Alex Frain

February 13, 2021

IT FDN 110 B – Foundations of Programming, Python

Assignment 05

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

To Do List Script

# Introduction

The Python script created in this assignment reads in a To Do list from a text file on the user’s computer and gives them the ability to modify the list by adding tasks, removing tasks, and writing the tasks back to the text file on their computer. Python’s dictionary data type is introduced in this script to hold the individual data pieces which are then added to a list to create a table of dictionary items. While the script also reinforces concepts from the prior weeks like lists, looping, conditional statements, and writing to files it also introduces a general computer science concept called Separation of Concerns for better code organization.

# Creating the Script

We were provided with a starter .py file that contained our script header and the variable declarations we were to use in the script. An overall script structure was also provided with the main body inside a while loop containing conditional statements for each of the menu of options the user could select from with pseudocode that provided us with general instructions on what was expected to be done in each section.

This entire structure was broken up into sections using the computer science concept known as Separation of Concerns. Separation of Concerns is looked as “the idea that a software system must be decomposed into parts that overlap in functionality as little as possible.” (<https://effectivesoftwaredesign.com/2012/02/05/separation-of-concerns/>, 2021)(External Link) The script was decomposed into three basic sections: Data, Processing, and Input.

## Data Section

The Data section is where a majority of variables and constants are declared that will be used by the script. This was already done for us in the starter script we were provided (Figure 1).

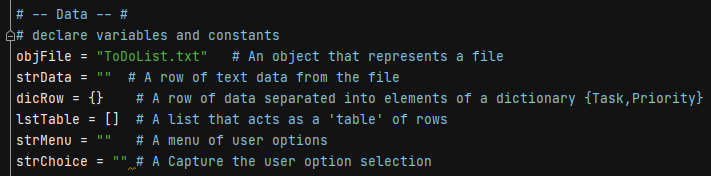


Figure : Variable declarations

## Processing Section

The next area of the script is the Processing section. This is where Step 1 (of 7) of the pseudocode was located which entailed reading in data from a text file on the user’s computer and placing it into a list of dictionaries. A Python dictionary is another data type, like a list, used for storing multiple items of the same or differing type. Unlike a list it uses a *key: value* pair where the *key* which is a string acts as the index for the specific value.

The data in the text file is a “Task” and its associated “Priority” separated by a comma with each pair of data on its own line. Both those terms are set as the dictionary keys for the associated values contained in the file. The file is opened in read mode and assigned to a file object. I then loop through each row of data in the file using a *for* loop where Python string’s *split()* method is used split the string into a two item list using a specified value which, in this case, is the comma separating the task and its priority. The two values in the new list are then assigned to a dictionary where the first item at index 0 is associated to the key “Task” and the second item at index 1 is associated with the key “Priority”. I should also note that since there is a new line entry in the text file after the priority value I use the string’s *strip()* method to strip the new line from the end of the priority value when it’s brought into the dictionary. This new dictionary item is then added to a list in order to create a list of dictionaries. Once the *for* loop is complete I close the text file. The completed section can be seen below (Figure 2).

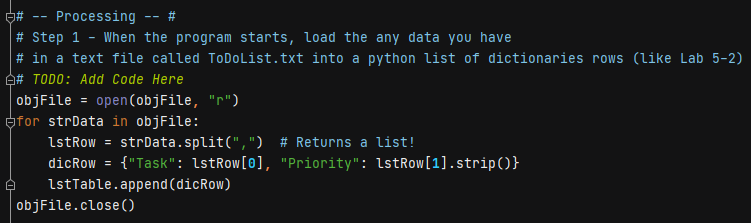


Figure : Read in data from a text file into a list of dictionaries

## Input/Output Section

The final section, Input/Output, handles all user input and any output to the screen or to the text file on the user’s computer. The remaining 6 steps of pseudocode provided in the starter script handles displaying a menu of options to the user with regards to their To Do list and provides the functionality behind each menu selection and is further explained below.

### Step 2 – Display a menu of choices

The entire Input/Output section is contained within a *while* loop that evaluates the Boolean value True which means this loop will run when the script is run. In this step a print statement is used to display a list of menu choices to the user for how they would like to interact with their To Do list. They are prompted to enter a specific number corresponding to a given menu choice and this input is captured, converted to a string using the *str()* function and assigned to a variable. Step 2 can be seen below (Figure 3).

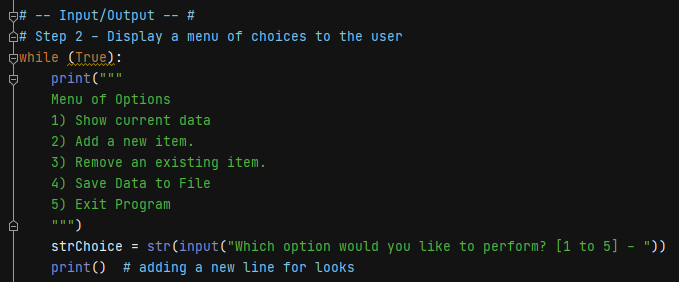


Figure : Display menu of choices

### Step 3 – Show the current items in the table

The functionality for each menu choice are part of *if-elif* conditional statements that compares the menu option input by the user to a literal string value for equality to determine whether the code for that menu selection gets run.

The first menu option allows the user to see the current data in their To Do list which has been read into a list table created by the script. As items tasks get added and/or removed from the list table this option will show those changes whenever this option is selected. Since this is the first menu option it is placed in the initial *if* statement which then contains a *for* loop to loop through the list table and print each row’s contents (Figure 4). Again, each row is a Python dictionary so I reference the dictionary *key* in order to get the associated *value* to be printed.

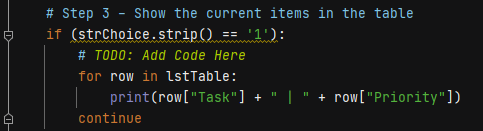


Figure : Show items in the list table

### Step 4 – Add a new item to the table

The second menu choice allows the user to add a new task and its priority to their To Do list and is contained within an *elif* statement. The task and its priority are captured via two separate input statements as strings that are assigned to their own variables. Those string variables are then added to a new dictionary item by pairing them with their respective keys, “Task” and “Priority”. The dictionary item is then added to the existing list table using the *append()* method. This all can be seen in the image below (Figure 5).

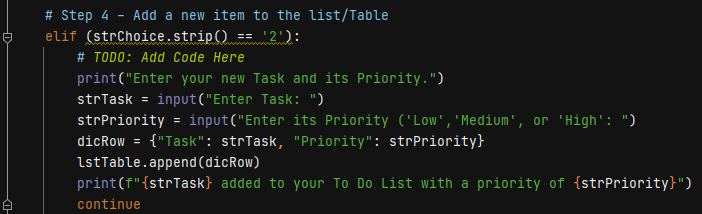


Figure : Add new task to the list table

### Step 5 – Remove an item from the table

The next menu option allows the user to remove a task from their To Do list and is contained within another *elif* statement. An *input* statement is used to capture the task the user types in and it’s assigned to a variable. I then use a *for* loop to loop through each dictionary in the list table and use an *if* statement within the loop to compare the user’s input to the task value paired with the “Task” key for equality. I also take case sensitivity into account by using the string’s *lower()* method to convert the user’s input and the dictionary value to lower case when checking for equality. If there is equality this dictionary is passed into the list’s *remove()* method in order to remove it from the list table. This can be seen in the image below (Figure 6).

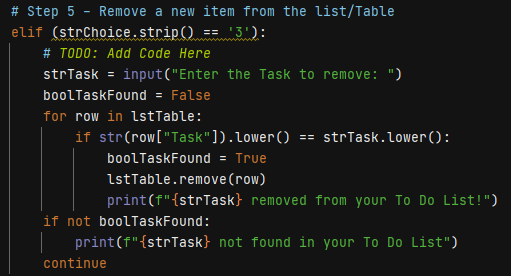


Figure : Remove a task from the list table

I also wanted to provide a message to the user in case the task they entered was not found in the list table. I initially tried doing this using an *else* statement that was part of the conditional *if* statement in the *for* loop. When I tried this, the message that would display to the user indicating the task was not found would be displayed each item through the *for* loop when the task was not found. This was not the desired result I was looking for as I just wanted the message to display once if the task was not found.

The method I landed on to accomplish this was to declare a Boolean variable that is used as a flag that I initially set to False before entering the *for* loop. If the task the user entered is found I set the Boolean variable to True within the *if* statement in the *for* loop. Once the loop is done running I use another *if* statement to check if the Boolean variable is False which would indicate the item was not found in the list table and print a message to the user indicating such.

### Step 6 – Save tasks to a text file

The fourth menu option allows the user to save their current To Do list to a text file on their computer and is contained in another *elif* statement. I open the same text file that the script initially reads from but this time it’s in write mode rather than read mode. I don’t want to use append like I did in the prior assignment since I want to overwrite any data in the file with the current To Do list the script generated and is being held in memory.

I use another *for* loop to loop through the list table to get each dictionary item. I use a f-string to format the dictionary’s “Task” value and “Priority” value to a single string that gets assigned to a variable. This variable is then passed to the Python File’s *write()* method to write this string to a file. Once the loop strops running I close the file object and display a message to the user indicating the data was saved to the file. The completed code for this section can be seen below (Figure 7).

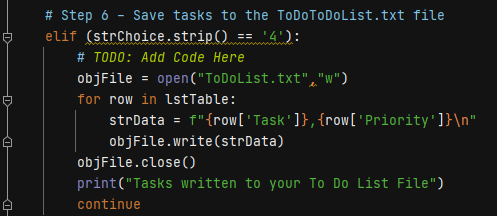


Figure : Save tasks to file

### Step 7 – Exit program

The last option in the menu is to exit the program and is contained in the final *elif* statement. The only lines of code within this step is a print statement displaying a message to the user that they have exited the script followed by a *break* statement which breaks out of the *while* loop causing the script to end (Figure 8).

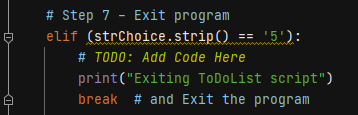


Figure : Exit program

# Running the Script

## Running in PyCharm

I ran the script in PyCharm to confirm everything ran as expected. Figure 9 shows the existing ToDoList.txt file and its contents before the script is run.

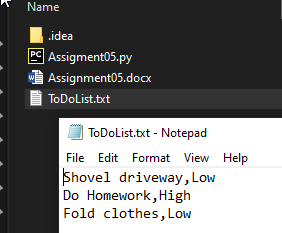


Figure : Initial text file being read into script

Figure 10 shows the script running in PyCharm with each menu option being selected.

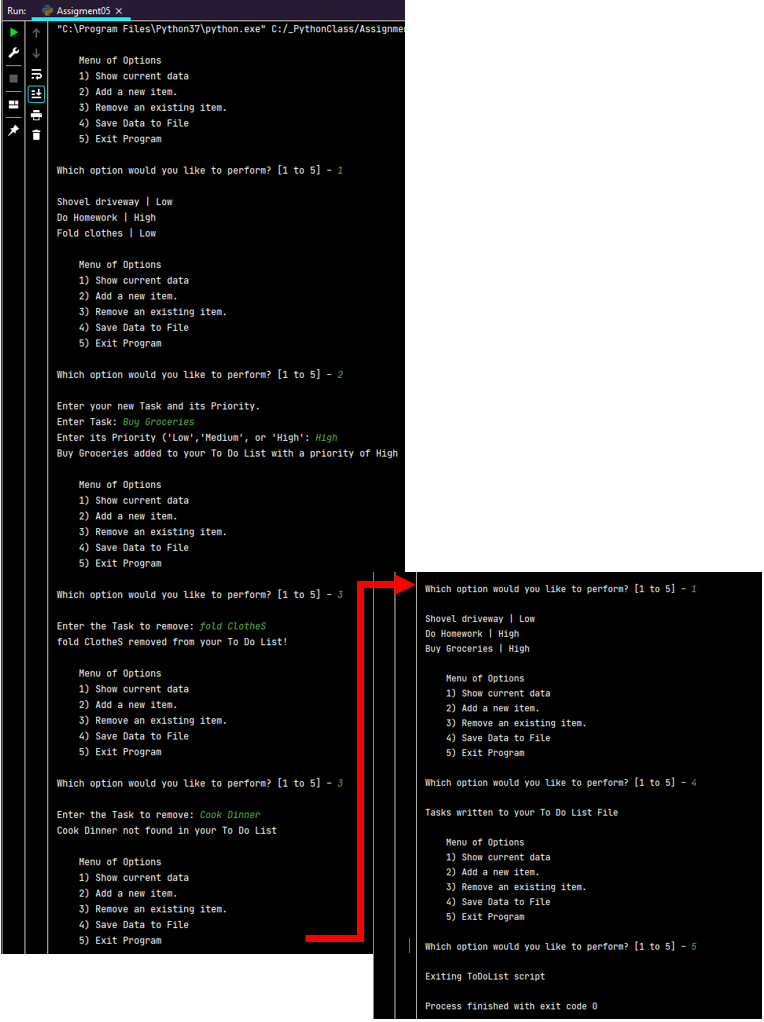


Figure : Script running in PyCharm

Figure 11 shows the ToDoList.txt file after the script wrote the updated list to the file.

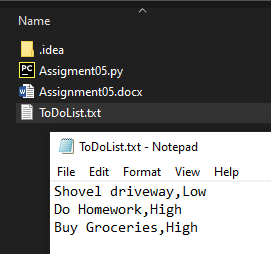


Figure : Resulting txt file after script run

## Running in Windows Command line

After confirming the script ran successfully in PyCharm I ran it from Windows Command line. The script read in the updated text file (Figure 11 above) from the PyCharm run through. I went through all the menu options confirming it ran successfully (Figure 12). Also shown is the newly updated text file after the script ran (Figure 13).

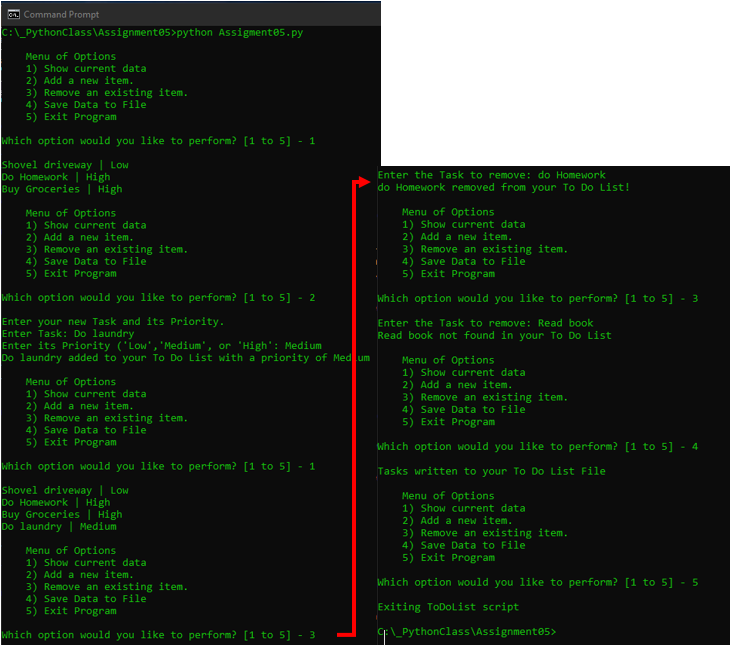


Figure : Script running in Windows Command line

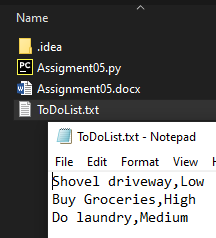


Figure : Resulting txt file after script was run

# Summary

In this assignment my script read a comma separated To Do list with priorities in from a text file on a computer into Python dictionary items and then these dictionary items were added to a Python list to create a table of dictionaries. A dictionary is a new data type that was introduced to us this week that is similar to a list in that it allows for the storage of multiple items within a single variable. The primary difference between the two is that a dictionary uses a *key: value* data pair where the *key* which is string type acts as the index. The script then gave the user the ability to add tasks to the list, remove tasks from the list, view the list, and write the list back to the text file on their computer. A *while* loop allows the user to keep looping through the menu options until they decide to end the program (via one of the menu selections) and the code behind each menu selections is contained within *if-elif* conditional statements. A new computer science concept was also introduced in this script called Separation of Concerns which is a way to organize code into purposeful sections whose functionality does not cross into the other sections if possible.