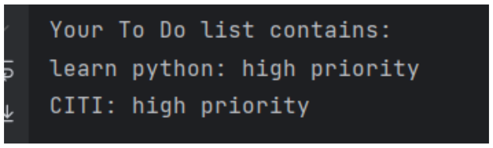
Printing the Table

Next, I wanted to display the entered data to the user with the print function. To show each task and priority pair on a separate line, I used a for loop to cycle through each row of the table (figure 2). I use the keys to index the values and format the output in a user-friendly manner. The resulting output is clear and easy to read, without any brackets (figure 3).



```
# Step 3 - Show the current items in the table
if (strChoice.strip() == '1'):
    # Added Code Here
    print('Your To Do list contains:')
    for row in lstTable:
        print(row['task'] + ': ' + row['priority'] + ' priority')
    continue
```

Figure 2. Printing the To Do items in the table with a for loop and indexing by key.

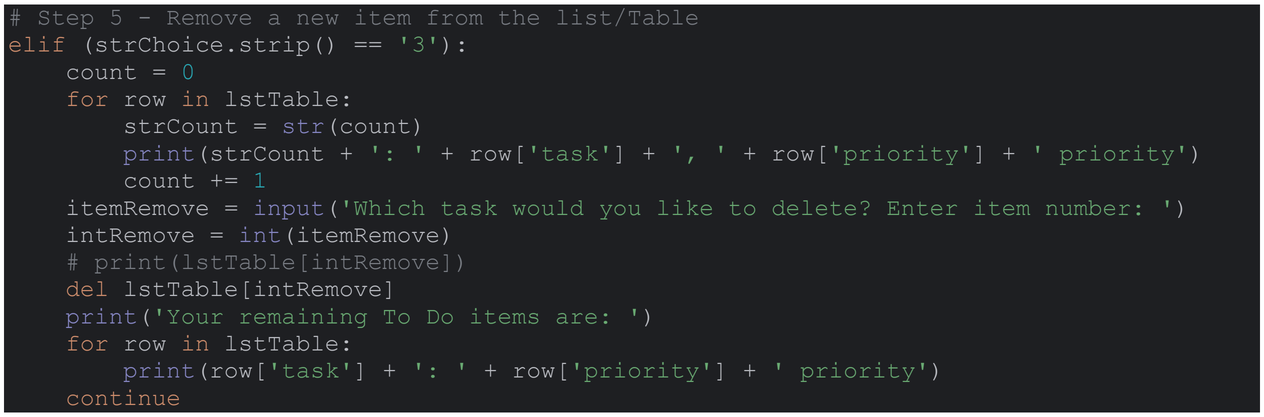


```
Your To Do list contains:
learn python: high priority
CITI: high priority
```

Figure 3. Display showing items entered in the To Do list.

Deleting Items from the To Do List

After setting up my table and confirming that I could add items and display them back to the user, I wanted to allow the user to delete the tasks that have been completed. In order to let the user delete an item I need the user to identify which item to delete and then eliminate the whole dictionary object containing both the task and priority data associated with that item. First, to have the user identify the list item to be deleted I display the index of each table item by setting up a counter that increases by a value of 1 with each row. I can then use a for loop to print the rows and associated index and have user choose item they wish to delete. I ask the user to enter the item number, which is equal to the list index, and use the delete method to remove the item at that position (figure 4). Finally, I display the remaining items back to the user and return to the program menu.

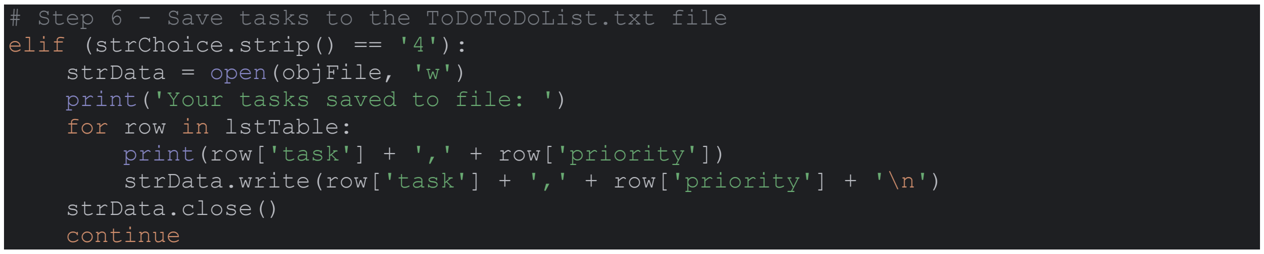


```
# Step 5 - Remove a new item from the list/Table
elif (strChoice.strip() == '3'):
    count = 0
    for row in lstTable:
        strCount = str(count)
        print(strCount + ': ' + row['task'] + ', ' + row['priority'] + ' priority')
        count += 1
    itemRemove = input('Which task would you like to delete? Enter item number: ')
    intRemove = int(itemRemove)
    # print(lstTable[intRemove])
    del lstTable[intRemove]
    print('Your remaining To Do items are: ')
    for row in lstTable:
        print(row['task'] + ': ' + row['priority'] + ' priority')
    continue
```

Figure 4. Deleting completed tasks from the To Do list with a counter and positional index in list.

Saving the To Do List to a Text File

At this point the user can view, add, and delete items for the To Do list, but all the data would be lost when the program ends. I set strData to open the object file designated by objFile. A for loop cycles through the table by dictionary row and key indexes separate out the values needed to be saved in the text file. Each pair is separated by a newline escape character, with a comma between the values (figure 5). Simultaneously, the same for loop displays the saved data to the user.



```
# Step 6 - Save tasks to the ToDoToDoList.txt file
elif (strChoice.strip() == '4'):
    strData = open(objFile, 'w')
    print('Your tasks saved to file: ')
    for row in lstTable:
        print(row['task'] + ',' + row['priority'])
        strData.write(row['task'] + ',' + row['priority'] + '\n')
    strData.close()
    continue
```

Figure 5. Saving table data to a text file with a for loop and indexing by key.

Loading Data from the To Do List Text File

Once the user has entered To Do List data in a file, they need to be able to pass it into the table to modify whenever they return to the program. When the program starts the object file is opened and a for loop reads each line in the file. The rows are split by the comma used to separate the values when saved to the file, and the objects are assigned to the task and priority key and value pairs (figure 6). The resulting dictionaries are added to the list table, which can then be manipulated as the user selects menu options.

Loading the file in this manner works well if the text file has data, but when the file is empty or does not exist, such as the first time a user would use the program, it returns an error. To prevent this error, I used the try/except method of structured error handling. This allows the program to load existing data from the file but skips this step when no data exist and prints a message to the user instead (figure 6).

```python
# 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:  # use try/except so initially empty file does not produce error
    strData = open(objFile, 'r')
    print('Your tasks saved to file: ')
    for row in strData:
        lstRow = row.split(',')
        print(lstRow[0] + ',' + lstRow[1])
        dicRow={'priority':lstRow[1].strip(), 'task':lstRow[0]}
        lstTable += [dicRow]
    strData.close()
except:
    print('Congratulations, your To Do list is empty. Would you like to add a new
task? ')
```

Figure 6. Loading data from the text file, inserting it into dictionary rows, and adding to table list.

Avoiding Errors from User Input

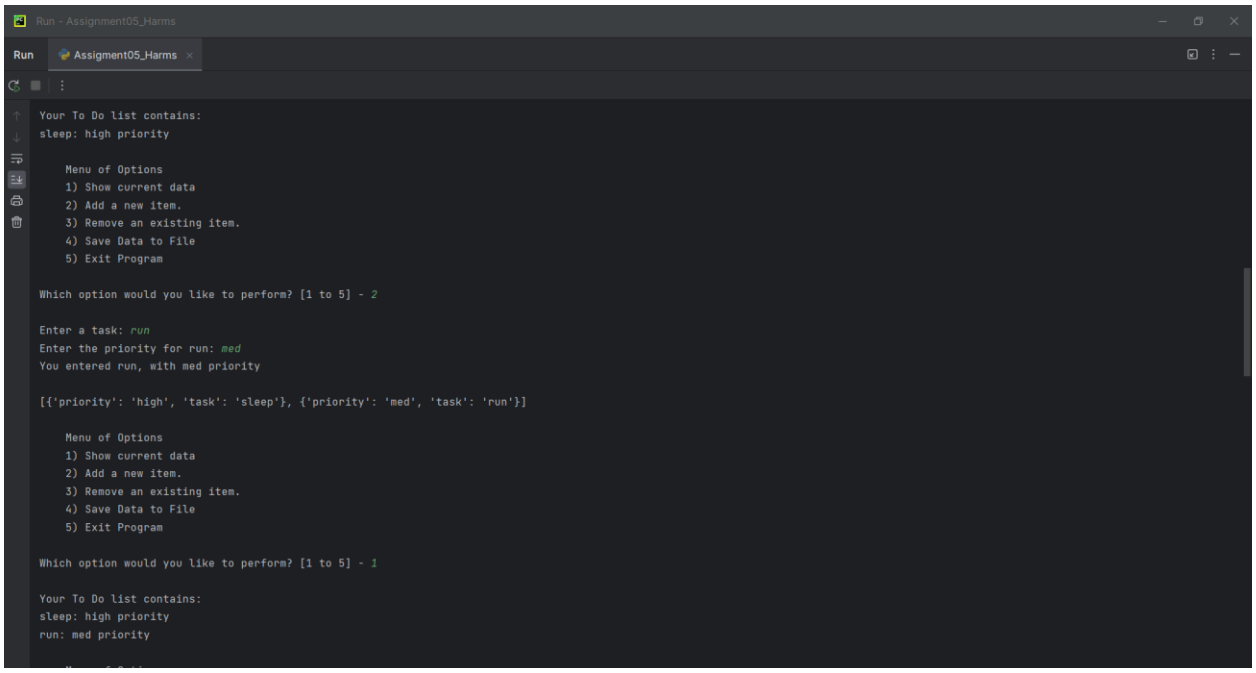
Finally, I wanted to sidestep a couple issues that could arise from invalid user input. I added and else statement that would direct the user to input an integer from 1 to 5 if anything else was entered at the menu. When the user chose to exit the program, I added an extra set of conditional statements with text to remind the user to save their changes before closing the program.

```python
# Step 7 - Exit program
elif (strChoice.strip() == '5'):
    strExit = str(input('Did you save all your changes? If you are ready to exit,
enter y. If you want to continue, enter n: '))
    if strExit.lower() == 'y':
        break  # and Exit the program
    elif strExit.lower() == 'n':
        continue

else:
    # any other input redirect to Menu
    print('Please choose 1 to 5 - ')
    continue
```

Figure 7. Capturing user input errors and confirming data saved before closing the To Do List program.

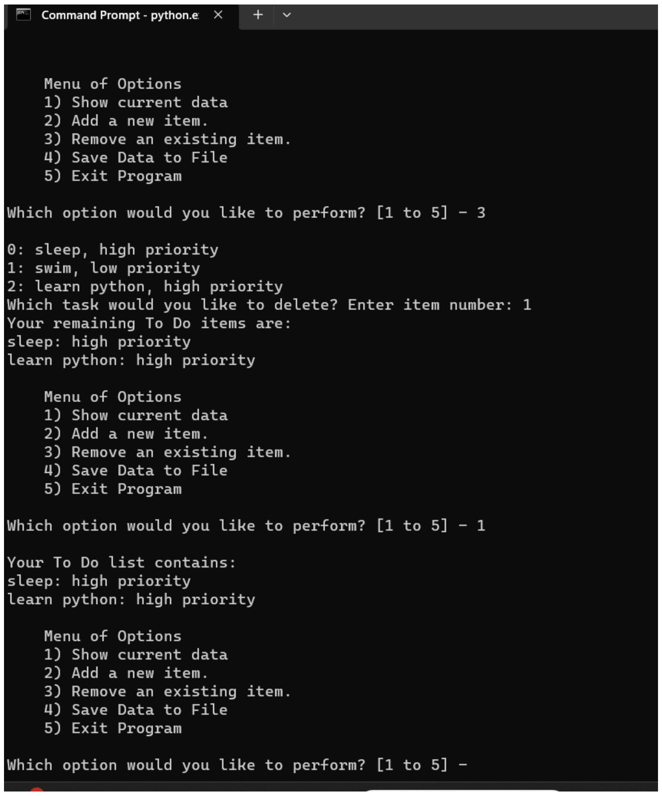
Testing the To Do List Program

After completing my code, I tested it both in the PyCharm IDE (figure 8) and from Command Prompt (figure 9). From either, my input data was successfully stored in the text file (figure 10).



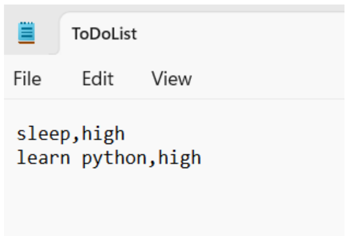
```
Your To Do list contains:
sleep: high priority

    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

Enter a task: run
Enter the priority for run: med
You entered run, with med priority

[{'priority': 'high', 'task': 'sleep'}, {'priority': 'med', 'task': 'run'}]

    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] - 1

Your To Do list contains:
sleep: high priority
run: med priority
```

Figure 8. Running the To Do List program in the PyCharm IDE.



```
    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] - 3

0: sleep, high priority
1: swim, low priority
2: learn python, high priority
Which task would you like to delete? Enter item number: 1
Your remaining To Do items are:
sleep: high priority
learn python: high priority

    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] - 1

Your To Do list contains:
sleep: high priority
learn python: high priority

    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] -
```

Figure 9. Running the To Do List program from the Command Prompt.



```
sleep,high
learn python,high
```

Figure 10. Data from To Do List program saved to the text file.

Summary

Using the provided starter code, I produced a program that could read, add, and delete data. These data could be passed into and back out of a text file, allowing it to be saved once the program closed as well as edited as needed. A two-dimensional list comprised of rows of dictionaries organized the data into tasks and priorities. For loops and indexing let the program step through each set of data, or row, to print, save, load, or format as needed. Structured error handling strategies kept the program running as the user navigated the different menu options to create their To Do list.

References

Randall Root, https://youtube.com/playlist?list=PLfycUyp06LG9fZllIqBrxLcNV4CR50HEX, module 5.

Michael Dawson, Python Programming for the Absolute Beginner, 3rd Edition (2010).

Suzy Stewart, *After Hours Programming*, Accessed on August 5, 2023. https://www.afterhoursprogramming.com/tutorial/python/python-overview/.

ComputerScienceUK, Python Coding -- Reading and Writing to Text Files, https://www.youtube.com/watch?v=m0o0CkYsDzI.

TutorialsPoint, https://www.tutorialspoint.com/How-to-remove-an-object-from-a-list-in-Python.