Boris Unigovskiy
August 8, 2022
https://github.com/bunigov/IntroToProg-Python/

IT FDN 100 B
Assignment 5

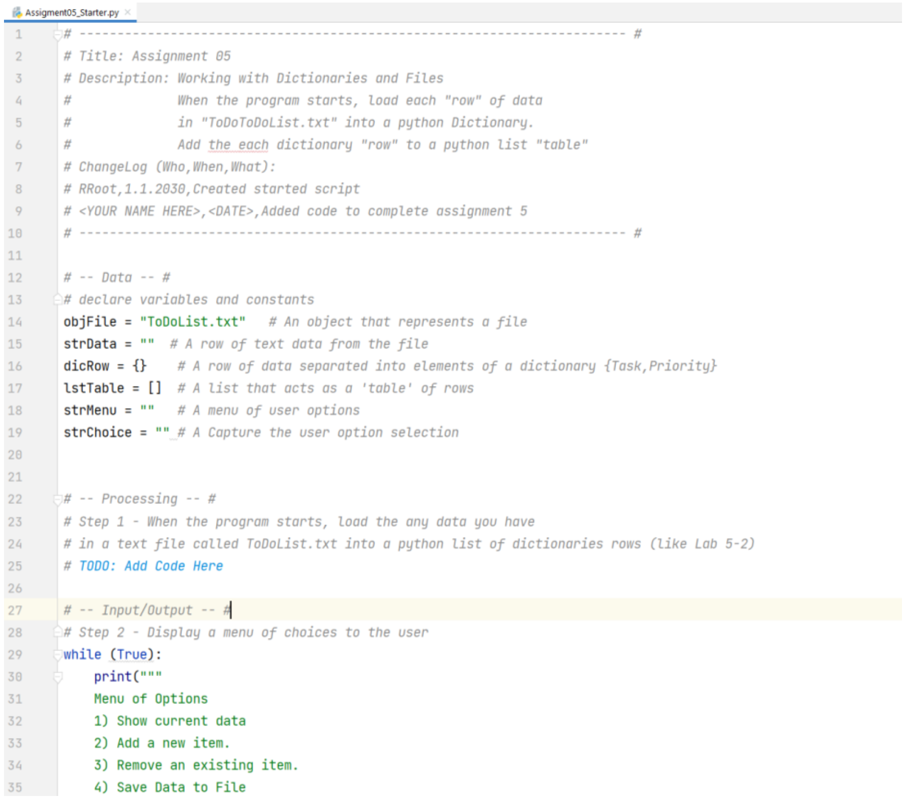
Working with Lists / Dictionaries

Introduction

In this week's lectures, modules and readings, we learned about Lists and Dictionaries (2 types of collections / sequences) and how to work with them to store data. This week's assignment challenged us to create a 'ToDo' list (of Tasks and their corresponding Priority) using these 2 aforementioned data structures.

Starter Template

The assignment requested that we begin with a starting template (as shown in **Figure 1** below) and modify it as needed.



Assigment05_Starter.py

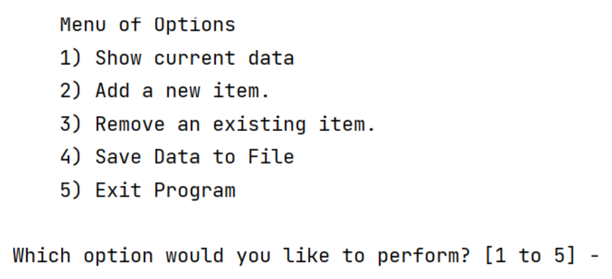
```python
# ---------------------------------------------------------------------- #
# Title: Assignment 05
# Description: Working with Dictionaries and Files
#              When the program starts, load each "row" of data
#              in "ToDoToDoList.txt" into a python Dictionary.
#             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

# -- 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


# -- 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)
# TODO: Add Code Here

# -- 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
```

Figure 1 *– Starting Template (snippet)*

The ask was to fill in the various 'ToDo' sections with code that will fulfill various capabilities of the program such as:

- Show current data in the file
- Add new items
- Remove existing items
- Save data to a file
- Exit the program

After updating the change log in the header, I read through the starter template to understand the structure of the code and the sections that I will need to fill in. I also ran the code to ensure that it did not come with any errors. Since only the display of the 'Menu of Options' had been coded (with no ability to exit the application)– the program simply repeated the prompt no matter what option was selected, as shown in **Figure 2** below.



```
    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] -
```

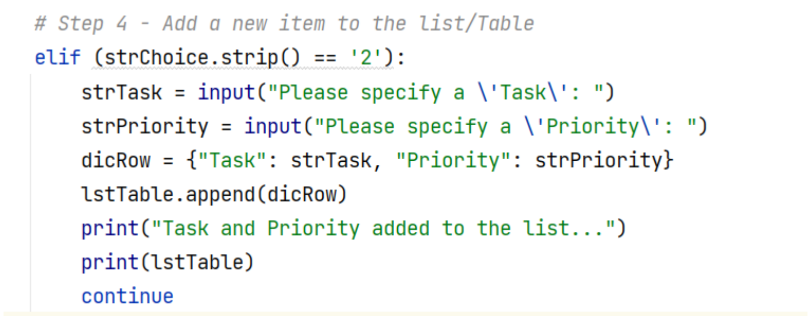
***Figure 2** – PyCharm UI - Menu Options*

Adding New Item

Since I had no starter file to work, I started with coding the functionality to add a new item (option 2). This would add the contents to temporary memory. I would then code option 4 to save the data to the file.

The pseudocode to do this would include getting user input for task name and priority, creating a "row to add" as a dictionary row, appending the dictionary row to the master table (stored as a list) and printing out information to the user (such as confirmation and display of what is currently in the list).

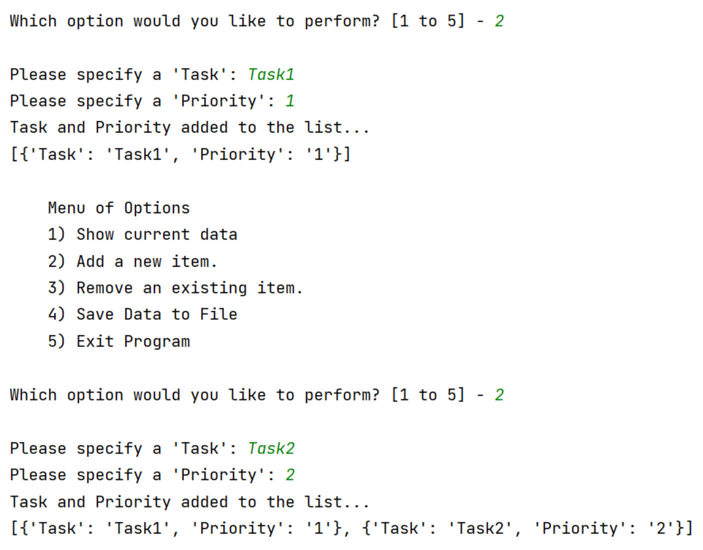
The resulting code is shown in **Figure 3** below.



```
# Step 4 - Add a new item to the list/Table
elif (strChoice.strip() == '2'):
    strTask = input("Please specify a \'Task\': ")
    strPriority = input("Please specify a \'Priority\': ")
    dicRow = {"Task": strTask, "Priority": strPriority}
    lstTable.append(dicRow)
    print("Task and Priority added to the list...")
    print(lstTable)
    continue
```

***Figure 3** – Code – Adding New Item (Tasks and Priorities)*

The result of the query (which confirms that the list of dictionaries was being generated) is shown in **Figure 4** below.



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

Please specify a 'Task': Task1
Please specify a 'Priority': 1
Task and Priority added to the list...
[{'Task': 'Task1', 'Priority': '1'}]

    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

Please specify a 'Task': Task2
Please specify a 'Priority': 2
Task and Priority added to the list...
[{'Task': 'Task1', 'Priority': '1'}, {'Task': 'Task2', 'Priority': '2'}]
```

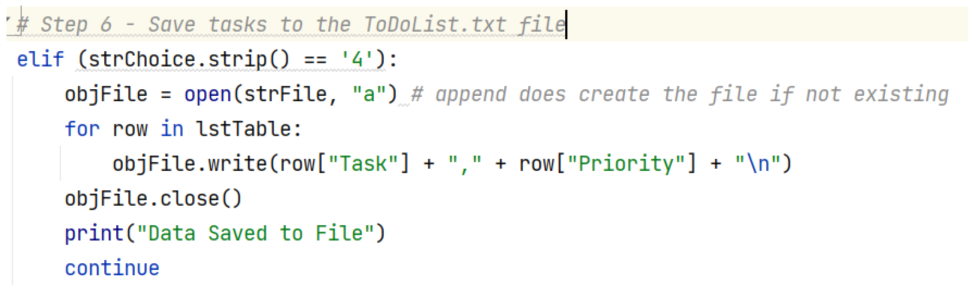
Figure 4 *– PyCharm UI – Adding New Item (Tasks and Priorities)*

Saving to a File

The next functionality I decided to tackle is option 4, the ability to save the tasks and priorities to a ToDoList.txt file.

The pseudocode to do this would include opening the file object in write or append mode, looping through the list of rows (writing a row to a file during each loop), closing the object connection and then giving some confirmation to the user.

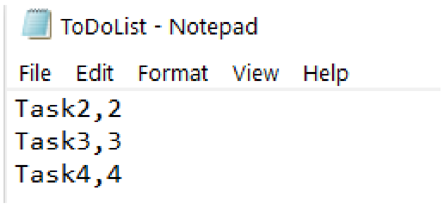
The resulting code is shown in **Figure 5** below.



```
# Step 6 - Save tasks to the ToDoList.txt file
elif (strChoice.strip() == '4'):
    objFile = open(strFile, "a") # append does create the file if not existing
    for row in lstTable:
        objFile.write(row["Task"] + "," + row["Priority"] + "\n")
    objFile.close()
    print("Data Saved to File")
    continue
```

Figure 5 *– Code – Storing to File (Tasks and Priorities)*

After entering some data using option 2, I confirmed that the file was indeed populated (appended) with the contents of the list, as shown in **Figure 6** below.



```
Task2,2
Task3,3
Task4,4
```

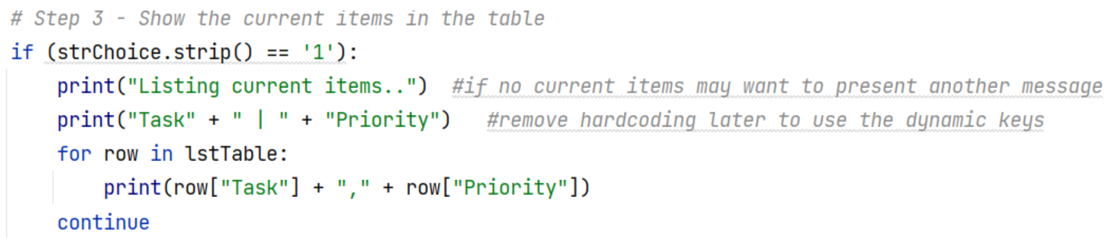
***Figure 6** – Confirmation – Storing to File (Tasks and Priorities)*

Showing Current Data

The next functionality I decided to tackle was option 1 – Show Current Data. This would allow easy visibility as to what is currently in the list (and help with troubleshooting by removing the need for extra print statements).

The pseudocode to accomplish this would be to print out the column headers and loop through the rows (using the print command at each iteration to print the row to the screen).

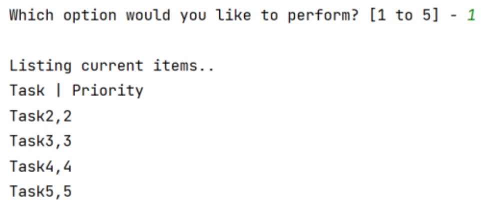
The resulting code is shown in **Figure 7** below.



```
# Step 3 - Show the current items in the table
if (strChoice.strip() == '1'):
    print("Listing current items..")  #if no current items may want to present another message
    print("Task" + " | " + "Priority")   #remove hardcoding later to use the dynamic keys
    for row in lstTable:
        print(row["Task"] + "," + row["Priority"])
    continue
```

***Figure 7 –** Code - Showing Current Data*

The result of the query is shown in **Figure 8** below. These 5 items were entered just beforehand using Menu Option 2 (Adding New Item).



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

Listing current items..
Task | Priority
Task2,2
Task3,3
Task4,4
Task5,5
```

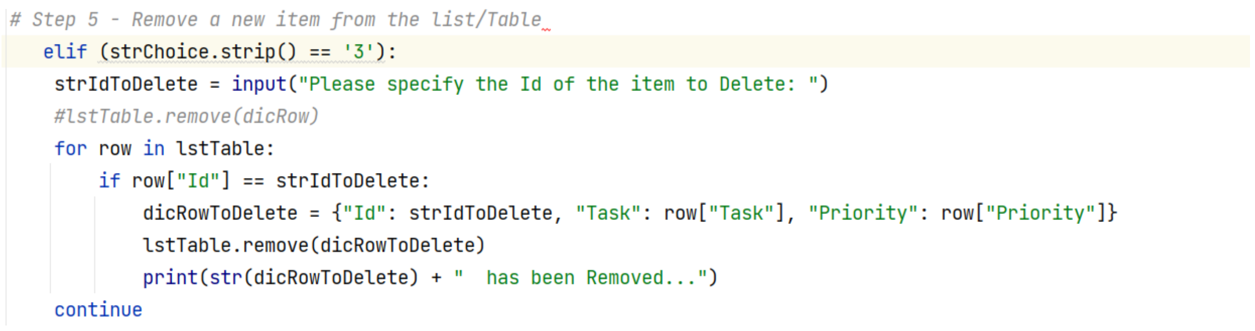
***Figure 8 –** PyCharm UI - Showing Current Data*

Removing an Item from the List

I then decided to tackle option 3- Remove an Existing Item. This one would be tricky as I would want a unique identifier for each row in case some of the tasks are duplicated. I decided that I would need to update my entry process to include an auto generated random Id (6 characters). Therefore each row that would be entered would have an Id column, a Task column and a Priority column.

The pseudocode for this was to ask the user to specify the item of the item to Delete (the user

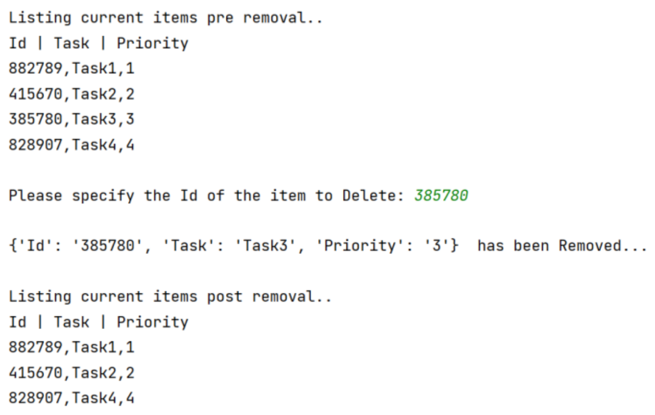
The resulting code is shown in **Figure 9** below.



```
# Step 5 - Remove a new item from the list/Table
   elif (strChoice.strip() == '3'):
    strIdToDelete = input("Please specify the Id of the item to Delete: ")
    #lstTable.remove(dicRow)
    for row in lstTable:
        if row["Id"] == strIdToDelete:
            dicRowToDelete = {"Id": strIdToDelete, "Task": row["Task"], "Priority": row["Priority"]}
            lstTable.remove(dicRowToDelete)
            print(str(dicRowToDelete) + "  has been Removed...")
    continue
```

***Figure 9 –** Code – Removing an Item*

Reusing some of the code from section ‘Showing Current Data’, I wanted to make the experience more user friendly by showing the list of available Ids for removal. Upon specifying the Id to remove, it advised the user of the row that was removed and the now current contents of the list (post removal), as seen in **Figure 10.**



```
Listing current items pre removal..
Id | Task | Priority
882789,Task1,1
415670,Task2,2
385780,Task3,3
828907,Task4,4

Please specify the Id of the item to Delete: 385780

{'Id': '385780', 'Task': 'Task3', 'Priority': '3'}  has been Removed...

Listing current items post removal..
Id | Task | Priority
882789,Task1,1
415670,Task2,2
828907,Task4,4
```

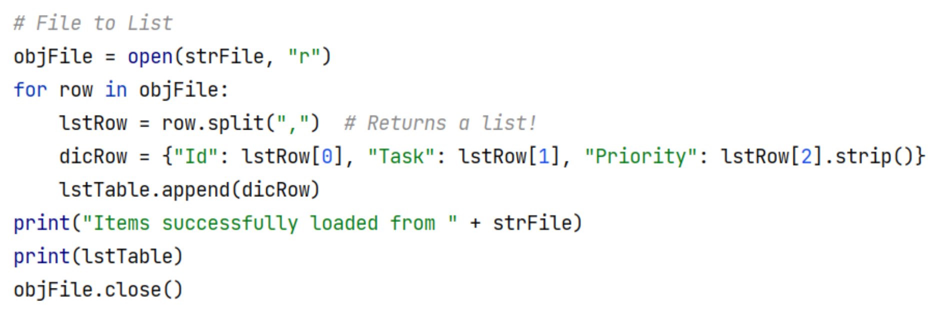
***Figure 10 –** PyCharm UI – Removing an Item*

Starting app with Loaded Data

The next step I wanted to work on was to load data from the ToDoList.txt text file when the application first opened. I used the starter code from Lab 5-2 to accomplish this.

The pseudocode for this was to open a file connection to the file, go through every row in the file to generate a list, generate a dictionary row from that list and append each of the dictionary rows to a table to create a master list of dictionary rows (forming a table).

The resulting code is shown in **Figure 11** below. It was put outside the while loop since the code needed to execute exactly once prior to the menu of options being available to the user.



```
# File to List
objFile = open(strFile, "r")
for row in objFile:
    lstRow = row.split(",")  # Returns a list!
    dicRow = {"Id": lstRow[0], "Task": lstRow[1], "Priority": lstRow[2].strip()}
    lstTable.append(dicRow)
print("Items successfully loaded from " + strFile)
print(lstTable)
objFile.close()
```

***Figure 11** – Reading data from the file at application start*

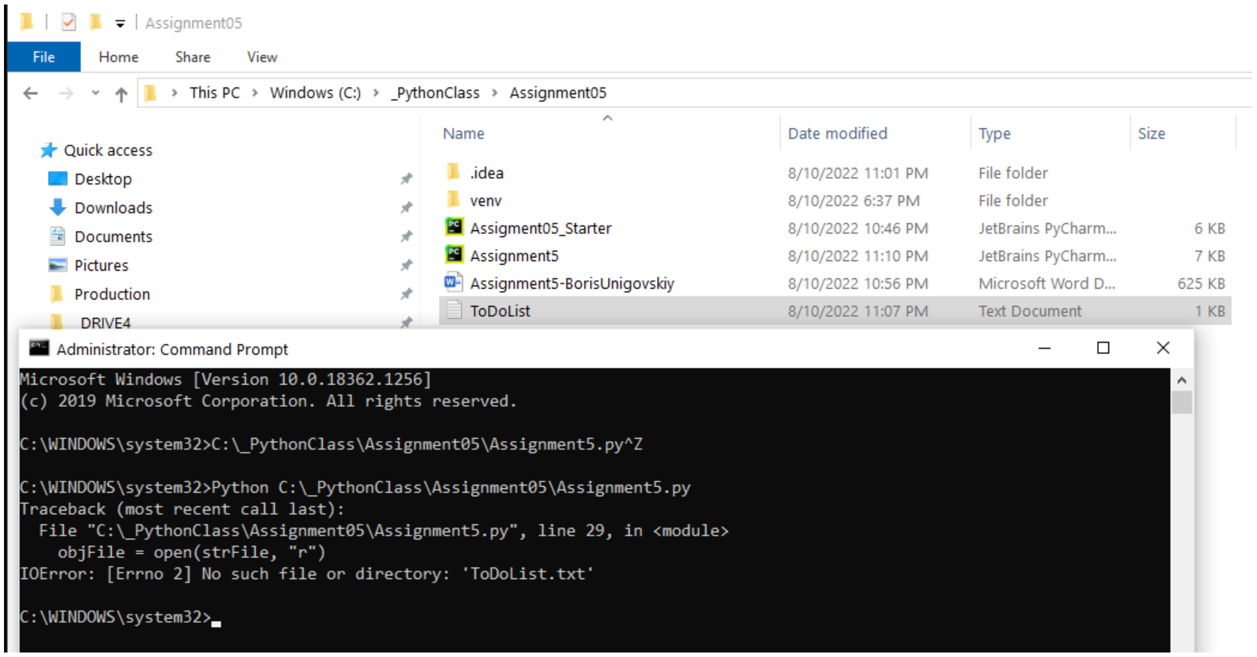
Exiting the app

Lastly, I tackled the last option – option 5 – Exit Program. The pseudocode and code for this one was rather simple (once user has chosen option 5): indicate to the user that the program has exited (using the *print* command) and exit the master while loop (using the *break* command) to stop the code.

Using the Command Prompt

I have provided several examples in the screenshots above of the application successfully working in PyCharm.

However, no matter what I did I was unable to get the application working in the Command Prompt. I kept getting this error even though clearly the file is in the directory with the expected name. As shown in **Figure 12** below, is my attempt to get it working in command prompt. I tried on several machines thinking it may be that immediate instance of Python but still could not resolve the issue. Code works perfectly in PyCharm so the issue must be specific to the command prompt.



Name	Date modified	Type	Size
.idea	8/10/2022 11:01 PM	File folder	
venv	8/10/2022 6:37 PM	File folder	
Asigment05_Starter	8/10/2022 10:46 PM	JetBrains PyCharm...	6 KB
Assignment5	8/10/2022 11:10 PM	JetBrains PyCharm...	7 KB
Assignment5-BorisUnigovskiy	8/10/2022 10:56 PM	Microsoft Word D...	625 KB
ToDoList	8/10/2022 11:07 PM	Text Document	1 KB

```
Microsoft Windows [Version 10.0.18362.1256]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>C:\_PythonClass\Assignment05\Assignment5.py^Z

C:\WINDOWS\system32>Python C:\_PythonClass\Assignment05\Assignment5.py
Traceback (most recent call last):
  File "C:\_PythonClass\Assignment05\Assignment5.py", line 29, in <module>
    objFile = open(strFile, "r")
IOError: [Errno 2] No such file or directory: 'ToDoList.txt'

C:\WINDOWS\system32>
```

Figure 12 – Proof of attempt to get it working in the Command Prompt

Boris Unigovskiy
August 8, 2022
https://github.com/bunigov/IntroToProg-Python/

IT FDN 100 B
Assignment 5

Summary

This week, we built off our existing knowledge of working with lists and files to also use dictionaries to enhance capabilities of our application.

Especially interesting additions this week was giving the user the ability to load existing data back from a file and to remove an added item, all giving more value and interactivity to the application.

Looking forward to diving into more advanced concepts next week.