

Creating and Managing a Task List

Introduction

This week we modified starter code to finish a script that inputs an existing task list (if one already exists), adds and removes tasks, and then can save the list back to an external text file.

Learning concepts this week

The script builds upon lists and menu of choices (using if and elif statements) that we started using last week.

New this week are dictionaries. Initially they seemed very similar to lists, and I questioned what was really different about them and why we needed them. Upon further study, there are two major characteristics that are different about dictionaries:

- While lists have numbered indices (0, 1, 2, etc.), a dictionary uses “keys” that are associated with a value. While a list might store someone’s name in the 2 index, in a dictionary you could tag that name with a key such as “Full Name,” which is more intuitive and easier for the programmer to work with than remembering which index of a list stores names.
- The indices of a list create an order. The 1 index occurs before the 3 index. In contrast, keys and their associated values are not in any order.

It is also my understanding that if you were working with a large amount of data, a dictionary can be searched by Python more quickly than a list.

File Reading/Writing has gotten more sophisticated. Previously we would read the text file data into a list. Now we have evolved to importing line of the text file into its own dictionary which is then appended into a table of dictionaries.

Since Python runtime errors are not very helpful to the average user, we started working with the concept of a try-except conditional statement. The idea is you nest a section of code inside

a try statement. If something causes one of the lines of codes to fail (e.g., trying to read a text file that doesn't exist or the user inputs something that was not expected), rather than going straight to a runtime error, Python moves down to the except section. The script execution is like when an if statement is not satisfied and the script moves down to the else section.

Finally, a couple of things were introduced that will not see immediate use because we worked with starter code:

- Separation of Concerns
 - The idea of separating each part of a script in its own section (for example: Data, Processing, and Presentation/Input-Output) to make it easier to trace through a script, modify part of a script without potentially breaking other parts
- Functions
 - It is hard to do all the processing up in the processing section because some of it cannot be done until after input has been received by the user. A function allows you to define a sub-routine up in the processing section and then down in the Presentation section, you can simply call the function. This makes for an easier to read script.

Process to make program

This week we began with some starter code for the first time. This was good and bad. It was great to have a head start, and the script layout already determined. I went to start creating variables and noticed that there were already variables initialized for somethings. I use the nomenclature of type_name (e.g., str_user) whereas the original author uses a typeName (e.g. strUser) so I had to adjust what I was doing to stay consistent. In a similar vein, I use single (') parentheses rather than double ("). I switched to that nomenclature and then later noticed both types were being used. I resisted the urge to start changing what was there and focused on just working on the sections of the code I was tasked with.

There was already a header (see Figure 1), so the only change needed was to add a line to the change log to explain what I was doing.

```
# ----- #
# Title: Assignment 05
# Description: Working with Dictionaries and Files
#
#         When the program starts, load each "row" of data
#         in "ToDoToDoList.txt" into a python Dictionary.
#         Add the each dictionary "row" to a python list "table"
# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# Mark Brugger,11.13.22,Added code to complete assignment 5
# ----- #
```

Figure 1 - Header of script

Most of the variables I would need were already initialized in the Data section, as shown in Figure 2, but I added a few for the sections of the script where the user can add a task or remove a task. Strictly speaking, Python does not require that variables be declared before using them, however it makes it easier to follow what is going on in a script. Additionally, since our programming languages do require variable initialization, it is good to get used to adding it to my code.

```
# -- 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
strTask = "" # Capture the name of the user's added task
strPriority = "" # Capture the priority of the user's added task
strRemoval = "" # Capture the task that the user wants to remove
```

Figure 2 - Initialization of variables for the script

The first step of the processing section (see Figure 3) is to read data from an external text file if any such data exists. If it is the first time a user is running this script or they chose not to save their data, there will not be a text file. This will create a run-time error in Python. You could make sure there is a blank text file with the name `ToDoList.txt` present in the folder the script is run from, but a more elegant choice is use of the try-except statements. If a file is not found, the script will move on to the except section where the user is told no file exists, but one will be created later when the save command is used.

```
# -- 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:
    dicRow = {}
    objFile = open("ToDoList.txt", "r")
    for row in objFile:
        strData = row.split(",")
        dicRow = {"Task": strData[0], "Priority": strData[1].strip()}
        lstTable.append(dicRow)
    objFile.close()

except:
    print("\nFile not found. One will be created when you save")
```

Figure 3 - Processing section where the script attempts to read user data from an external text file

Next, the script goes to a menu of choices and uses a while loop as seen in Figure 4. This portion of the script was part of the starter code.

```
# -- 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 4 - User Menu of the script

If the user selects 1, the script will print the list of tasks that were read in from the text file and/or were added by the user while the script was running. If there is nothing to show, then only the headings (Task Name and Priority of Task) will be printed.

```
# Step 3 - Show the current items in the table
if (strChoice.strip() == '1'):
    print("Task Name, Priority of Task")
    print("=" * 27)
    for row in lstTable:
        print(row["Task"], row["Priority"], sep="_", " ")
    continue
```

Figure 5 - Subsection of the script for showing current tasks in the table

The next step (see Figure 6 below) allows the user to add a new task to the table. I thought about adding a nested while loop so the user could add multiple tasks at once. However, since the main while loop takes you right back to the menu where you can choose to add another task, that too much of an inconvenience for the user. This section of the script makes use of the list append command which we first started using last week. A confirmation message lets the user know their addition was successful.

```
# Step 4 - Add a new item to the list/Table
elif (strChoice.strip() == '2'):
    strTask = input("Enter the name of the task: ")
    strPriority = input("Enter the priority level of the task, (Low/Medium/High): ")
    lstTable.append({"Task": strTask, "Priority": strPriority})
    print("\nYour task has been recorded.")
    continue
```

Figure 6 - Section to add a task to the table

As an advanced feature, the user can remove a task that already exists in the table. The code for this is shown in Figure 7 and uses an if statement inside a for loop to search for the task name. The .lower() tag was added to both the Task name and the user's input to convert both into lower case since Python is case sensitive. If the script was searching for "Wash Clothes" and the task had been recorded as "wash clothes" or any other form that didn't match the exact case that the user input, then Python would not find a match.

```
# Step 5 - Remove a new item from the list/Table
elif (strChoice.strip() == '3'):
    strRemoval = input("Enter the name of the Task to Remove: ")
    for row in lstTable:
        if row["Task"].lower() == strRemoval.lower():
            lstTable.remove(row)
            print("\nYour task has been removed")
        else:
            print("\nTask was not found")
    continue
```

Figure 7 - Searching for and removing a task if a match is found

Next, is the portion of the script that will save tasks to the external text file. Previously I was a fan of using the append ("a") command when opening these files with the rationale that users are often going to want to grow their lists over time. However, with the way this script is written, append would cause problems. At the beginning of the script, the contents of the text file are read into memory (if there is a text file that has contents). For the duration of the script, it is working with the table in memory, not the external text file. As such, if we were to use the append command, it would create duplicate items. Instead, the write ("w") command is used because it will clear the contents of the text file and the entire table from memory will be written. As a confirmation, the user is told that their tasks have been successfully saved.

```
# Step 6 - Save tasks to the ToDoToDoList.txt file
elif (strChoice.strip() == '4'):
    objFile = open("ToDoList.txt", "w")
    for row in lstTable:
        objFile.write(str(row["Task"]) + "," + str(row["Priority"]) + "\n")
    objFile.close()
    print("\nYour tasks have been saved to a file.")
    continue
```

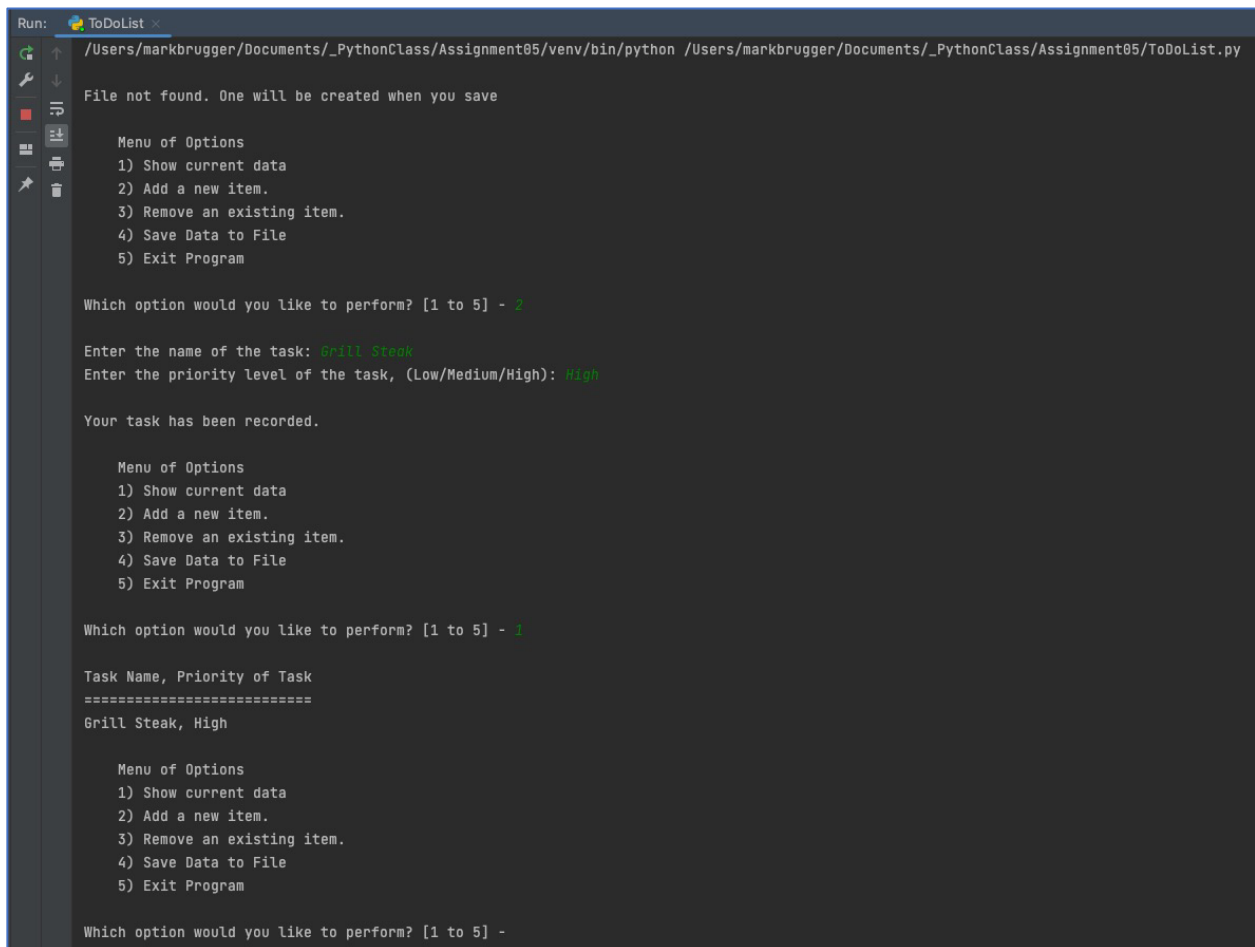
Figure 8 - Subsection to save the table of tasks to an external file

Finally, when the user is finished using the script, they select option 5, and this section of the code (see Figure 9) breaks the loop after telling the user that they are exiting the program.

```
# Step 7 - Exit program
elif (strChoice.strip() == '5'):
    print("Now exiting the program...")
    break # and Exit the program
```

Figure 9 - Exiting the script

The script was then run from within PyCharm IDE (see Figure 10 below) and the Terminal in Mac OS (see Figure 11) to confirm that is running properly.



```
Run: ToDoList x
/Users/markbrugger/Documents/_PythonClass/Assignment05/venv/bin/python /Users/markbrugger/Documents/_PythonClass/Assignment05/ToDoList.py

File not found. One will be created when you save

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 the name of the task: Grill Steak
Enter the priority level of the task, (Low/Medium/High): High

Your task has been recorded.

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

Task Name, Priority of Task
=====
Grill Steak, 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 10 - View of the script running in PyCharm


```
Assignment05 — python3 ToDoList.py — 140x56
Last login: Tue Nov 15 22:00:43 on console
(base) markbrugger@Marks-MacBook-Pro ~ % cd /Users/markbrugger/Documents/_PythonClass/Assignment05
(base) markbrugger@Marks-MacBook-Pro Assignment05 % python3 ToDoList.py

File not found. One will be created when you save

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 the name of the task: Wash Car
Enter the priority level of the task, (Low/Medium/High): Medium

Your task has been recorded.

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

Task Name, Priority of Task
=====
Wash Car, Medium

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] - 4

Your tasks have been saved to a file.

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 11 - Excerpt of the script running in MacOS Terminal

After exiting the program (and after the program was told to save data to file), there is a text file in the script folder and Figure 12 shows that it successfully recorded the task table.

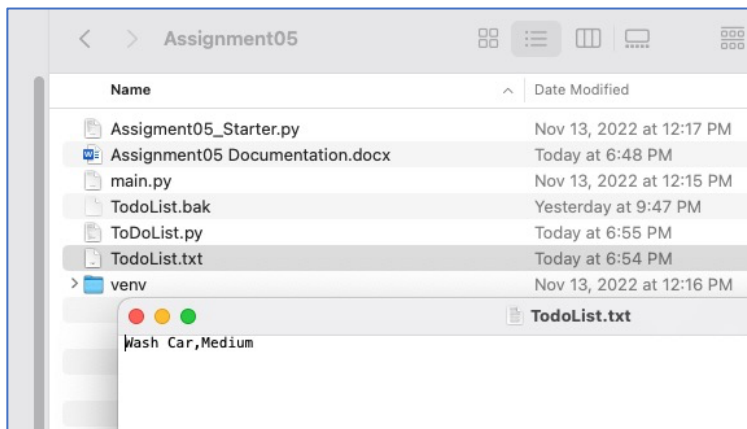


Figure 12 - External text file recorded the task

Had multiple tasks been added to the external file, it would have looked like Figure 13 below.

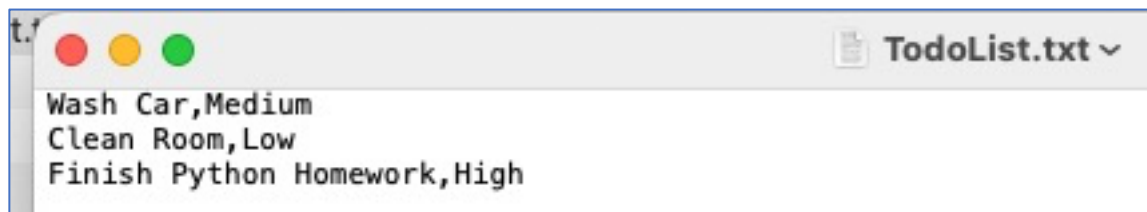


Figure 13 - Text file with more data

Summary

This week we took some starter code which gives the user a menu of choices to work with a task list that will reside in a table of dictionaries while the script is running and in an external text file for future use if the user chooses to save. It was the first time using try-except loops for some simple error handling and dictionaries as a more efficient way to store data in memory.

If the script was more sophisticated and there were options to organize the list based on priority, I would have restricted input to the choices of Low, Medium, and High. However, based on the parameters of the assignment this week, I did not see the need, which simplified the script. This does, however mean that the user could input anything they wanted to and instead of “Medium” above in Figure 13, it could have said “Taylor Swift sold a lot of tickets yesterday” or anything on the user’s mind.