Craig Muth

11/13/2022

IT FDN 110 A Au 22: Foundations Of Programming: Python

Assignment 05

<https://github.com/cjmuth/IntroToProg-Python>

TODO List - Read/Write an External File, and the GitHub Repository

# Introduction

The goal of this project is to create a TODO list program, that will read data from an existing text file, allow the user to view the existing data, make changes, and save the data back to the text file.

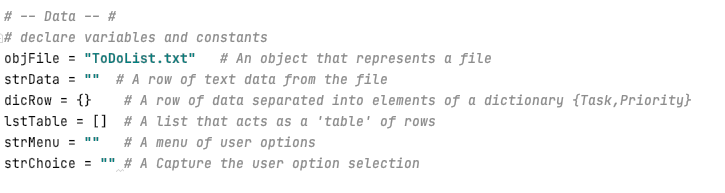
The program will be organized according to a “Separation of Concerns” pattern to help group sections of code according to the types of operations they perform.

Finally the project files will be uploaded to a GitHub repository to make them available to other users, but still maintain control and history of revisions.

# Designing the program

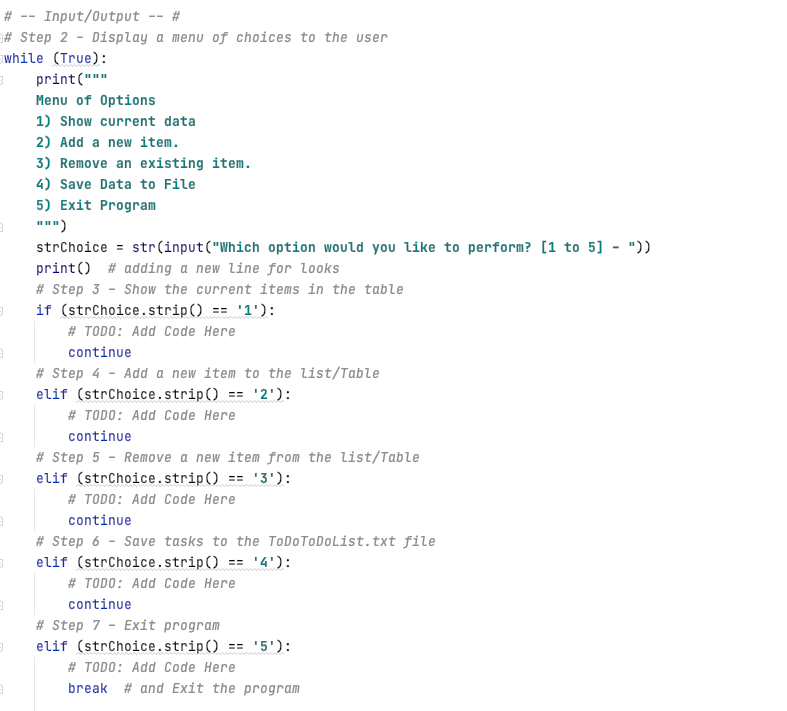
For this project, a starter file was provided that lays out the basic structure and some of beginning code for portions of the program. The program will be divided into three sections for data, processing and input/output.

A set of variable definitions was provided for the data section, as seen here.



***Figure 1: Given variable definitions***

Structure for a menu to inform the users of options, collect their input, and controlling the flow of the program was also included in the starter file.



***Figure 2: Given code for menu structure***

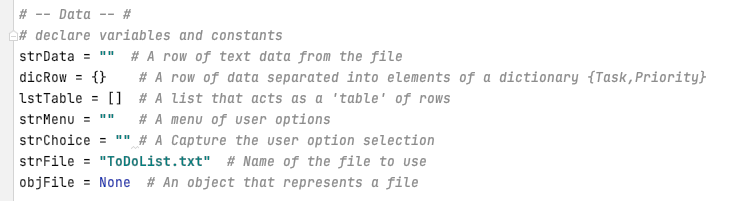
The processing section of the starter provided only comments, and no executable code - so it is not included here.

The logic for the program in this project can be shown as:

* Data section
  + Define variables and constants (given with starter)
* Processing section
  + Read the file
    - Open the file in read mode
    - Loop through rows in data file
      * Read data row from file as list
      * Save list data to dictionary
      * Add dictionary to table list
    - When end of rows, close file
* Input/Output section
  + Display the menu (given with starter)
  + Get user option
  + If option == ‘1’
    - Loop through dictionaries in list
      * Print values to screen
  + If option == ‘2’
    - Get user inputs as dictionary
      * Add new dictionary to table list
  + If option == ‘3’
    - Get user input for item to remove
    - Remove item from list table list
  + If option == ‘4’
    - Open the file in write mode
    - Loop through dictionaries in table list
      * Extract values, concatenate as string, write to text file
    - Close text file
  + If option == ‘5’
    - Exit program

## Data

Looking at the original code, we can see a variable objFile was created to represent the file, but was given a string value. So the first action will be replacing the objFile with a generic object rather than the original string value, and creating a separate string variable for the filename.



***Figure 3: Updated variable definitions***

No additional changes to this section are needed at this time.

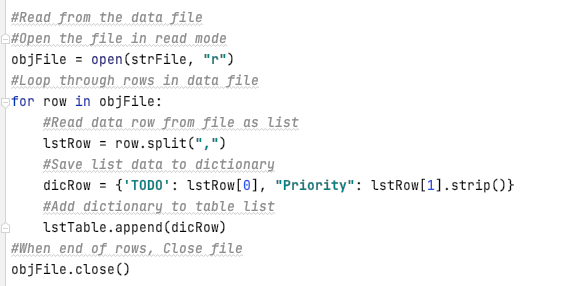
## Processing

In this section the only thing we will do is read existing data from a text file.

The file is opened in read mode and assigned to objFile - which is then used by a for loop to read the data into memory. Since the data was originally written to the file as strings, the split()method is used to divide them into lists. Because the original data use a comma to separate the discrete values we have given this as the delimiter on which to separate the data (*Python String split() Method,* (n.d.). Retrieved November 13, 2022, from <https://www.w3schools.com/python/ref_string_split.asp>).

With the data now in a list, we can use index values to extract the individual elements and assign them to a dictionary with the appropriate key values that can be used to address them later. (*Python - Dictionary* (n.d.). Retrieved November 13, 2022, from <https://www.tutorialspoint.com/python/python_dictionary.htm>) (External site).

The individual dictionaries are then added to a list - allowing all the data to be passed from one section of code to another with a single reference.

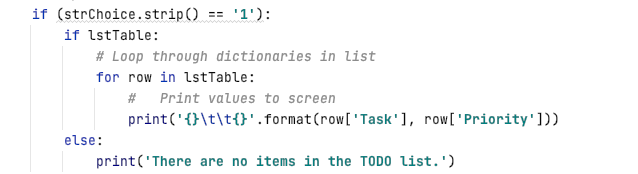


***Figure 4: Code for reading the data***

## Input/Output

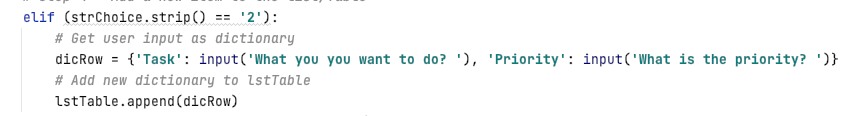
After the existing data has been read into memory - the program will then execute the code containing the menu, which also contains most of the processing code in addition to that required for managing the user interactions.

To display the data we will use a for loop to iterate through lstTable and print out the contents of each dictionary within it. As the program will include an option to remove items from the list, it’s possible that a user could remove all the items - which would then cause the program to throw an exception if they chose to display the existing data. Therefore, the code includes an if…else statement to first check if there is any data to display, if not it will provide a message to alert the user instead of crashing.



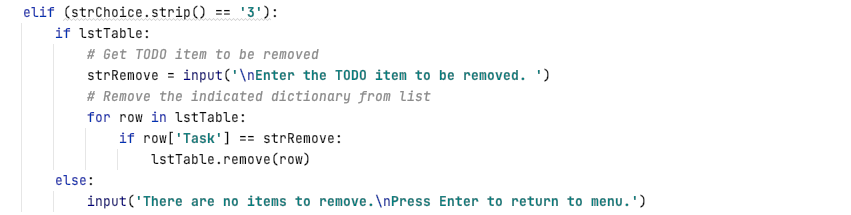
***Figure 5: Code to display the data***

The next option, when selected, prompts the user for information and feeds that input directly into a dictionary, then adds it to the list. It could have used a list to collect the data, similar to the code reading the file - but this eliminates the need for a conversion.

******

***Figure 6: Code to get new entries***

The remove option - like the display option - will only work if there is data in lstTable. Therefore it contains a comparable if…else statement to verify the data exists and notify the user if not.

******

***Figure 7: Code to remove entries***

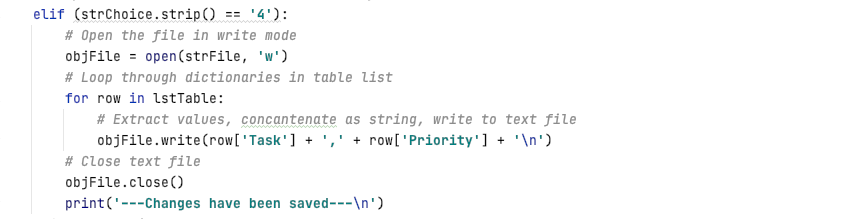
With the data in a list, we could address the value to remove using the index - if we knew what it was. Since the dataset doesn’t have provisions to map the dictionaries to their indices, we need to use a different approach.

Instead, after the user gives their input, the program loops through the list checking the value of Task against the user input. When a match is found, the list item is then passed to the remove() function (devanshigupta1304 (2022, Nov 9). *How to remove and item from the list in Python?* <https://www.geeksforgeeks.org/how-to-remove-an-item-from-the-list-in-python/>) (External site.), and eliminated from the list in memory.

When the save option is selected we will open the file in write mode, which will cause the existing file to be overwritten. Since all the data in the file was read into memory, nothing should be lost.

If the file was opened in append mode, the user could still add new items, but the items they removed would not go away and be reloaded the next time the program is run - and new copies of the existing items would be added to the list each time the user saves.

With the text file open, a for loop causes the data to be written to the file line by line - then closes the file when the values from the last dictionary are extracted, converted to a string, and saved to the text file.

******

***Figure 8: Code to save data***

Since there is already a separate option to save the data, we’ll leave the final selection as is and let the program exit and close the console without an additional message this time.

***pasted-image.png***

***Figure 9: Code to exit program***

Excluding the variable definitions in the Data section, the code now looks like this.

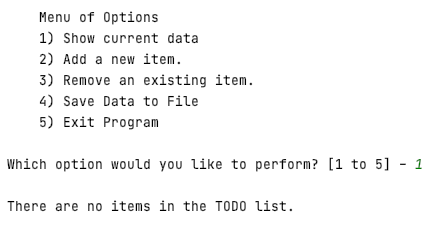
# *pasted-image.png*

***Figure 10: Processing and Input/Output***

# Running the program

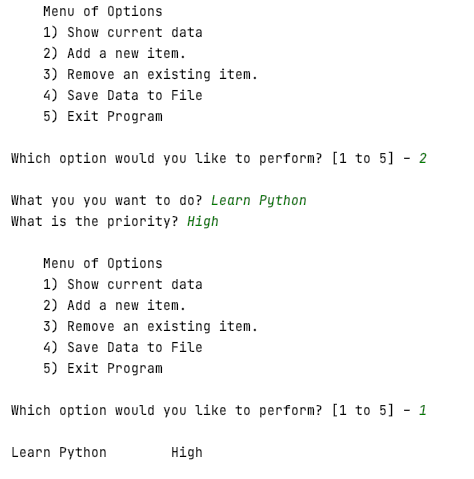
### Executing the program in Pycharm:

Though it was not shown here, a blank text file was created in the working directory to give the program something to attempt to load data from without throwing an exception. Since the file is blank, when we try to display the contents we get the message that there is no data in the list to display.

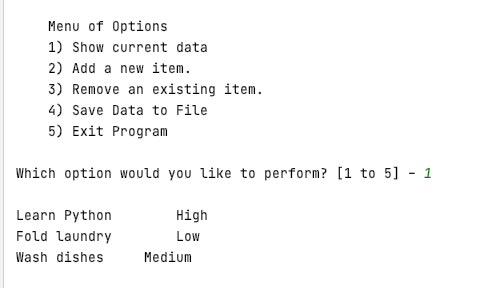


***Figure 11: Display data with empty list***

If we add an item, then try the display again, we get the following:

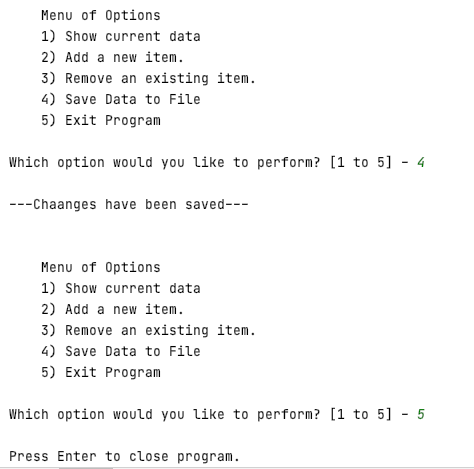


***Figure 12: Add item and display***

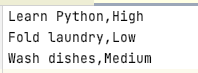
Adding a few more items, we see that the add and display operations appear to be working.

***Figure 13: Display with multiple items***

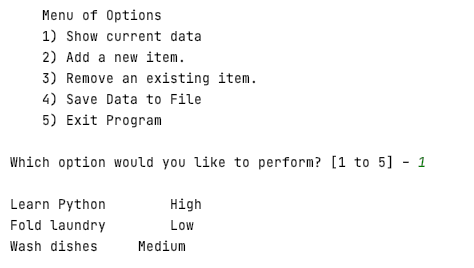
Now that we have some data to work with, we will save the data and exit the program, and check the contents of the text file. Then reopen the program and try the display again to see if the data loads.



***Figure 14: Save and exit***

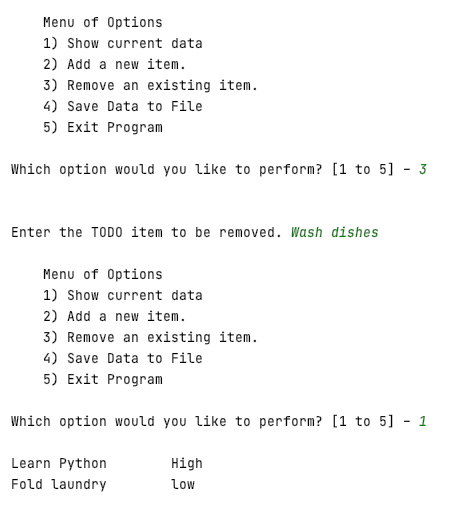
******

***Figure 15: Contents of text file***

******

***Figure 16: Restart and display data***

We see that the data was saved, and the program was able to read the file back into memory. Now we need to test removing an item from the list.

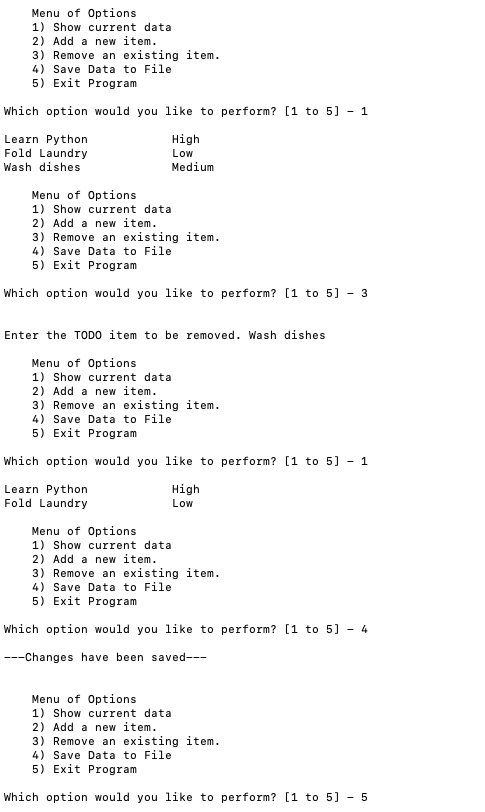
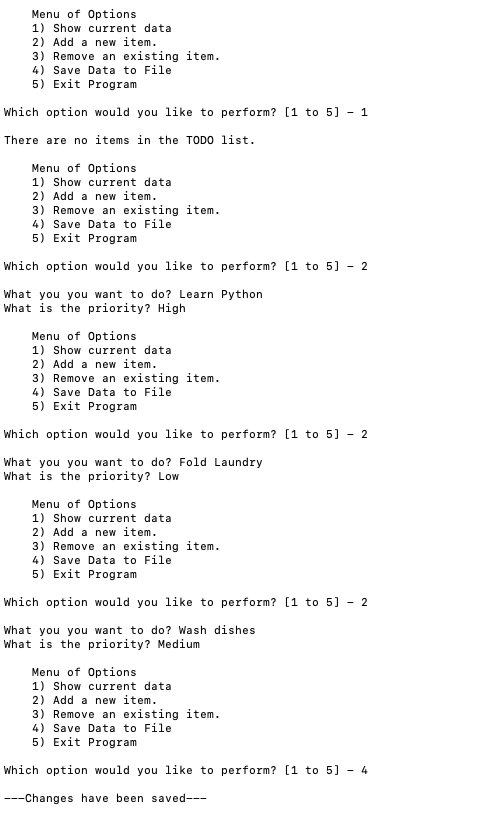


***Figure 17: Remove an item***

The removal appears to work, however the way the program is currently constructed - the user must first select to display the list first, to see if what they wanted to delete is in the list. A future upgrade may be to automatically display the existing list when the remove option is selected.

### Executing in a Terminal window

As the program appears to be running correctly in Pycharm - we try it in a terminal window, and see similar results.

******

***Figure 18: Running in a Terminal window***

# Post to GitHub

GitHub is a hosting site using version control software original developed by Linus Torvalds for his Linux kernel (McQuiad, Mike. *Git in Practice*. Manning, 2015). Among other things is provides a means to securely store their code off-site, make it available to share with others as needed, and keeping a history of revisions so programmers can compare with or retrieved archived versions of their code.

<https://github.com/>

# Summary

In this project we created a program to interact with text file - reading data into memory, modifying it, then writing back to the text file for later use. As the data loaded, it was converted into a set of dictionaries to allow the data to be addressed by keys identifying the elements by name rather than position within the collection.

Although the use of dictionaries instead of lists may not provide a noticeable difference to the users - within the coding environment, the programmer can make use of a dictionary without knowing it’s detailed structure. Knowing the name of the keys within a given dictionary is enough information to be able to interact with a given element regardless of location. By using meaningful descriptors for keys, the intent of data elements become more clear - making the code easier to understand and maintain.