Matthew Kojetin
05/18/20
IT FDN 100 A
Assignment 05
https://github.com/matthewkojetin/IntroToProg-Python

Module 05: Lists, dictionaries, error handling, and functions

Introduction

In this assignment I'll describe how to use a combination of lists and dictionaries to store data, how to retrieve data from a file and store it into a list or dictionary, and print custom error messages. I'll use this information to create a To do list script that allows you to manipulate the data and save it to file. I'll also describe the basics of functions.

Loading list data from a file

Last assignment we looked at list syntax (`listEx = [1,2,3]`) and how to write lists to a file. Building on that, you can also read data from an existing file when you open a text file in read mode. It's best to load the data into memory in order to work with it. To do this, open the file in read mode, then loop through the contents to place it in a list object, split the data based on the commas, and append a table (another list) to include that row. For easy processing don't include spaces between data elements (Figure 1).

```
lstTable = []

objFile = open("textfile.txt", "r")
for row in objFile:
    lstRow = row.split(",")
    lstTable.append(lstRow)
objFile.close()
print(lstTable)
```

Figure 1. Loading data to a table. Separating at commas.

Dictionaries

Dictionaries are similar to list, tuple, and string sequences except that the subscript doesn't use an index, it uses a key to label the values (Figure 2).

```
dicExample = {"keyname": "key value", "keyname2:" "key 2 value"}
print("keyname" + "," + "keyname2")
```

Figure 2. Saving keys and values to a dictionary.

Loading dictionary data from a file

It is helpful to use dictionary as a row of data, and store those rows in a list. To seperate the values before assigning them to a key, we use a temporary list object (Figure 3).

```
objFile = open("textfile.txt", "r")
  for strData in objFile:
      lstRow = strData.split(",")
      dicRow = {"key1": lstRow[0], "key2": lstRow[1].strip()} # The .strip() funct
      lstTable.append(dicRow)
  objFile.close()
```

Figure 3. Loading data to a table. Separating at commas.

Separation of concerns

Separation of concerns is a design principle for organizing your code in a way that can be repurposed easily. The idea is to break your content up into the following sections.

- Layer of data: Declare variables
- Layer of processing: Manipulate that data as needed (functions help accomplish this cleanly)
- Layer of presentation: User interaction flow

Functions

Named set of one or more statements that you can call later in a script (Figure 4).

```
def DoThisAction(): # this block of code loads but does not run yet
  return (int1 / int2)
...
DoThisAction() # Calling the function at the appropriate time, later in the script
```

Figure 4. Basic function syntax.

Error handling (try-except)

This is a way to present the user of your script with a human-readable error code if you anticipate a point where the script could fail (Figure 5).

```
try:
    x / y
except:
    print("Could not divide " + str(x) + " by " + str(y))
```

Figure 5. Try/except error handling.

To-do script

To complete the assignment I started by loading data from a text file as dictionaries (rows) stored in a list. I kept encountering errors that would prevent my script from running if there was no data in the text file (or if it didn't exist), so I used try/except for error handling. For each row in the file, my script splits the data at the commas, strips any extra characters at the end of the line (the `\n\`) then assigns the values to the keys `task` and `priority`, using a temporary list variable. I then append this row to the `lstTable`. (Figure 6).

```
try:
    objFile = open(strFile, "r")
    for strData in objFile:
        lstRow = strData.split(",")
        dicRow = {"task": lstRow[0], "priority": lstRow[1].strip()}
        lstTable.append(dicRow)
    objFile.close()
    print("Data loaded from file \"ToDoList.txt\"")
except:
    print("No existing data in file \"ToDoList.txt\"")
```

Figure 6. Importing data from a file, storing it in dictionary row, then adding each row to a list.

Then the menu is presented. When the user makes a selection of 1, I cycle through each row of the table using a for loop and print the data. When the user selects 2, the user is prompted to enter a task and priority, which are saved to a row that is appended to the list table (Figure 7).

```
print("Add task to list:\n")
    dicRow = {"task":str(input("Task: ")).strip(), "priority":str(input("Priority:
    lstTable.append(dicRow)  # adds row to table
```

Figure 7. Collecting user data and saving it to a row in the data table.

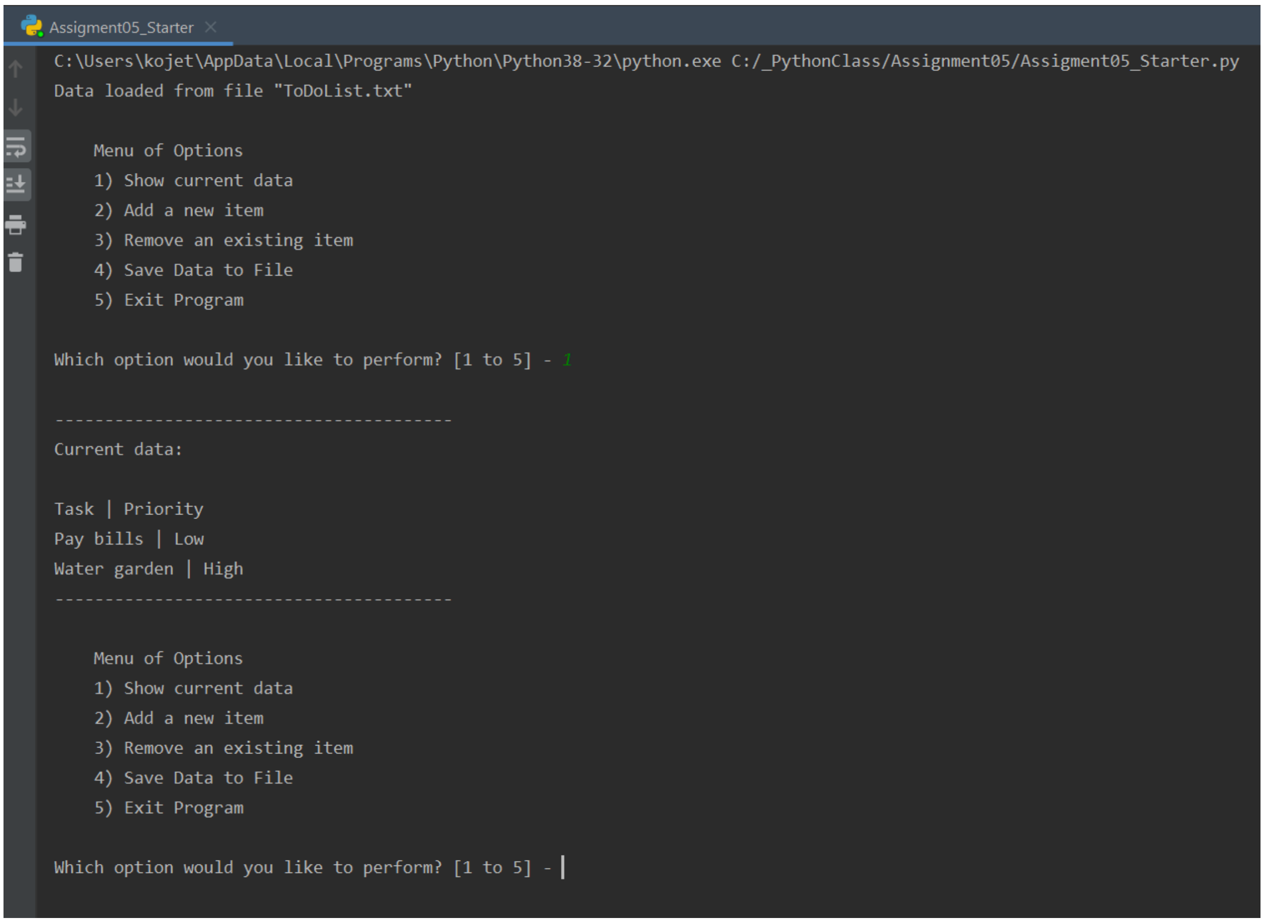
When the user selects 3, I list the `task` key values and ask them to enter which item they'd like to delete. I used a combination of `strip()` and `lower()` functions to eliminate some potential string errors. I also added a boolean variable `boolRemove` that is assigned a value of True when an item is removed. If `boolRemove` is false after looping through all rows, I print a message that there is no matching item. I use the `.remove()` function to remove a dictionary from the list if there is a match (Figure 8).

```
print("Remove an item from the list:\n")
for dicRow in lstTable:
    print(dicRow["task"])
print("\n")
boolRemove = False
strRemoveChoice = input(str("Enter task would you like to remove: ")).strip()
for dicRow in lstTable:
    if dicRow["task"].lower() == strRemoveChoice.lower():
        lstTable.remove(dicRow)
        print(strRemoveChoice + " removed.")
        boolRemove = True
if boolRemove == False:
    print("No matching item.")
```

Figure 8. Deleting an row from the data list.

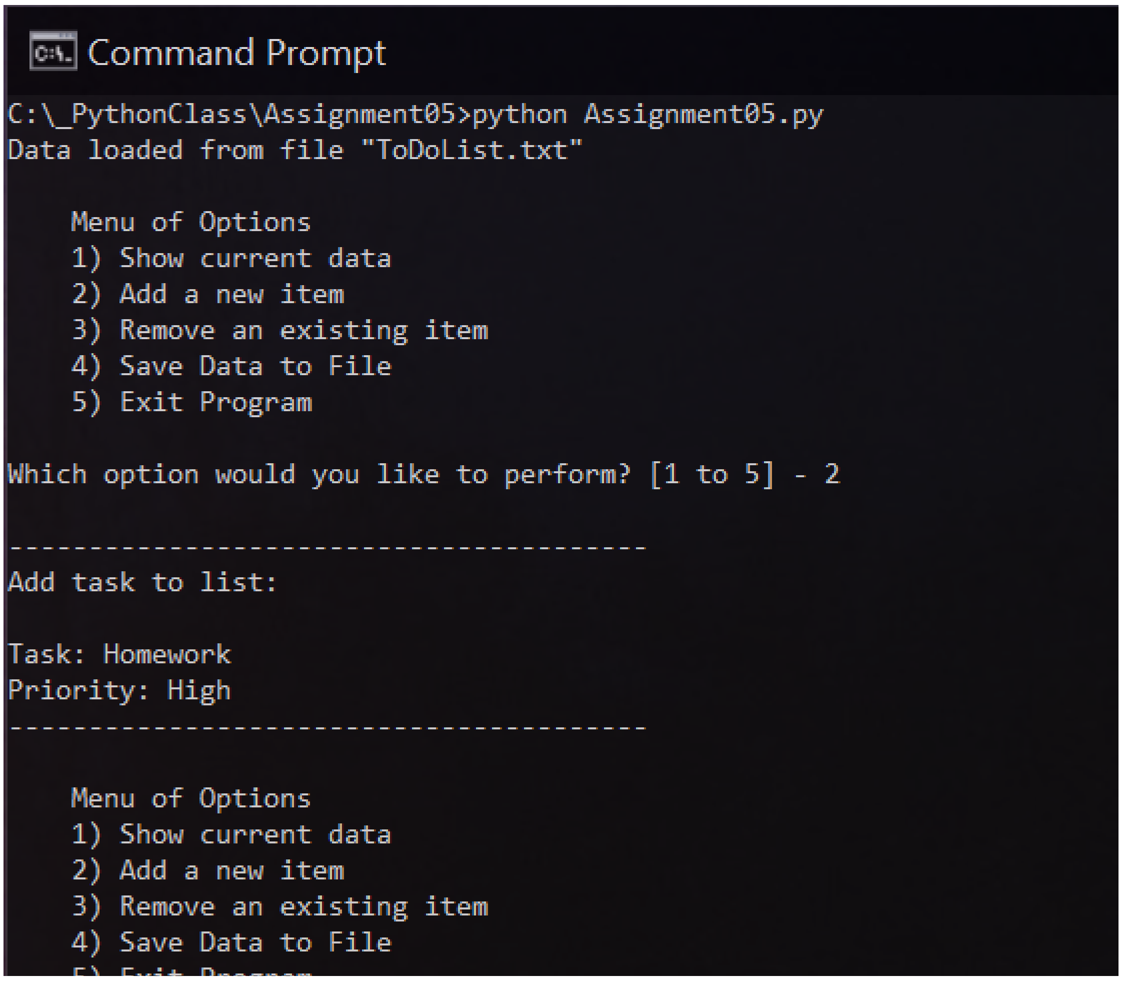
If the user selects 4 from the menu, I save the data to a file by looping through the rows and adding formatting. If the user selects 5 the loop breaks. I also added an `else` statement at the end to indicate if the user makes an invalid selection at the main menu.

The script runs in both PyCharm (Figure 9) and from the Windows command line (Figure 10).

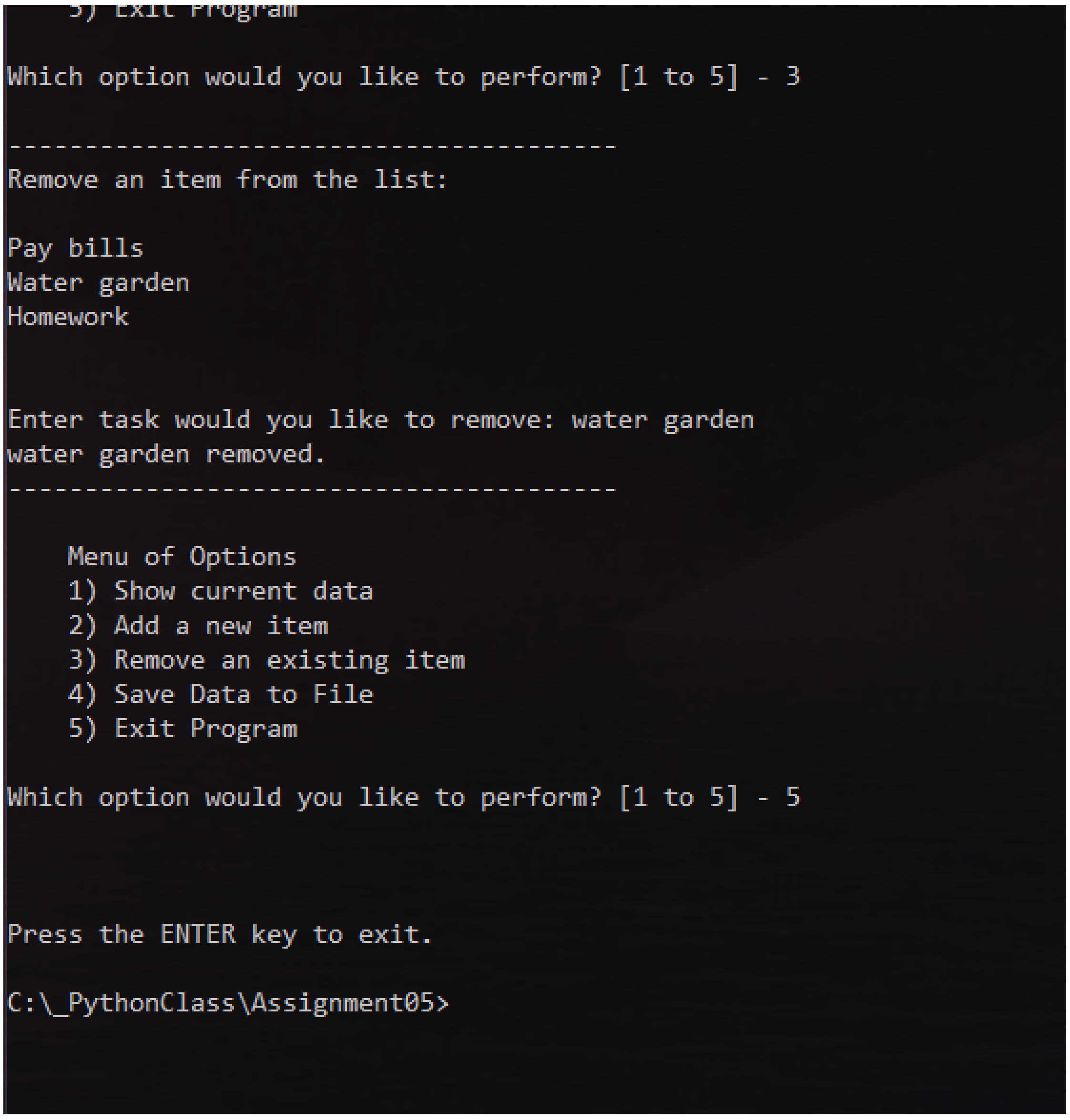


```
C:\Users\kojet\AppData\Local\Programs\Python\Python38-32\python.exe C:/_PythonClass/Assignment05/Assigment05_Starter.py
Data loaded from file "ToDoList.txt"

    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

----------------------------------------
Current data:

Task | Priority
Pay bills | Low
Water garden | High
----------------------------------------

    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. Script running in PyCharm.



```
C:\_PythonClass\Assignment05>python Assignment05.py
Data loaded from file "ToDoList.txt"

    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

----------------------------------------
Add task to list:

Task: Homework
Priority: High
----------------------------------------

    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

----------------------------------------
Remove an item from the list:

Pay bills
Water garden
Homework


Enter task would you like to remove: water garden
water garden removed.
----------------------------------------

    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] - 5



Press the ENTER key to exit.

C:\_PythonClass\Assignment05>
```

Figure 10. Script running in Windows command line.

Summary

You can have persistent data for your scripts if you save them to a text file. You can easily recall that data and store it in a combination of lists and dictionaries to manipulate in memory while your script is running.