Aaron Lanphear

February 14 2021

IT FDN 110 B Wi 21: Foundations of Programming: Python

Assignment 5

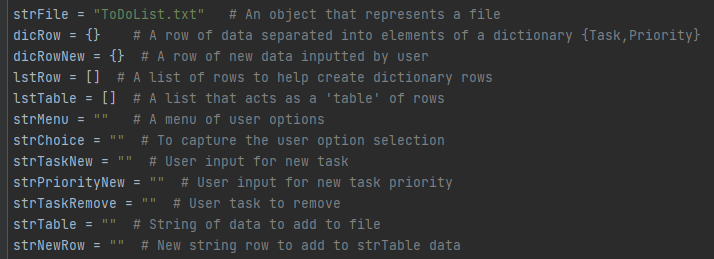
# To-Do List

## Introduction

This week’s script introduces dictionaries and the ability to read data from a file. Rather than create a file and the data from scratch with the program, we open an existing file to read and manipulate the data, and to save any additions or changes.

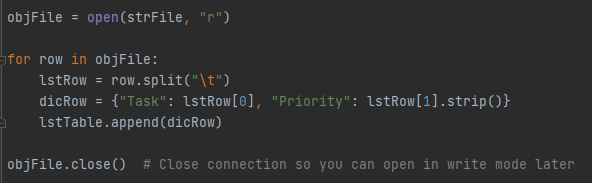
## The Start of the Script

This week we started off the script by declaring all of our variables (Figure 1). While not a requirement with python, it *can* be for other programming languages, so this was a good way for us to try it out. It can also make reading code, or editing it, much easier down the road. Especially for another programmer. I organized mine by type.



***Figure 1: Declaring the script’s variables***

The first block of allows to read the file we are working with by opening it in read mode. It grabs each row and places them in a list, which is then used to organize each row into their own dictionaries. Dictionaries are similar to lists, but use keys instead of indexes (i.e., names instead of numbers) to organize the data, and can allow us to create unique rows. From there, we add each of these dictionaries to a new list, which acts as a table for our rows (Figure 2). This table will be used within our main section of code to work with the data.

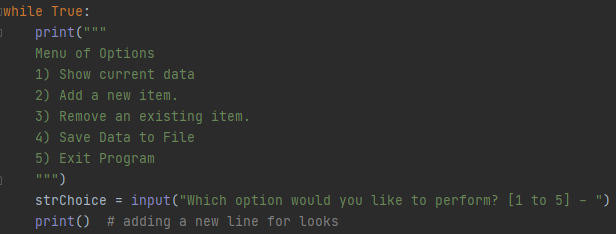


***Figure 2: Opening the file in read mode and extracting the data. Closing the connection here prevents an error message when we open the file in write mode later.***

## The Main Code

### While Loop

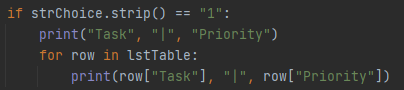
The nuts and bolts of the program are stored within a while loop (Figure 3). This keeps the program active until the user explicitly asks it to exit. What the user inputs here will tell the program which action to perform. This is controlled with a series of conditional if and elif statements.



***Figure 3: The beginning of the while loop, including the menu of options and user input***

### Option 1

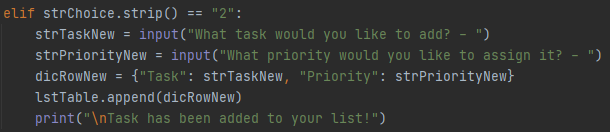
Choosing option 1 will display the current list of data back to the user (Figure 4). This originates in our first block of code that reads the data from the file, but can also include data we’ve added to it during our use of the program. The code here is relatively simple. It just iterates through the rows in our list table and prints them in a prettier format.



***Figure 4: Option 1, which prints out data to the user***

### Option 2

Option 2 will allow the user to add a new task to the existing data list (Figure 5). It will first ask them to provide the task name, which is then followed by the priority the user wants to assign to it. Each of these inputs are placed in a new dictionary row. This row is then added to our existing list table. A user-friendly response is printed to let them know it was added successfully.

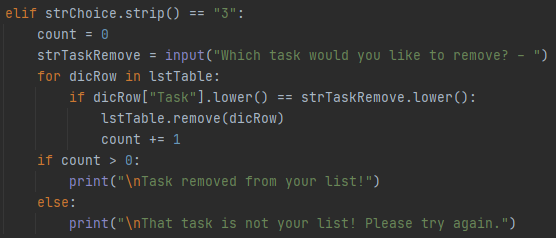


***Figure 5: Option 2, which adds user-provided data to our list table***

### Option 3

To remove a task, the user will choose option 3 (Figure 6). The program will allow the user to input the task they want removed. As it iterates through the rows in the list table, it will remove any instance of a match and the let the user know. It’s ability to match is based on the key value in each dictionary. If the “Task” key value matches what the user inputted, then the item is removed. Using the *lower* method allows the program to find matches even when the inputted data or list data are entered differently (i.e., capitalized vs. lower case).

I’ve also added a counter in the code to catch instances where a task that doesn’t exist in the list is entered by the user. This works by starting with the count equal to 0. If a match is found and removed, the code will add 1 to the count. After the iteration of rows is complete, the code will check to see if the count is greater than 0. If it is, we know that a match was found and removed. If it is still 0, then we know the task didn’t exist.

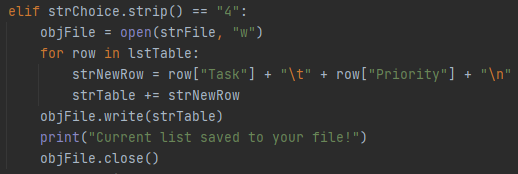


***Figure 6: Option 3 allows the user to remove a task from the list. If one doesn’t exist with that name a specific message is printed.***

### Option 4

If the user wants to save the current list of data to the file, they would choose option 4 (Figure 7). This code reopens the original file, but this time in write mode. Write mode will overwrite all data currently in the file, which is fine in this case because we are writing the list extracted *from* the file itself, plus/minus any changes.

The code creates a new string row for each dictionary row, separated by a tab and closed with a new line. It then adds each of these new string rows to a string “table”, which is then used to write to the file. This allows us to control how we want to data to appear in the file. After a message is printed to the user, the connection to the file is closed.



***Figure 7: Option 4 allows the user to save the current list of data to the file***

### Option 5

Finally, option 5 allows the user to exit the program. The code here couldn’t be any simpler. The break statement simply breaks the program out of the while loop. Because there is no more code outside of the loop, the program will finish and close.



***Figure 8: Option 5 to exit the program.***

### No Option

I also included an else statement at the end of the while loop to catch any instance of a user entering an option other than 1 – 5 (Figure 9). If they do, the program will let them know and will restart the loop.



***Figure 9: Statement to catch a user entering an invalid menu option***

## Summary

This week we learned how to work with dictionaries and how to read data from a file. This assignment displayed how we can get data from a file, manipulate it, add or remove data, and write it back to the file. We learned that dictionaries are a great way to store rows of data, and how combining them into a list can give us a very useful table.