Jeremy Patoc

08/09/2023

IT FDN 110 A

Assignment 05

https://github.com/jpatoc/IntroToProg-Python

To Do List

Intro

Module 5 introduced a Dictionaries, a new type of collection. They are similar to lists, but have benefits of their own like being able to refer to a 'key' instead of an 'index', which makes them more flexible. Many concepts are beginning to layer on top of one another. For example, you can have a list of dictionary rows. This module also explores new ways to read and write data to files as well as manipulate data using FOR LOOPs.

Beyond pure technical concepts, module 5 also covers what I would call 'good programming hygiene' like:

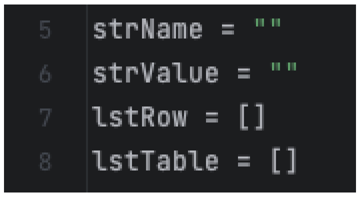
- Proper code organization via "Separation of Concerns"
- Using a script template as the starting point
- Custom error handling to make a program more user-friendly

Creating the Program

Unlike previous programs, that were largely created from scratch, the To Do List program started with a script template. I started with a script header and three code sections - Data, Processing, and Input/Output.

1) **Data** - this is single place where any variables that will be used can be defined. Many of the neccessary variables were already defined, but I added a few of my own based on how I built the program
2) **Processing** - the text file that the program reads/writes to and from is loaded
3) **Input/Output** - the 'heart' of the program where all of the logic is contained

To begin with, and as mentioned above, I added some additional variables beyond what was included in the script template. This was largely due to familiarity and what I've used and/or seen in previous programs and labs. I created ***objFile*** to store the reference to the name of the file, ***lstRow*** to hold the indivdual rows from the table, and ***strTask*** and ***strPriority*** to hold the user inputs to add a new record to the table and file



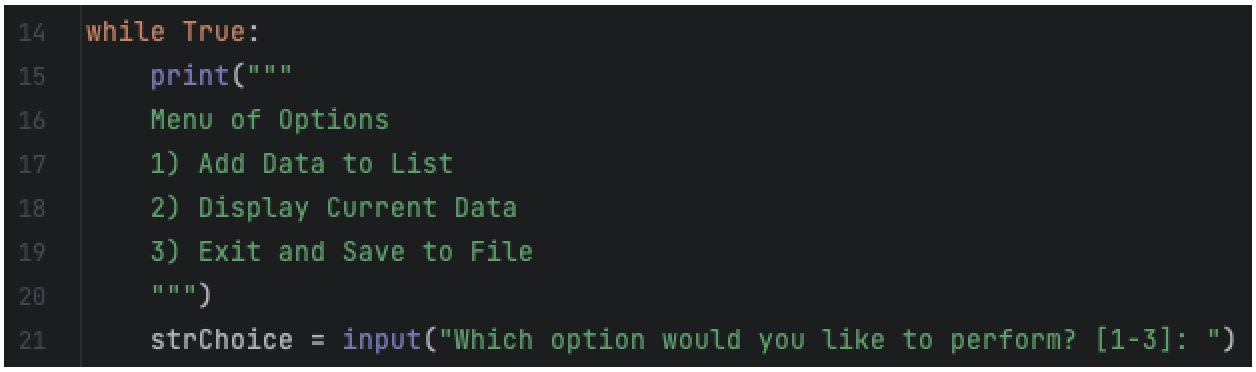
```
strName = ""
strValue = ""
lstRow = []
lstTable = []
```

Figure 1 : Empty Variables to Store User Inputs

Next, I created a While Loop to display a menu of options for the user to choose from. There are 3 options to chose from and the user is prompted to enter a corresponding input. The While loop is nice because the menu will continue to display at the beginning of each new flow (provided the user doesn't exit the application).

<u>Menu Options:</u>

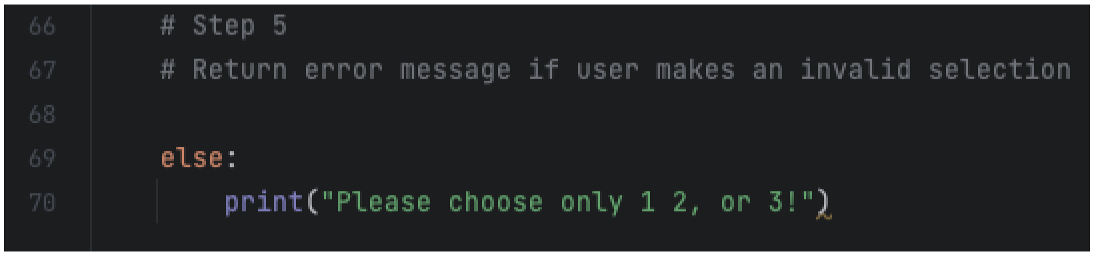
1) Add Data to List
2) Display Current Data
3) Exit and Save to File



```
while True:
    print("""
    Menu of Options
    1) Add Data to List
    2) Display Current Data
    3) Exit and Save to File
    """)
    strChoice = input("Which option would you like to perform? [1-3]: ")
```

Figure 2 : Menu of Options and Input Selection

The user's input will define the behavior of the program using a series of IF statements. As an error handling measure there is also logic to account for if a user makes a selection that doesn't correspond with one of the menu options (1, 2, or 3).

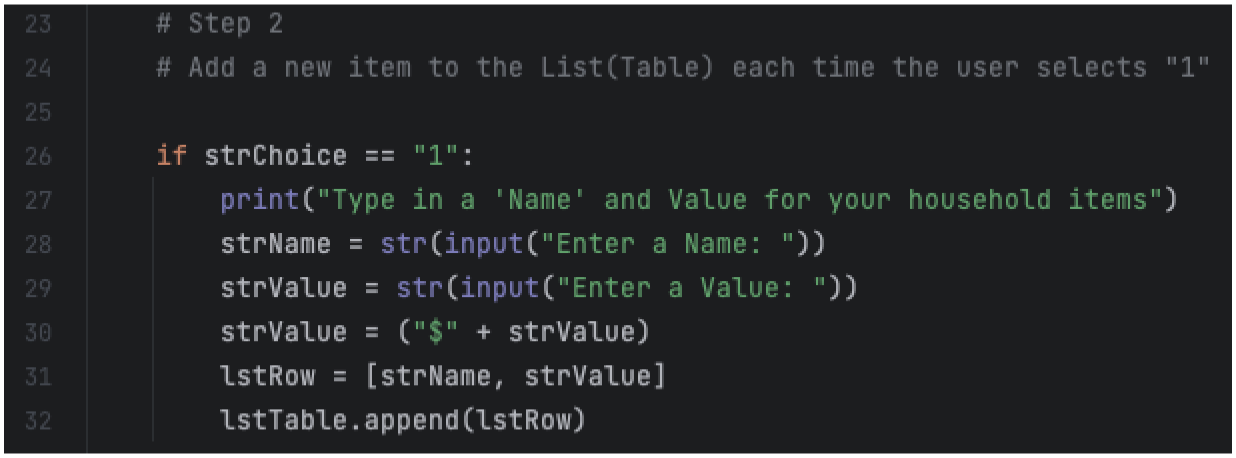


```
    # Step 5
    # Return error message if user makes an invalid selection

    else:
        print("Please choose only 1 2, or 3!")
```

Figure 3d : Error Handling

Option 1:

If the user selects option 1, the user will be prompted to enter two items: Name and Value. These will be stored in the previously created variables, ***strName*** and ***strValue***, respectively. I performed some light formatting on ***strValue*** to make it appear more like currency by prefixing ***strValue*** with a dollar sign ($). Both variables are then stored as a list in the variable ***lstRow***, and lastly ***lstRow*** is added to the ***lstTable*** list (table) using the **.append** list function.

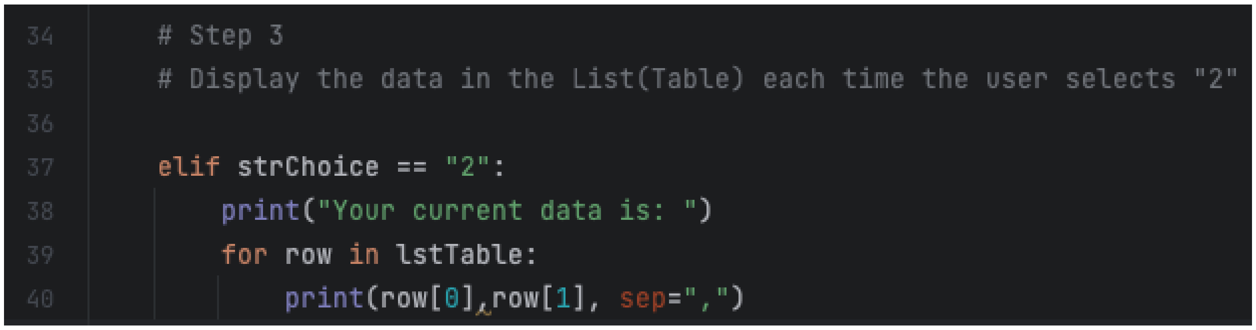


```
    # Step 2
    # Add a new item to the List(Table) each time the user selects "1"

    if strChoice == "1":
        print("Type in a 'Name' and Value for your household items")
        strName = str(input("Enter a Name: "))
        strValue = str(input("Enter a Value: "))
        strValue = ("$" + strValue)
        lstRow = [strName, strValue]
        lstTable.append(lstRow)
```

Figure 3a : Option 1 - Add Data to a List

Option 2:

Alternatively, if the user selects option 2, all of the current within the ***lstTable*** list (table) will be displayed. This part of the program uses a FOR LOOP to automatically iterate over each of the individual rows within ***lstTable***. This allows the program to get to the second level dimension (the individual ***lstRow*** items). The output is formatted to show both items: Name and Value with a comma delimiter.



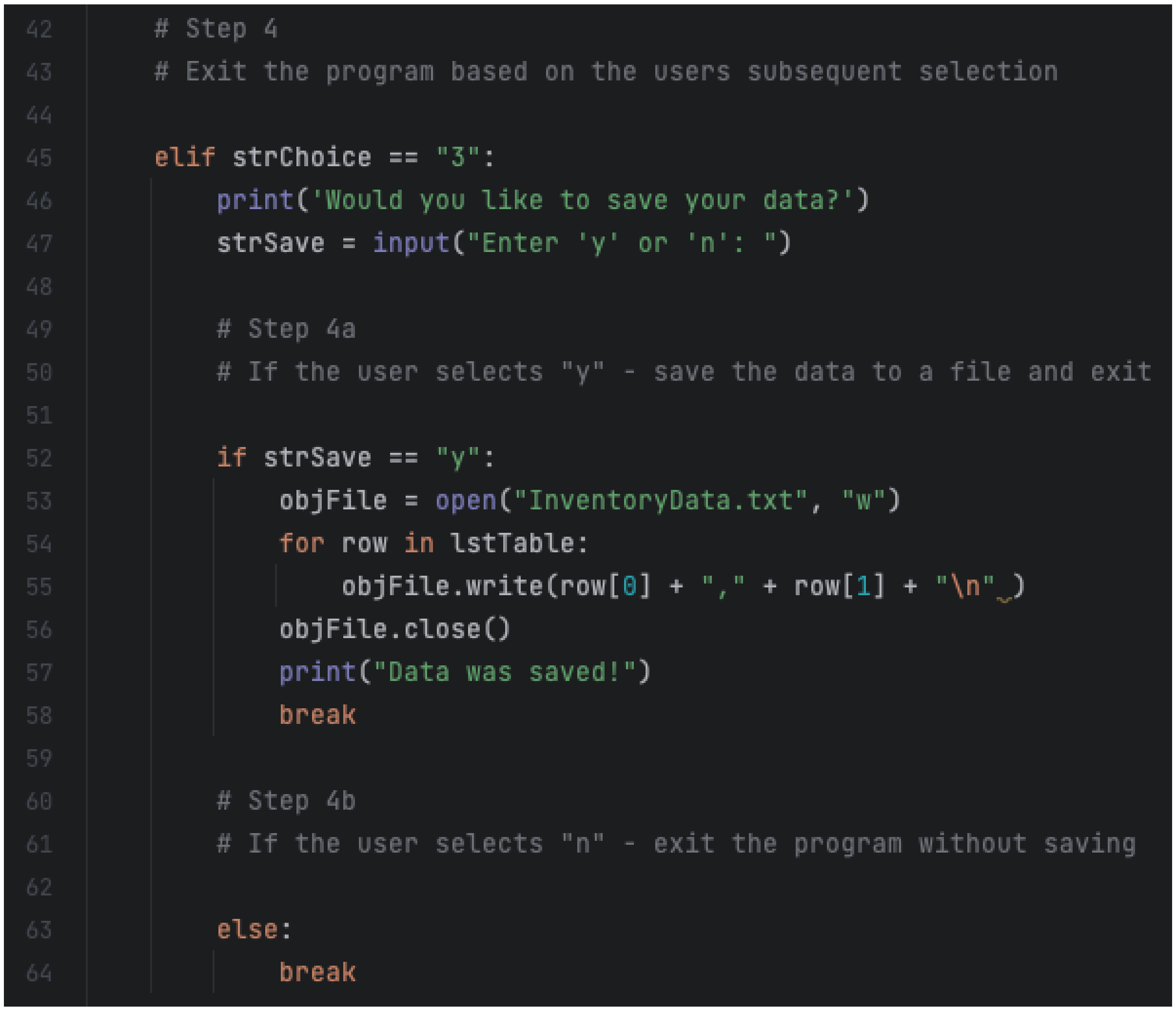
```
    # Step 3
    # Display the data in the List(Table) each time the user selects "2"

    elif strChoice == "2":
        print("Your current data is: ")
        for row in lstTable:
            print(row[0],row[1], sep=",")
```

Figure 3b : Option 2 - Display Table Data as Individual Records

Option 3:

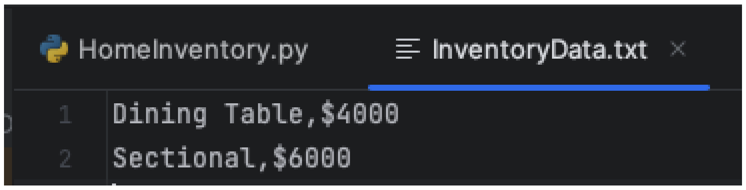
The last option initiates the termination of the program, but offers two sub-selections - either save the data to a file OR close the program without saving, and evaluates the behavior using a nested IF statement.

- **Exit and Save:** If the user selects "**y**" another FOR LOOP executes to iterate through each ***lstRow*** row within ***lstTable*** and saves the data as separate comma-separated records each on individual lines in a text file called ***InventoryData***. Additionally, a confirmation message is displayed to let the user know that the data was successfully saved.
- **Exit without Saving :** If the user selects "**n"** the program will simply end via the **break** statement.



```
    # Step 4
    # Exit the program based on the users subsequent selection

    elif strChoice == "3":
        print('Would you like to save your data?')
        strSave = input("Enter 'y' or 'n': ")

        # Step 4a
        # If the user selects "y" - save the data to a file and exit

        if strSave == "y":
            objFile = open("InventoryData.txt", "w")
            for row in lstTable:
                objFile.write(row[0] + "," + row[1] + "\n")
            objFile.close()
            print("Data was saved!")
            break

        # Step 4b
        # If the user selects "n" - exit the program without saving

        else:
            break
```

Figure 3c : Option 3 - Exit the Program, Save or Not?

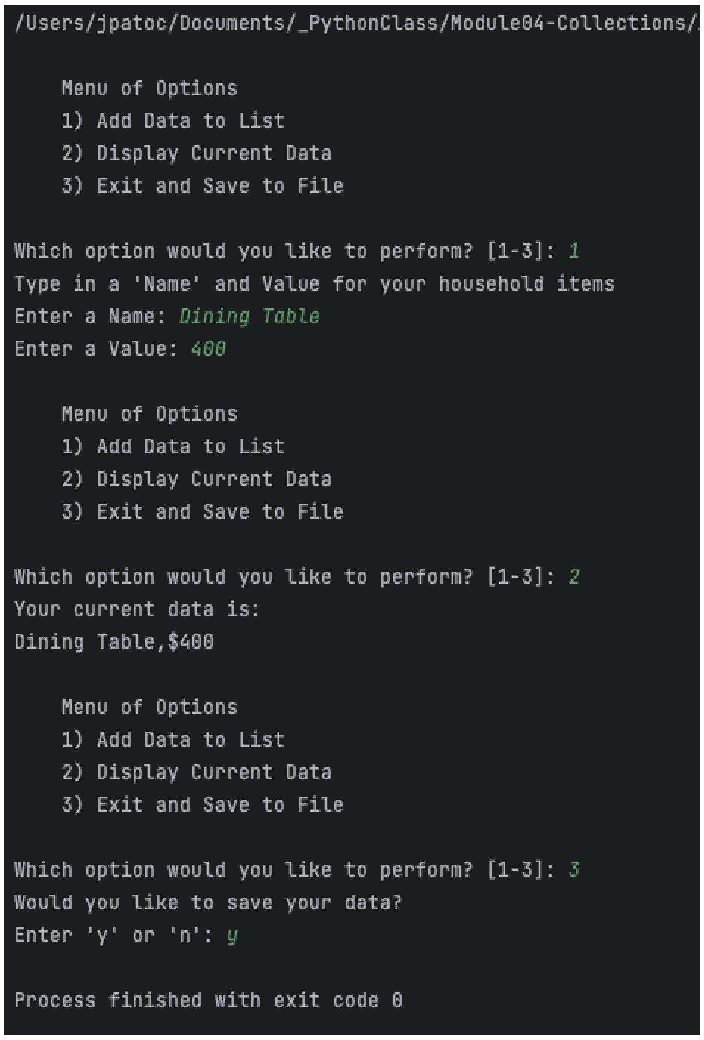


```
Dining Table,$4000
Sectional,$6000
```

Figure 4 : Data Saved in InventoryData.txt

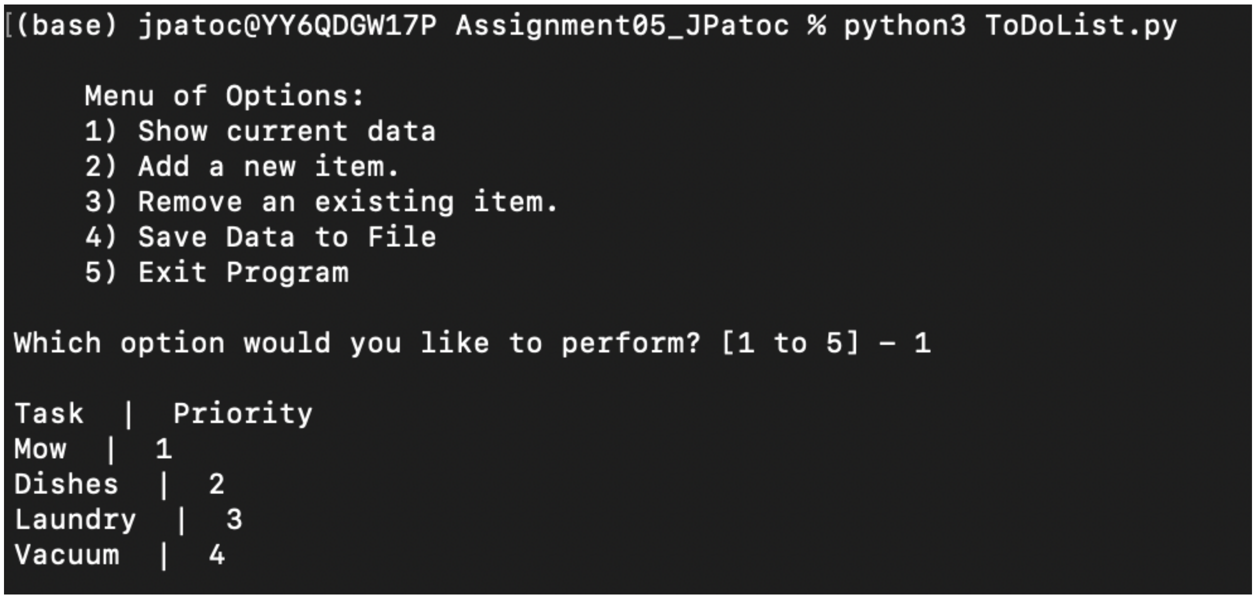
Testing the Program

PyCharm’s integrated environment allows for writing and testing the code in a single application. I can modify the code in the IDE and quickly see the output in the pane below. This makes development, testing, and debugging easier.



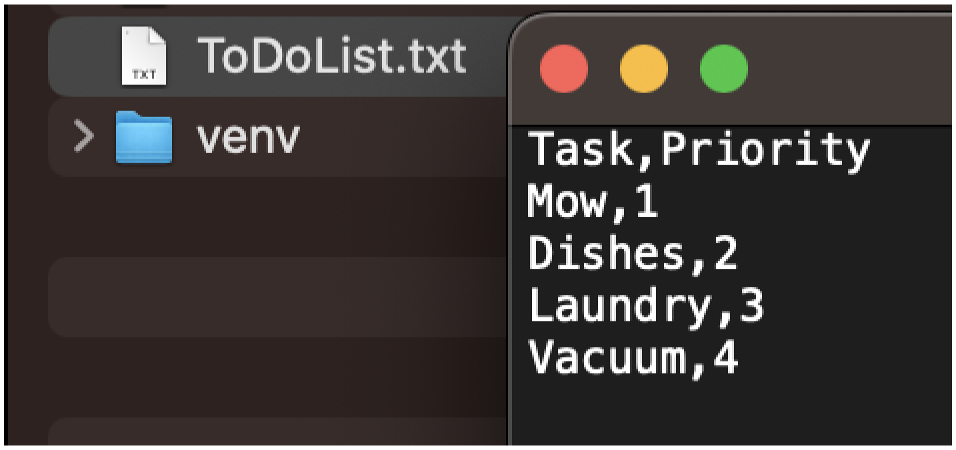
```
/Users/jpatoc/Documents/_PythonClass/Module04-Collections/

    Menu of Options
    1) Add Data to List
    2) Display Current Data
    3) Exit and Save to File

Which option would you like to perform? [1-3]: 1
Type in a 'Name' and Value for your household items
Enter a Name: Dining Table
Enter a Value: 400

    Menu of Options
    1) Add Data to List
    2) Display Current Data
    3) Exit and Save to File

Which option would you like to perform? [1-3]: 2
Your current data is:
Dining Table,$400

    Menu of Options
    1) Add Data to List
    2) Display Current Data
    3) Exit and Save to File

Which option would you like to perform? [1-3]: 3
Would you like to save your data?
Enter 'y' or 'n': y

Process finished with exit code 0
```

Figure 5a : Testing the Program in PyCharm



```
[(base) jpatoc@YY6QDGW17P Assignment05_JPatoc % python3 ToDoList.py

    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  |  Priority
Mow  |  1
Dishes  |  2
Laundry  |  3
Vacuum  |  4
```

Figure 5b : Testing the Program in Terminal



```
Task,Priority
Mow,1
Dishes,2
Laundry,3
Vacuum,4
```

Figure 5c : Verifying Data in Text Editor

Summary

The Home Inventory with Collections program is the most robust application we've built to date. It brings together many of the concepts we've covered of the course of the modules. This was undoubtedly the most time-intensive module I've gone through, but I appreciated being forced to learn about the nuances between strings and collections, as well as, tuples vs. lists. The behavior of each data type is unique and each offers their own set of advantages.

I’m pretty comfortable with IF ELSE logic because of my SQL background. I’m much newer to Loops though. Especially FOR LOOPs. It took me a while to get the hang of:

- what goes where
- when to indent (thankfully PyCharm helps with this)
- figuring out which dimension-level I’m working on

I also didn’t immediately grasp that the word tha comes right after “**for**” is effectively a variable declaration. I thought there was some set of reserved words that needed to be used like “col”, “row”, “item”, etc. It turns out I can use just about any word I choose.