Brittany Burdelsky
November 9-16, 2021
Fundamentals of Programming: Python
Assignment 05
https://github.com/bburdel/IntroToProg-Python

Working With Dictionaries and Files

Introduction

For this module, we were given pre-written code and asked to complete sections of that code. These modifications ultimately allow a user to work through a menu of options to create a to do list text file. Not only does this assignment give us practice working with someone else's code but it also exposes us to the way in which code can be organized and designed. Additionally, we were also introduced to GitHub[1] which is a platform for sharing, collaborating, and learning programming.

Separation of Concerns

This week we discussed a design principle for organizing and structuring code called the separation of concerns. The script we were provided to modify was already organized into sections: data (variable/constant declaration), processing, and input/output (user interaction). Within those sections there is care to provide helpful comments that guide the programmer to fill in code or instruct the programmer as to what should be done in that section. As a new programmer, I had been declaring variables on-the-fly and not necessarily organizing them upfront in the code. I found myself naturally declaring a variable in a different section of the code and then looping back to add it into the data section. I do think it will be challenging to keep discrete sections as coding becomes more complex but learning this now means I will be able to keep it in mind as I mature as a programmer. I did not add or move any of the inherited sections but that might be necessary in future scripts to keep things in discrete sections for data, processing, and presentation if possible.

Final Script

While I breakdown each section of the code, you can see the results of the code in the Terminal /PyCharm section. Feel free to skip ahead.

Header

This section of the script is self explanatory. A header notes the information about the code: who wrote it, who changed it, when it was written or changed, etc. I modified the header to

[1] You might be reading this documentation on GitHub so I won't really get into that.

include my information in the change log. While I will be highlighting my text in blue below to show my edits, know that those are my formatting changes and they do not reflect what the code looks like in PyCharm.

```
# ------------------------------------------------------------------------ #
# 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
# BBurdelsky,11.13.21,Added code to complete assignment 5
# ------------------------------------------------------------------------ #
```

Code Snippet 1: The header portion of the code. My modifications are in **blue**.

Data

This part of the script outlines and initializes variables that will be used in the proceeding sections. I also modified this portion of the code to include additional variables. I added variables that would help in opening a file and adding new tasks to the text file output.

```
# -- Data -- #
# declare variables and constants
strFile = "ToDoList.txt"   # An variable that represents a file
objFile = None  # 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 = ""  # A new task for the list
strPrior = ""  # A new priority
```

Code Snippet 2: My changes to the code in **blue**.

Processing

Step 1 - Load Data From File

In this section of the script, I read the contents of a text file by opening the file in read mode. I used a *for* loop to go through the file contents line by line and populate a dictionary row with those contents. A list is then appended so as to include those dictionary rows. Then the file is closed since we are finished reading from it. (I pre-populated the text file with a couple of tasks)

```
# -- 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
```

```
# -- Processing -- #
# Step 1 - When the program starts, load any data you have
# in a text file called ToDoList.txt into a python list of dictionaries rows
(like Lab 5-2)

objFile = open(strFile, "r")
for line in objFile:
    lstData = line.split(",")
    dicRow = {"task": lstData[0].strip(), "priority": lstData[1].strip()}
    lstTable.append(dicRow)
objFile.close()
```

Code Snippet 3: The processing section of the script. Edits are not highlighted but can be seen under the **blue** line in the text box.

Input/Output

This section is rather long but we can break it down one step (there are actually 7 steps) at a time. Here is the outline we were assigned:

```
# -- 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
```

An always true while loop contains conditional statements that will allow the user to loop through menu options until they decide to exit the program (step 7). In the following sections, I break down this start code with some brief explanations of how I accomplished the TODOs.

```
    # Step 3 - Show the current items in the table
    if (strChoice.strip() == '1'):
        # TODO: Add Code Here
        continue
    # Step 4 - Add a new item to the list/Table
    elif (strChoice.strip() == '2'):
        # TODO: Add Code Here
        continue
    # Step 5 - Remove a new item from the list/Table
    elif (strChoice.strip() == '3'):
        # TODO: Add Code Here
        continue
    # Step 6 - Save tasks to the ToDoToDoList.txt file
    elif (strChoice.strip() == '4'):
        # TODO: Add Code Here
        continue
    # Step 7 - Exit program
    elif (strChoice.strip() == '5'):
        # TODO: Add Code Here
        break  # and Exit the program
```

Step 2 - Menu

For this step, I made some formatting edits using print statements and removed periods in menu options 2 and 3 to keep formatting consistent and improve readability (I realize this is subjective). I will skip that code here fore brevity and since most new Python programmers get a grasp of the print statement with the good ole, print("Hello World!").

The menu is part of the while loop so that while true is a true condition, the menu will keep displaying and the user will be prompted to choose a menu option.

Step 3 - Show Data

This is the first *if* conditional statement encountered in the while loop. This statement, along with all the other nested *elif* statements, says that while the user choice is equivalent to that value (a string input choice of a number stripped of anything but the number)— execute the following code.

In order to show the user data, I needed to be able to read what data was in the text file (ToDoList.txt) I created in Step 1. I opened that file in read mode and used a *for* loop to grab each dictionary row and output that data in a friendly format. For a sanity check, and so the user could see the data was stored in a dictionary, I printed out the unformatted dictionary data.

Step 4 - Add Data

To add data, I asked the user for input. I used the input function (and converted their entry to a

```
# Step 3 - Show the current items in the table
# File to List
if (strChoice.strip() == '1'):
    objFile = open(strFile, "r")
    print("Formatted data:")
    for row in objFile:
        t, p = row.split(",")
        dicRow = {"task": t.strip(), "priority": p.strip()}
        print("\t" + dicRow["task"] + ", " + dicRow["priority"].strip())
    print("Raw data:")
    print(lstTable)
    print()  # adding a new line for presentation
    objFile.close()
    continue
```

string since you can only write strings to files using the write mode of the write function) to ask the user to provide a new task and priority level for that task. The input data is then appended to the existing list (table) comprised of dictionary rows populated in Step 1. As a check for the user, I print out what their to do list looks like so far by making use of a *for* loop to iterate through the dictionary rows in the list.

```
# Step 4 - Add a new item to the list/Table
elif (strChoice.strip() == '2'):
    print("Let's add a task to your list.\n")
    strTask = str(input("\tPlease enter a task: "))
    strPrior = str(input("\tPlease enter this task's priority [High, Medium,
Low]: "))
    lstTable.append({"task": strTask, "priority": strPrior})
    # loop through to print contents of list
    dicRow = {"task": strTask, "priority": strPrior}
    print("\tYour to do list now looks like this: ")
    for dicRow in lstTable:
        print("\t"*2 + dicRow["task"] + ", " + dicRow["priority"].strip())
    print()  # adding a new line for presentation
    continue
```

Step 5 - Remove Data

To remove data from the list, I asked the user to enter what task they want removed. I was specific in asking for the exact phrase and did not build in any fail safes/contingencies in case

the entered something erroneous or not in the dictionary. That said, I used an *if* statement nested in a *for* loop to loop through the dictionary entries to find a match for that task if what the user entered matched the dictionary task key. Again, as a check I printed what the to do list looked like with this task entry removed.

```
# Step 5 - Remove a new item from the List/Table
elif (strChoice.strip() == '3'):
    strData = str(input("Enter the complete text of the task to be removed:
"))
    for dicRow in lstTable:
        if dicRow["task"].lower() == strData.lower():
            lstTable.remove(dicRow)
    print("\tThis task was removed from the To Do List\n")
    print("The remaining tasks on the list are: ")
    for dicRow in lstTable:
        print("\t" + dicRow["task"] + ", " + dicRow["priority"].strip())
    print()  # adding new line for presentation
    continue
```

Step 6 - Save Data to File

To save data in a text file, I opened the file in write mode and used a *for* loop to iterate through the dictionary rows of the list and write each line to the text file. Again, I presented the data to the user as a check to make sure it resembles the contents they were expecting.

```
# Step 6 - Save tasks to the ToDoToDoList.txt file
elif (strChoice.strip() == '4'):
    objFile = open(strFile, "w")
    for dicRow in lstTable:
        objFile.write(dicRow["task"] + ", " + dicRow["priority"] + "\n")
    objFile.close()
    print("Your edits have been saved.")
    print("\tThe contents of the saved list are:", lstTable)
    # print()  # adding a new line for presentation
    continue
```

Step 7 - Exit Program

The final step exits the programming by breaking (break) the loop. I print an input message to the user to make the exit slightly smoother. For good measure, I closed the file again.

```
# Step 7 - Exit program
elif (strChoice.strip() == '5'):
    # pause to confirm exit
    print(input("You are closing the program. Press Enter to 'exit.'"))
    objFile.close()
    break  # and Exit the program
```

Terminal/PyCharm

Here are the results of running my program in Terminal.app and PyCharm.app (I am using a Mac). You can see the menu of options looped through and presented to the user after each choice's task is completed.

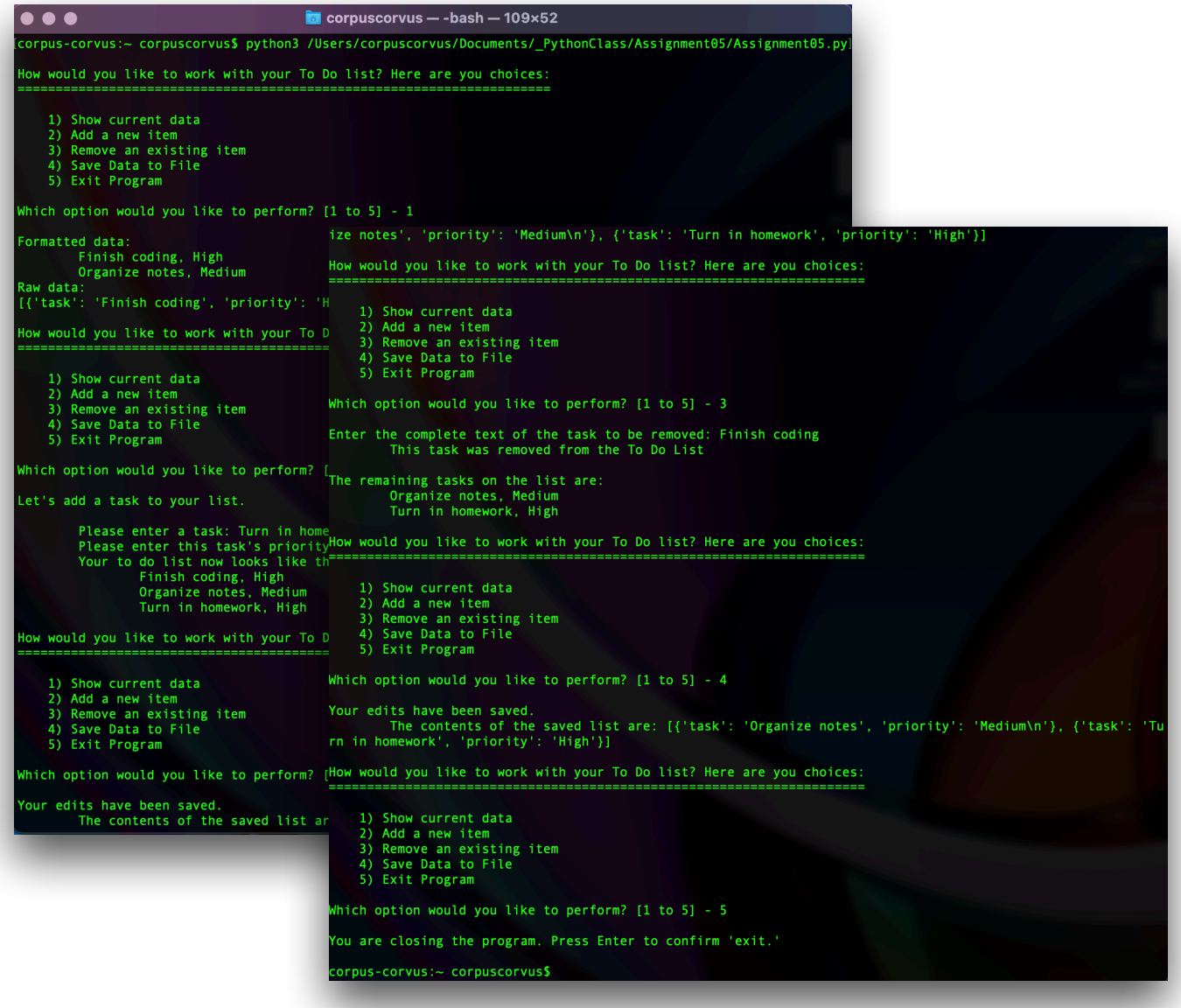

Figure 1. The program running in command prompt.

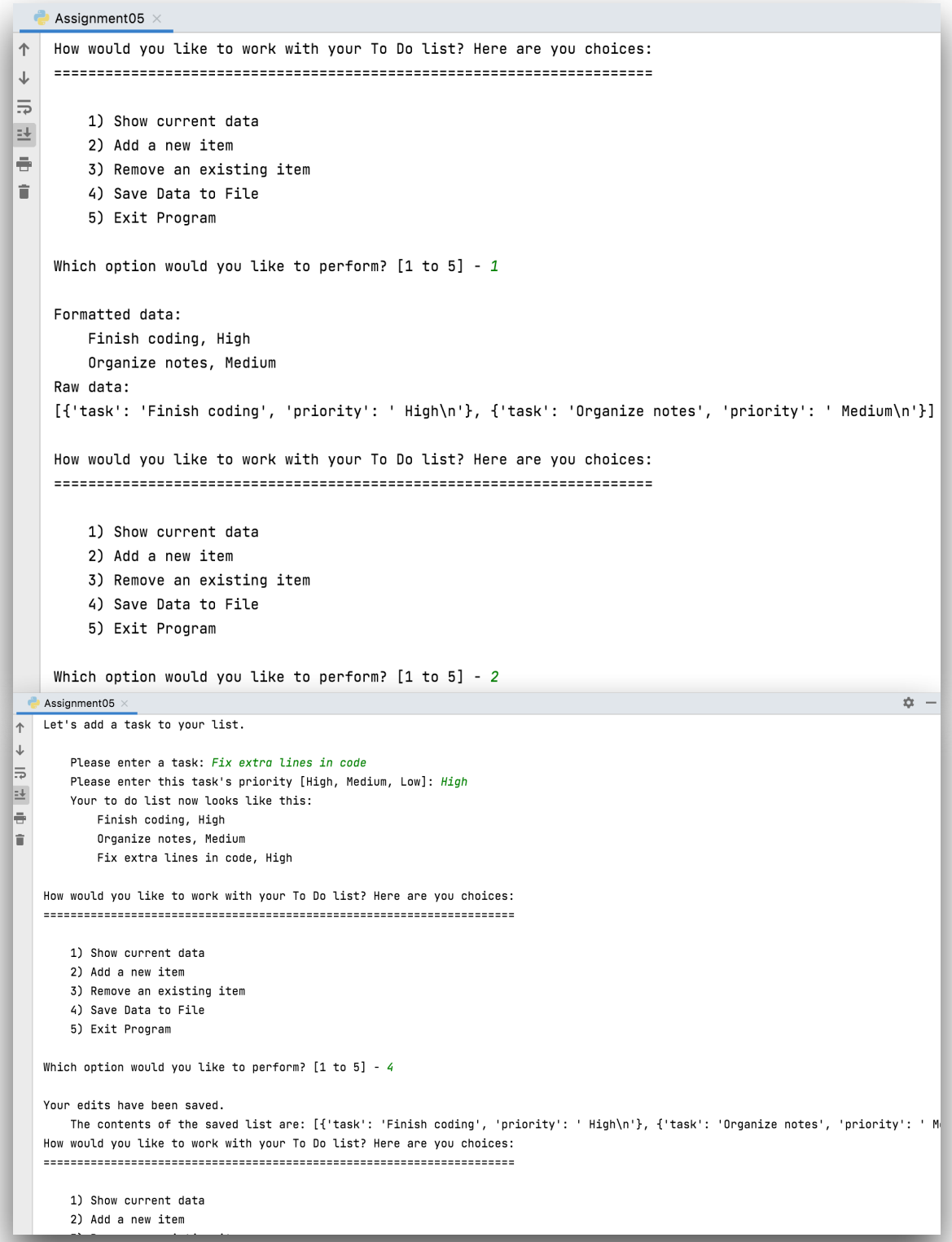

Figure 2. The program running in PyCharm

Summary

I knew this assignment would be a challenge but I under estimated how much of a challenge it would be. I leaned on the lecture review more that I have in the past. I did find that it was easier to address each step individually, get that portion of the code functioning, and then move on to the next step. I also wrote some bare bones code to get everything functioning and then filled it out with print statements, I felt, guided the user through the program. I also added some formatting by way of extra, empty print lines, new lines, and tabs. While the code runs and looks to be formatted nicely, it does output carriage returns that are unexpected. The draft code I wrote output data to file in an expected format (each dictionary row on a line with no empty rows between each entry). Then I enhanced that code with print statements. Something else is going on that I was not able to troubleshoot which results in a text file that has extra lines between dictionary rows[2].

[2] Alas, beauty is in the eye of the beholder.