

Chiyo Miyake

2023.02.15

IT FDN 110 A

Assignment 05

<https://github.com/bread-buddies/IntroToProg-Python>

Assignment 05: Using Dictionaries

Introduction

This assignment is to utilize different types of sequences to create a file that saves a To-Do list. We use what we learned from the last module, such as lists, and combine it with a new built-in sequence method, dictionaries.

To Do List

From the last module, we learned how to utilize lists to store data. Those lists are useful when holding any number of values and are able to display like a table. For this assignment, we use a list only to hold the table for the rest of the data. Since we want to be able to store the table and the information in it to a file, we first declare some data at the beginning of the script (Figure 1).

```
# -- 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
```

Figure 1: Declaring variables and constants as well as establishing a file to store the data.

Notice how we make a table using a list because we just want to store the rows into a grouping. For the actual input of data, we utilize a dictionary. Dictionaries are useful in that instead of having to call indexes, they are replaced by key subscripts. A dictionary is indicated by braces rather than the brackets we use for lists.

```

21 # -- Processing -- #
22 # Step 1 - When the program starts, load the any data you have
23 # in a text file called ToDoList.txt into a python list of dictionaries rows (like Lab 5-2)
24 # TODO: Add Code Here
25 data = open(objFile, "w") # open the file in write mode
26 dicHeader = {"Task": 'TASK', "When": "WHEN SHOULD THE TASK BE DONE?"}
27 dicRow = {"Task": "Make Dinner", "When": "5:00PM"} # add a task
28 lstTable.append(dicHeader)
29 lstTable.append(dicRow)
30 data.write(dicHeader['Task'] + ':' + dicHeader['When'] + '\n')
31 data.write(dicRow['Task'] + ':' + dicRow['When'] + '\n')
32 data.close()

```

Figure 2: Loading data into the file.

When we first start the program, we load any data we want as an example (Figure 2). We call on the text file that stores our To Do list. We then define a dictionary, and in my script, I format the text to include headers to show what the keys are defined as. The keys are first called and then the value is added following the colon. For the first task, we have 'make dinner' at '5:00PM' so that the user knows that we add a task and the time to do the task.

We add both rows of dictionaries to the list table we created previously. Then we make sure to record the dictionaries into the file by using the 'write' function.

User Input and Displaying the Output

Now that we have the introduction of the script done, we move onto asking for the user's input with the list of options the user has to choose from (Figure 3). We simply print a formatted string to provide the user with options. Figures 4 and 5 show the program menu display in PyCharm and the command prompt, respectively.

```

34 # -- Input/Output -- #
35 # Step 2 - Display a menu of choices to the user
36 while (True):
37     print("""
38     Menu of Options
39     1) Show current data
40     2) Add a new item.
41     3) Remove an existing item.
42     4) Save Data to File
43     5) Exit Program
44     """)
45     strChoice = str(input("Which option would you like to perform? [1 to 5] - "))
46     print() # adding a new line for looks

```

Figure 3: Beginning of the program loop and program menu.

```
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
```

Figure 4: Program menu ran in PyCharm.

```
C:\Users\pooch\_PythonClass\Assignment05>python Assignment05_Starter_ChiyoMiyake.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] - |
```

Figure 5: Program menu ran in the command prompt.

User Options

The user is now able to put in a corresponding number to an option on the menu. So let's take a look at the code for each option. We utilize an 'if-elif' statement to check what the user input is and for each number, the code will run the corresponding lines (Figure 6). The first option is to show the current list of data. We need to open the file as a readable file first, then for each dictionary row, we print the task and when the task needs to be done.

```
47         # Step 3 - Show the current items in the table
48         if (strChoice.strip() == '1'):
49
50             read = open(objFile, "r")
51             for dicRow in lstTable:
52                 print(dicRow['Task'], ': ', dicRow['When'])
53             continue
```

Figure 6: Coding for option 1 to show current data.

When we run this code, the terminal should display the initial data we loaded in the beginning of our script (Figure 2). The header and single task is printed in the command prompt (Figure 7).

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

TASK : WHEN SHOULD THE TASK BE DONE?
Make Dinner : 5:00PM
```

Figure 7: Selection of option 1 ran in the command prompt.

The next option we code for is for the user to be able to add a new item to the data list (Figure 8). We add a new dictionary row and ask the user to input each key with its new value. This new dictionary row is then added to the end of the table by 'appending' it to the table we already have. For my script, I added the code to show the updated table.

```
54         # Step 4 - Add a new item to the list/Table
55         elif (strChoice.strip() == '2'):
56             newRow = {} # new dictionary for user input
57             newRow['Task'] = input("Enter a Task: ") # user input task
58             newRow['When'] = input("When should this task be completed? ")
59             lstTable.append(newRow)
60             print() # adding a space
61             # Show the updated table
62             read = open(objFile, "r")
63             for dicRow in lstTable:
64                 print(dicRow['Task'], ': ', dicRow['When'])
65             continue
```

Figure 8: Code for the second option, adding a new item to the table.

When the code is run and we select option 2, the program will ask the user to enter a task and when the task should be completed (Figure 9). We can run the same code through command prompt and in both runs, the program displays the updated table (Figure 10).

```
Which option would you like to perform? [1 to 5] - 2

Enter a Task: Laundry
When should this task be completed? By noon

TASK : WHEN SHOULD THE TASK BE DONE?
Make Dinner : 5:00PM
Laundry : By noon
```

Figure 9: Running the script in PyCharm and adding a task to the list.

```
Which option would you like to perform? [1 to 5] - 2

Enter a Task: Drink water
When should this task be completed? Every 1-2 hours

TASK : WHEN SHOULD THE TASK BE DONE?
Make Dinner : 5:00PM
Drink water : Every 1-2 hours
```

Figure 10: Adding a task, code ran using the command prompt.

Now we want to be able to remove the last task we added. We can simply use a built in function, `.pop()`, to remove the last task we added (Figure 11). We could even add a user input to delete a specific index, or in this case key for dictionaries. Once the last item is removed, the script will print the updated table once again.

```
66         # Step 5 - Remove a new item from the list/Table
67         elif (strChoice.strip() == '3'):
68             lstTable.pop()
69             print("The last task has been removed.\n")
70             read = open(objFile, "r")
71             for dicRow in lstTable:
72                 print(dicRow['Task'], ': ', dicRow['When'])
73             print()
74
75             continue
```

Figure 11: Option 3 script to remove the last item added.

For the sake of example, I added a task to shower at 8PM (Figure 12). I then finished that task, so I selected option '3' to remove the last task added (Figure 13). The code can of course be ran in the command prompt, and so to be consistent, I removed the last task added (Figure 14).

```
Enter a Task: Shower
When should this task be completed? 8PM

TASK : WHEN SHOULD THE TASK BE DONE?
Make Dinner : 5:00PM
Laundry : By noon
Shower : 8PM
```

Figure 12: New task being added.

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

```
The last task has been removed.
```

```
TASK : WHEN SHOULD THE TASK BE DONE?
```

```
Make Dinner : 5:00PM
```

```
Laundry : By noon
```

Figure 13: Last task of showering has been removed and the updated table is displayed.

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

```
The last task has been removed.
```

```
TASK : WHEN SHOULD THE TASK BE DONE?
```

```
Make Dinner : 5:00PM
```

Figure 14: Same option selected, code ran in the command prompt.

Finally, we want to save the file and exit the program. Option '4' is selected when the user wants to save the data to the file. We do this with code by opening the while as editable file, then we write the updated table into the file (Figure 15). This is to ensure the data is properly updated into the text file.

```
76      # Step 6 - Save tasks to the ToDoToDoList.txt file
77      elif (strChoice.strip() == '4'):
78          save = open(objFile, "w")
79          for row in lstTable:
80              save.write(row["Task"] + ':' + row["When"] + '\n')
81          save.close()
82          print("Tasks have been saved to file.")
83          continue
```

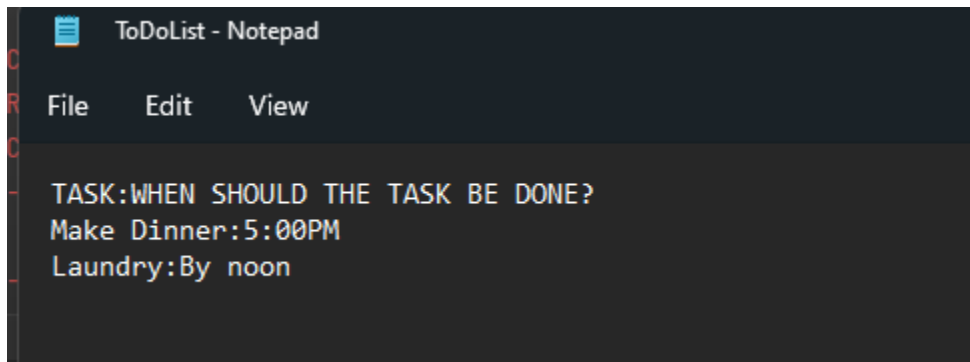
Figure 15: Code to save the data to the text file.

When the option is selected, the script will simply print a statement to tell the user the file has been saved (Figure 16). We can double check to make sure the text file is updated by opening the text file (Figure 17).

```
Which option would you like to perform? [1 to 5] - 4
```

```
Tasks have been saved to file.
```

Figure 16: Option 4 is run to save the file.

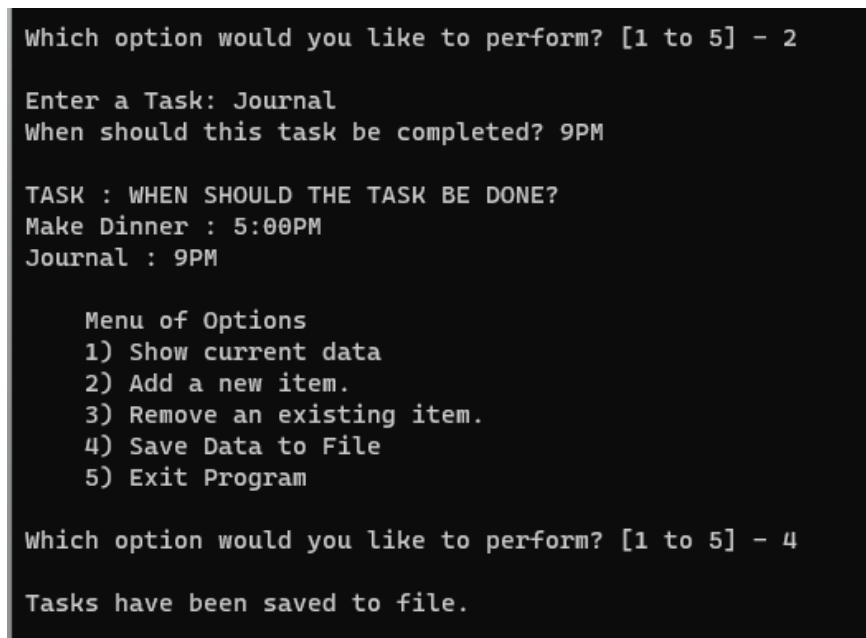


```
File Edit View

TASK:WHEN SHOULD THE TASK BE DONE?
Make Dinner:5:00PM
Laundry:By noon
```

Figure 17: Text file that successfully contains the data the user inputted.

To make sure the code can be run consistently in the command prompt, I added another task and saved the file (Figure 18). We can check the text file to make sure the data was saved successfully (Figure 19).



```
Which option would you like to perform? [1 to 5] - 2

Enter a Task: Journal
When should this task be completed? 9PM

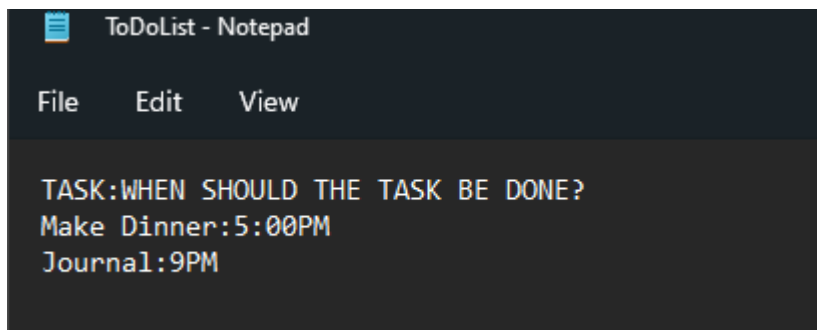
TASK : WHEN SHOULD THE TASK BE DONE?
Make Dinner : 5:00PM
Journal : 9PM

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

Tasks have been saved to file.
```

Figure 18: Running the program through the command prompt to ensure that the program works.

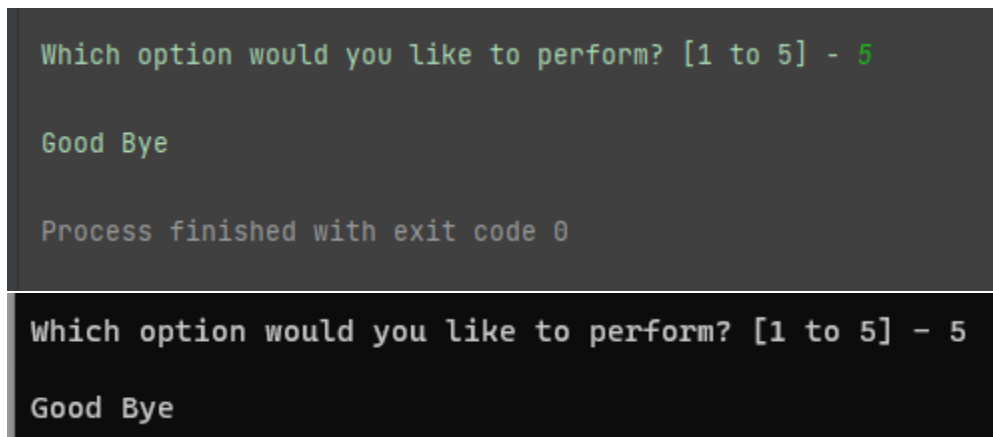


```
File Edit View

TASK:WHEN SHOULD THE TASK BE DONE?
Make Dinner:5:00PM
Journal:9PM
```

Figure 19: Text file that contains the data from the command prompt run.

Once the user is done editing the text file with their updated To Do list, option 5 can be selected to exit the program (Figure 20). In case the user makes a wrong input, the program will ask the user to input a valid option (Figure 21).



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

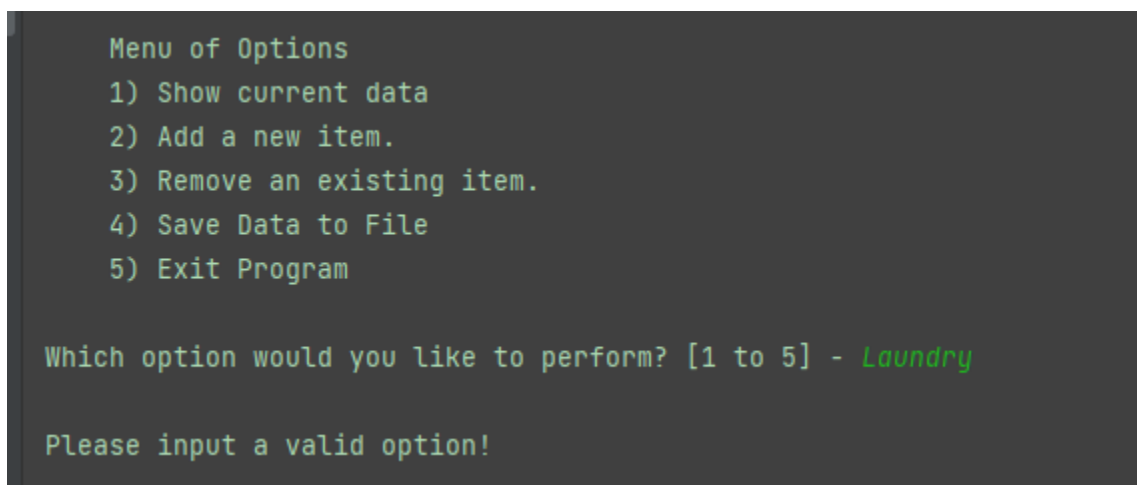
Good Bye

Process finished with exit code 0

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

Good Bye
```

Figure 20: Option 5 selected to exit the program both in PyCharm and Command Prompt.



```
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] - Laundry

Please input a valid option!
```

Figure 21: Program has invalid input, so the program will loop back to the menu options.

Summary

In summary, this program utilizes different techniques and functions that we have been learning up until now. We used a while loop to constantly loop through the program menu until the user is done with editing the text file with their tasks. In this assignment, we practice using dictionaries by utilizing the key subscripts to call on the values in the table rather than relying on indexes when using lists.