

Dictionaries

Introduction

This assignment builds upon the concepts used in previous weeks, introducing dictionaries, script templates, and much more about coding best practices and structure. I also learned about GitHub and began using it for file sharing and review.

Dictionaries and Code Management

Dictionaries are similar to lists and other collections in many ways. The primary difference is that dictionaries use a key subscript (with characters) instead of the numeric index subscripts used by these other types. These keys can be thought of as built-in “column” headers for a row of data values. Data can be read from a file such as a text file into memory as a dictionary, just as it can be into a list. To read to a dictionary object, you first read to a list row, then create the dictionary row by entering the key before each indexed value in the list row (together, the pair is called an item).

There are a number of ways to streamline and organize code to make it easier to share, reuse, and revisit. One concept is “Separations of Concerns,” which means that code is divided into sections with distinct purposes that do not overlap. Programs designed this way are modular and can be more easily broken down and understood or used in other applications. A script template that divides code into sections for Data, Processing, and Presentation is a common way to encourage separation of concerns. Like their name implies, script templates provide a pre-defined, starting structure to a new code project.

A function is a section of statements that have been given a name, which can then be called elsewhere in the script. Because the functions can be defined in the Processing section and then not run until the Presentation section, they can help you keep your code organized.

Error handling can make a script easier for end users to interact with successfully. The Try-Except construct is a common method of error handling that runs a block of code and moves on if successful. If not, it prints out an error message tailored to the situation.

GitHub is a source control software that allows people to store and interact with files on the internet. It allows people to track versions of their own work as well as collaborate with others.

To-Do List Script

The To-Do list script illustrates reading data from a text file as lists and dictionaries, writing to a text file, and interacting with tables containing dictionary objects via user input.

Setup

Using the prepared starter code structure, I first reviewed the existing code and checked which portions needed to be added. Where a section was similar to a lab exercise or past assignment, I copied in code and modified it for this use case.

Tasks

The sections corresponding to each user choice required additional code, in addition to the initial step of reading the existing txt file to memory as rows of dictionary objects. I worked on the code one block at a time, to be sure each was running as expected. Reading to/from the text file and adding a new item were similar to tasks performed previously, but needed to be modified to use dictionaries and this use case. Deleting a row was a new operation. I used the code below, which prints out the existing list (like option 1), prompts the user for a task, and then cycles through the dictionary rows in the table looking for that value with the “task” key. If found, it deletes the entire row from the table.

```
# Step 5 - Remove a new item from the list/Table
elif (strChoice.strip() == "3"):
    print("Your current to-do list (task, priority):")
    for dicRow in lstTable:
        print(dicRow["task"], dicRow["priority"], sep=', ')
    delTask = input("Which task from the list above would you like to delete?: ")
    for dicRow in lstTable: # Search table by row for the task to delete
        if delTask.lower() in dicRow["task"].lower():
            lstTable.remove(dicRow) # Delete the indicated row
    print(delTask.title() + " has been removed from the list.")
    continue
```

Running the Script

After iterating on each portion of the script, running the script together went relatively smoothly. Because I elected not to add checks on user input or error handling, I did have a few runs where I accidentally entered a “unaccepted” value for priority that was written to the txt file. This would be an area to improve in future iterations. Overall, barring user error, the script behaved as expected in PyCharm and Terminal.

Figure 1: Script running on PyCharm, with txt file before and after

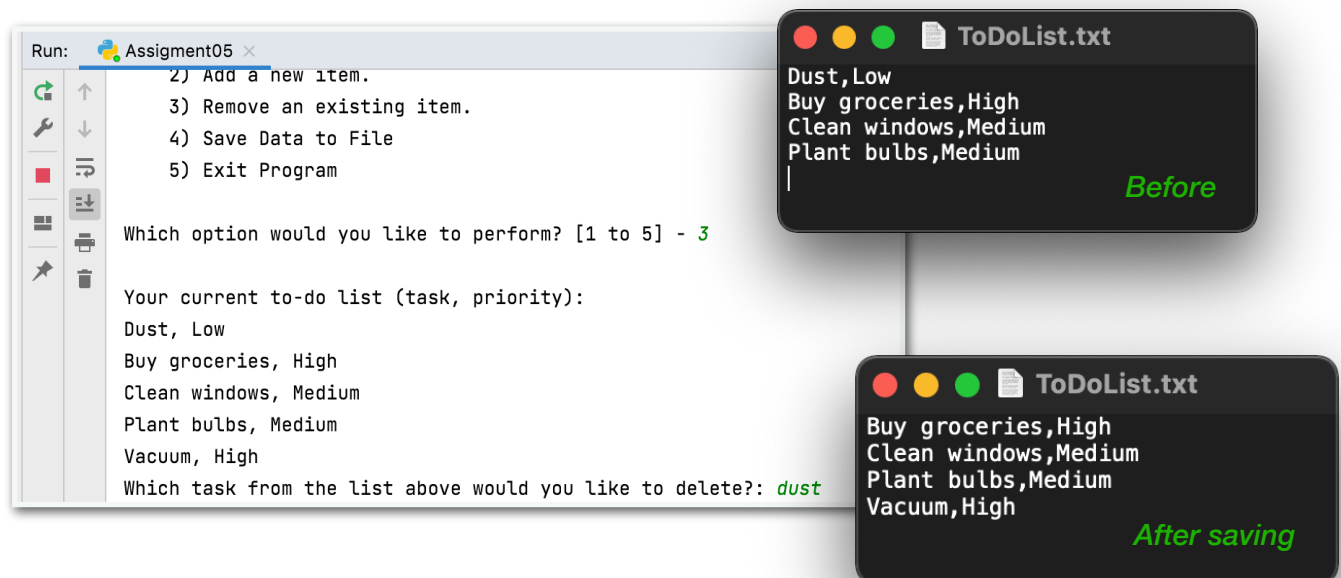
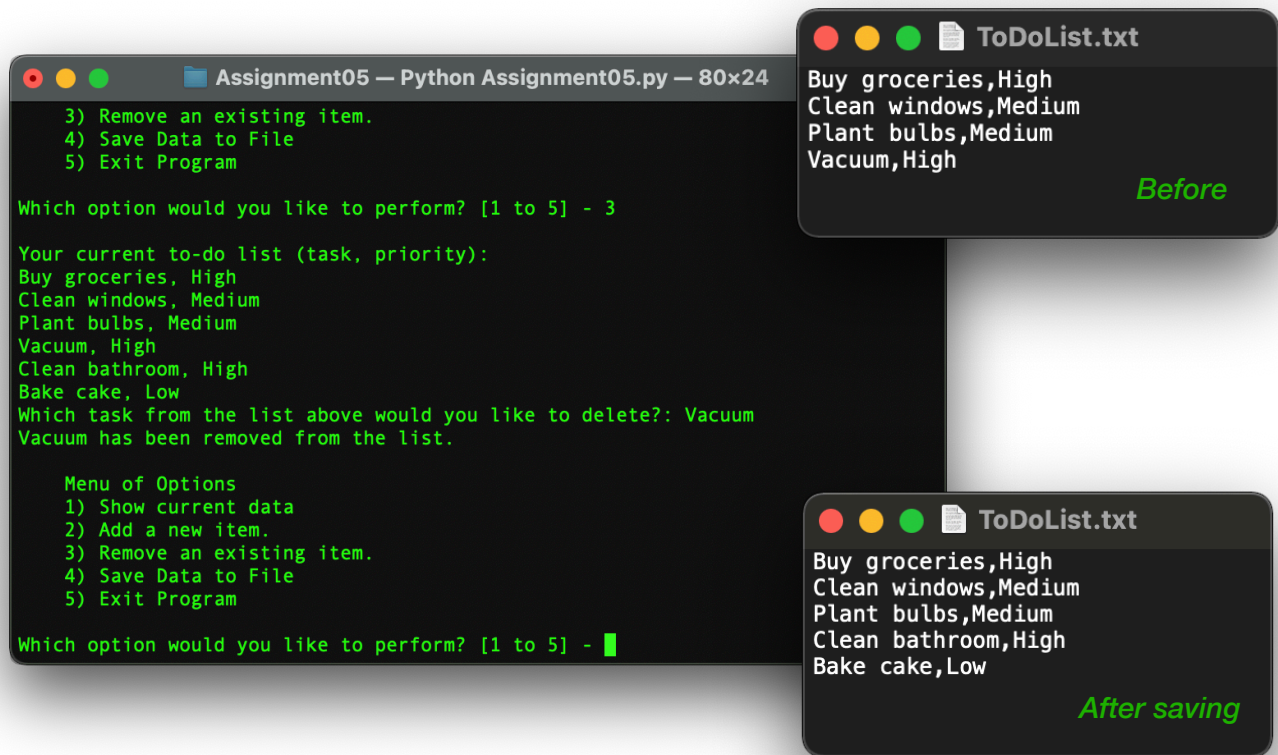


Figure 1: Script running on Terminal, with txt file before and after



```
Assignment05 — Python Assignment05.py — 80x24
3) Remove an existing item.
4) Save Data to File
5) Exit Program

Which option would you like to perform? [1 to 5] - 3

Your current to-do list (task, priority):
Buy groceries, High
Clean windows, Medium
Plant bulbs, Medium
Vacuum, High
Clean bathroom, High
Bake cake, Low
Which task from the list above would you like to delete?: Vacuum
Vacuum has been removed from the list.

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] - █
```

Before

```
ToDoList.txt
Buy groceries,High
Clean windows,Medium
Plant bulbs,Medium
Vacuum,High
```

After saving

```
ToDoList.txt
Buy groceries,High
Clean windows,Medium
Plant bulbs,Medium
Clean bathroom,High
Bake cake,Low
```

Conclusion

This assignment and module reinforced previous concepts related to lists, conditionals, and saving to files. Building on this, I learned about dictionaries and practiced using them to interact with both existing and user-created data. With the introduction of GitHub for sharing and versioning, script templates, and error handling (though not all used within this assignment), it is encouraging to be exposed to industry best practices to build upon going forward.