Alexander Bryant

5/17/2020

Foundations of Programming

Assignment 05

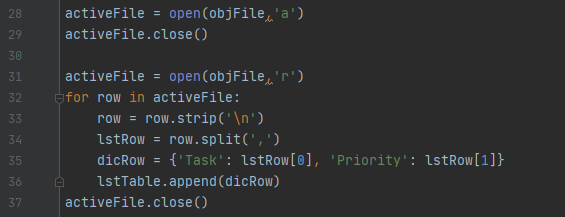
The Python ToDo List

Intro

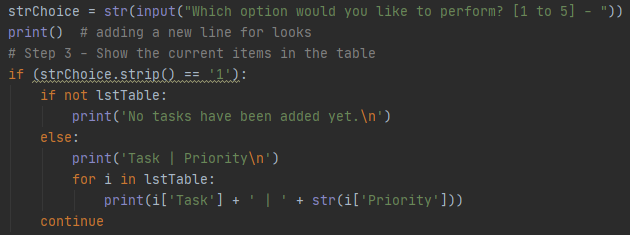
This week’s module focused on additional collections within Python, particularly lists and dictionaries. This week’s assignment was to finish a code starter to create a Python ToDo list, with an emphasis on applying lists and dictionaries to solve these problems. The largest personal challenge within this assignment was to use a code starter instead of coding from scratch, as it required both interpreting someone else’s existing code in addition to writing new code.

Process

Similar to the previous Home Inventory programs, the ToDo list program should solicit user tasks and associated priority levels, while also allowing the user to view, remove, and save tasks. The code starter included several existing variables already used within the code. In ***Figure 1*** below, the processing begins with an open and append statement. This searches for the ToDoList.txt file, and creates it if it doesn’t yet exist. If the file does already exist, the following statement reads the file’s content and uses a for loop to organize the contents into dictionary rows which are then fed into a larger table, lstTable. This makes the file’s contents accessible while the program is running. The file is then closed so that it may be accessed as needed later.

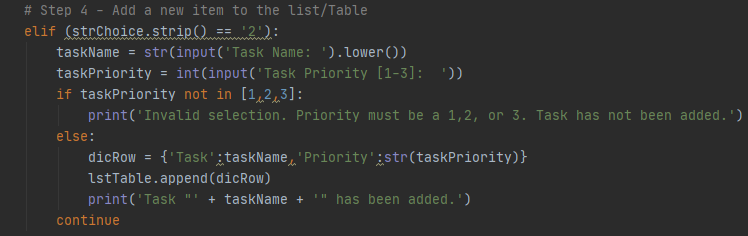


***Figure 1: The processing begins by creating a new file or opening an existing file.***

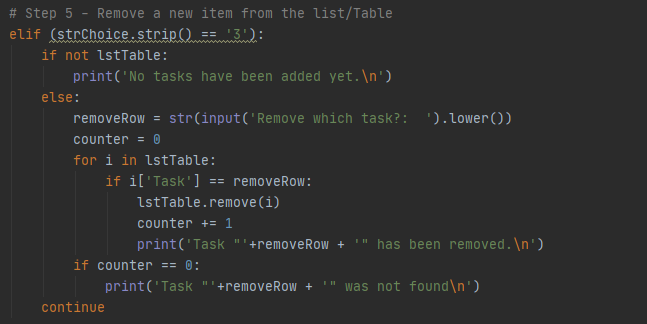
******

***Figure 2: Menu options rely on conditional statements to function properly.***

After the file is loaded into memory, the user is presented with a menu. Visible in ***Figure 2***above, the menu prompt relies on an extended if statement to interpret the user’s choices. In menu option 1, the user is able to review any existing tasks and their priority level. The code checks if lstTable is empty and if so, confirms for the user that no tasks have been added. This will run if no tasks have been added and the ToDoList.txt file is empty when the program initializes. If lstTable is not empty, a for loop prints out the task list by interpreting their dictionary keys, ‘Task’ and ‘Priority’. In menu option 2, visible in ***Figure 3*** below, the user can add new tasks and their priorities. A user prompt requests a lower case string for the task name, and then an integer value for the priority level. The prompt specifies that the input should be between 1 and 3. An if statement enforces this by checking the variable taskPriority against a list with a not in statement. If the priority level is not an integer between 1 and 3, the code reverts to the menu without saving the task. If the priority is an acceptable value, the task name and priority are saved as a dictionary row, which is then appended to the existing lstTable list. A confirmation prints out that this task has been successfully added to the list.

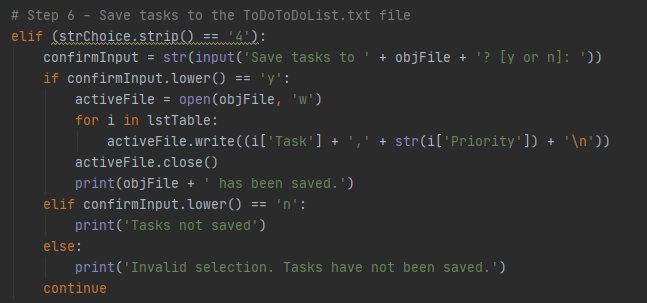


***Figure 3: The user can add new tasks, but conditions exist for the task name and priority level.***



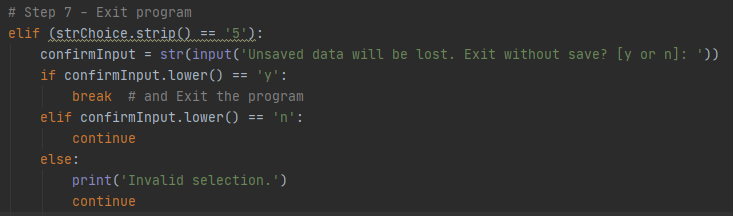
***Figure 4: The user can remove existing tasks from the to do list.***

In menu option 3, visible above in ***Figure 4***, the user can remove existing tasks. Similar to menu option 1, the code first checks to make sure that lstTable is not empty. If it is empty, a confirmation prints to let the user know that there are no tasks available for removal. If lstTable is not empty, a user prompt requests a lower case string task name which is saved as removeRow. Given that dictionaries were the focus of this week’s module, the assignment specified that tasks and their priority level should be saved as a dictionary, within a list. Given that dictionaries can be searched by keys but not values, a counter and a for loop is used to evaluate whether or not the user has submitted a valid task name. The counter is set to zero when the user chooses menu option 3 and submits a task name. If lstTable exists and is not empty, the code iterates through lstTable until a row’s ‘Task’ key matches the submitted task name, removeRow. Once the matching row is identified, the row is removed, the counter is increased by one, and a confirmation prints that the specified task has been removed. If the for loop reaches the end of lstTable and the counter is still equal to zero, this means that there was no matching key value pair for the submitted task, and a confirmation prints out that there was no task found with that name. A counter was used as I prefer to confirm for users when values don’t exist, but confirming within the for loop would mean that for every row that didn’t match, it would print a confirmation, which would be an undesirable outcome and likely unpleasant for the user.



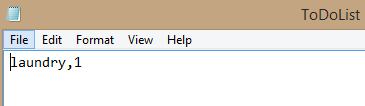
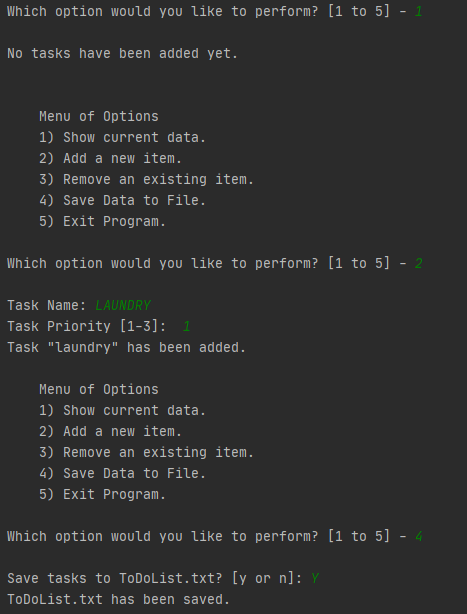
***Figure 5: The user must save before exiting to write their tasks to the ToDoList.txt.***

In menu option 4, visible in ***Figure 5*** above, the user can save the tasks that have been added to the lstTable to the ToDoList.txt. The code requests a ‘y or n’ confirmation just in case the user is not ready to save or has changed their mind. If the user replies ‘y’, the file is opened for overwriting and then a for loop writes the lstTable contents as a string to the text file. Because the text was read and the contents were copied to lstTable at the program’s initialization, overwriting prevents duplicate items from being readded from lstTable back to the text file. If the file was appended instead of overwritten, the resulting text file could contain duplicate tasks. After overwriting, the file is closed and a confirmation is printed for the user. If the user selects ‘n’ or chooses an invalid entry, the tasks are not saved and a confirmation is printed for the user. Finally, in menu option 5, visible below in ***Figure 6***, the user can exit the program. Because saving and exiting the program are two different menu option, the user is reminded that unsaved tasks will not be written to the text file. This gives the user an opportunity to go back and save their work before exiting. If the user enters ‘y’, the program’s while loop ends and the program exits. If the user enters ‘n’ or an invalid selection, the program continues and the user is returned to the menu.



***Figure 6: The user is reminded upon attempted exit that unsaved tasks will be lost.***

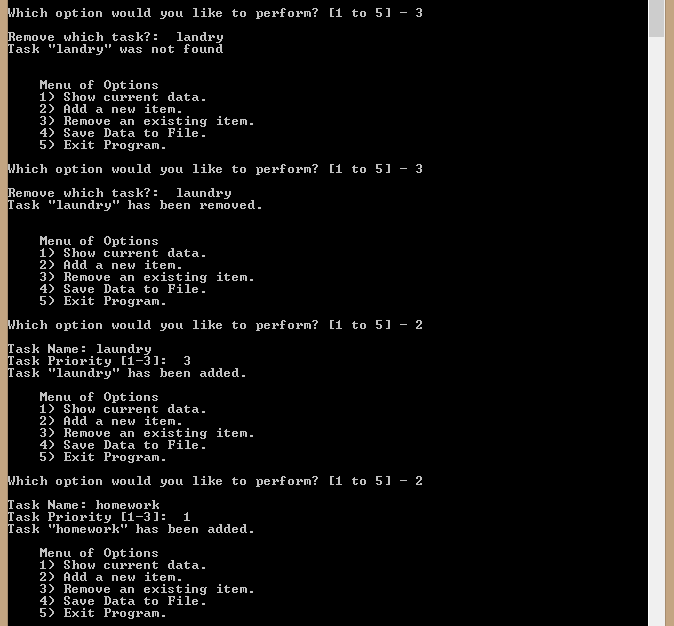
Demonstration



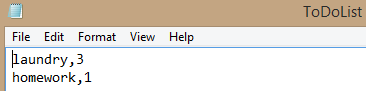
***Figure 7 (left) : A PyCharm user checks the task list, adds laundry, and then saves the tasks.***

***Figure 8 (right): The resulting text file contains only laundry and it’s priority level.***

In ***Figure 7*** and ***Figure 8*** above, a user in PyCharm runs the program and first checks to see if there are any existing tasks using menu option 1. As this is the first time the program has run, the text file was created upon initialization and is currently empty. Using menu option 2, the user adds laundry as priority level 1, and then saves the tasks to the text file using menu option 4. Later, in ***FIgure 9*** and ***Figure 10*** below, a user uses the command prompt to make some edits. After realizing homework is more important than laundry, the user attempts to remove laundry but makes a spelling error, as there is no task named ‘landry.’ Using option 3 again, laundry is removed, then re-added at priority level 3 using option 2, and then the user adds homework as priority level 1 before saving and exiting.

******

***Figure 9: A command line user edits the task list after re-evaluating priorities.***

******

***Figure 10: The user’s new ToDoList.txt after reconsidering the importance of laundry.***

Summary

Assignment05 built on many of the concepts presented earlier in the course, but was particularly challenging for me due to using a starter code instead of writing from the beginning. Working with code written by other people can be challenging but is very common in the tech industry. While lists, loops, and inputs were simple, providing confirmations to anticipate user actions and concerns proved challenging for me due to using dictionary collections. However, receiving no feedback from a program can be frustrating for a user, so I added confirmations and explanations wherever possible to improve the user experience.