Jessica Carnes

November 16, 2020

Foundations of Programming: Python

Assignment05

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

Creating a To Do List in Python

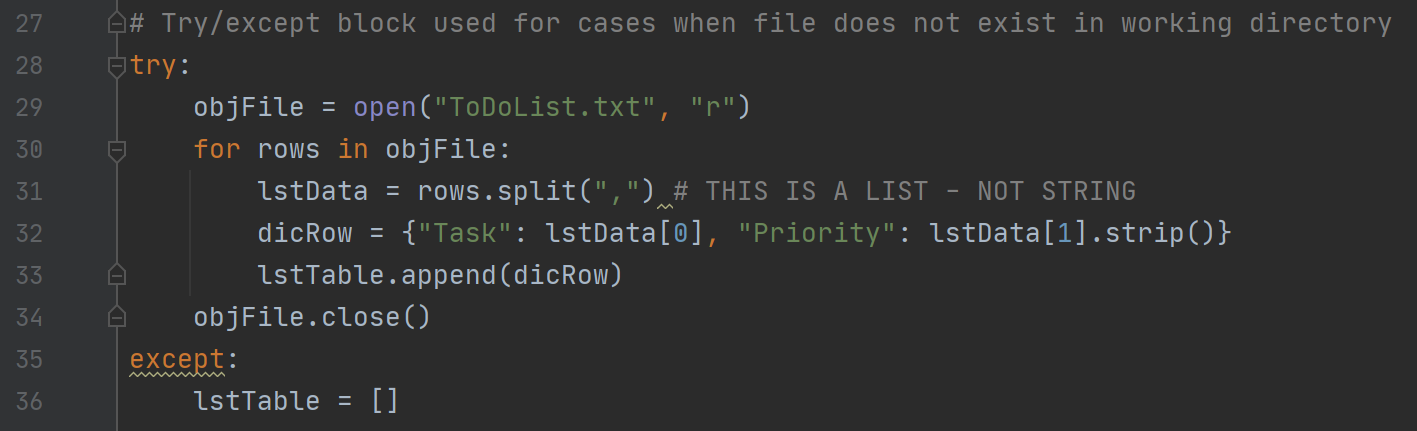
# Introduction

This paper will discuss how to create a To Do List program in python. This program will load data from an existing ToDoList.txt file then provide the user a menu of options to choose from to modify the data in the to do list. The program will load the contents of the file into a list of dictionaries, then allow the user to modify this list by selecting options from a menu prompt. The menu will have the options of displaying the current data in the list, adding items to the list, removing items from the list, saving the list back to the text file, or exiting the program. For loops, while loops, lists, and dictionaries will all be utilized to make these options available.

# Loading the Data to a List of Dictionaries

The first step in creating the to do list program is to load any data stored in an existing ToDoList.txt file to a list of dictionaries in Python. A try and except block is used around the loading of the text file to handle errors if the file does exist in the working directory. The list of dictionaries is represented by the variable lstTable in the script. The ToDoList.txt file is opened in read mode then a for loop runs through each row of the file, using the split() method to separate the data on each row into separate elements of a list based on the comma separation in the text file. This row data is stored in the variable lstData. Once in in a list, the rows are converted into a dictionary and stored in the variable dicRow by entering the keys “Task” and “Priority” then calling their respective index of lstData. The variable lstTable is then appended with dicRow and this process repeats for the remainder of the rows in the file. The code for loading the text file to lstTable is shown in lines 29-34 of Figure 1.

Under the except portion of the block, the variable lstTable is set to the empty list (lines 35-36 of Figure 1). This will be the case when the file ToDoList.txt does not exist in the working directory as the file must exist to be opened in read mode.

  
Figure 1: Loading the existing ToDoList.txt file to lstTable

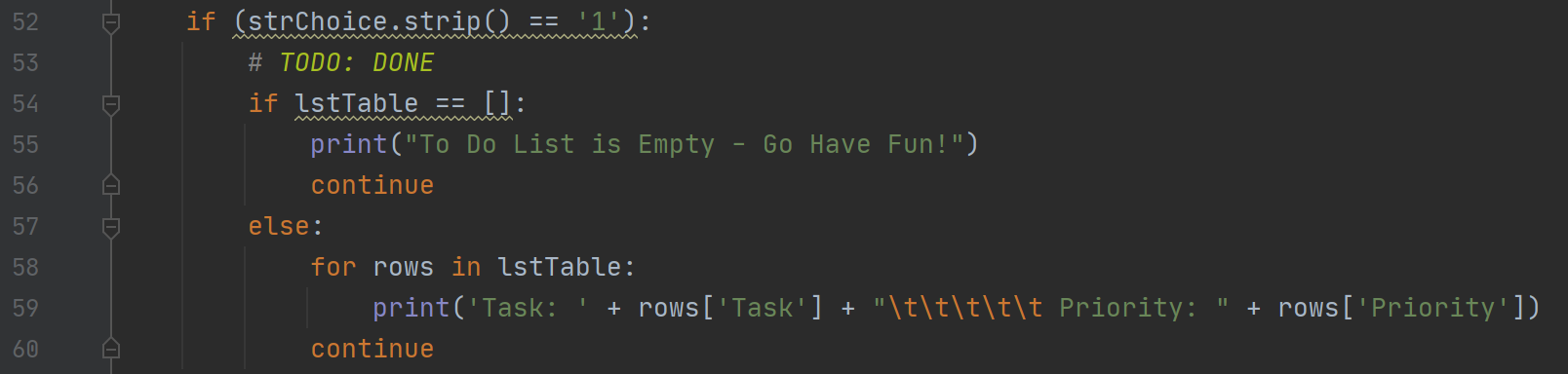
# The Menu Loop

Once the initial data from the ToDoList.txt file is loaded, the program displays a menu of options for the user to select. The menu continues repeating until the user chooses option 5 to break this while loop and exits the program. The following contains the details of the logic behind each of the menu options.

## Option 1: Display Existing Data

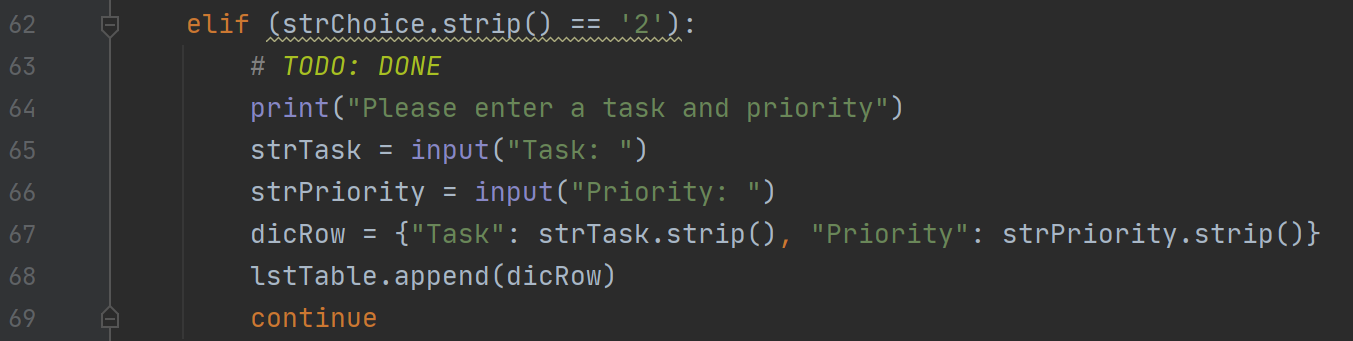
If the user enters 1 into the menu prompt then the program will display the contents of the lstTable variable. If this is the first option the user chooses, then the lstTable variable will contain the contents of the ToDoList.txt file or be an empty list if the file does not exist. Before printing the contents to the screen an if function checks to see if the lstTable is equivalent to an empty list. If that is the case a message is printed to the user stating the list is empty as shown in lines 54-55 of Figure 2.

Otherwise, the script runs through a for loop in lstTable. The program loops through all rows of lstTable and prints the task and priority as shown in lines 58-59 of Figure 2.

  
Figure 2: Displaying the current contents of the list

## Option 2: Adding a New Item to List

When the user enters 2 into the menu, they are prompted to add a new item to their to do list. The input function is assigned to the variables strTask and strPriority to capture the user entry for the task and priority respectively. These inputs are then added to a dictionary under the variable dicRow as shown in line 67 of Figure 3. The dictionary is then added to the lstTable variable using the append method as shown in line 68 of Figure 3.

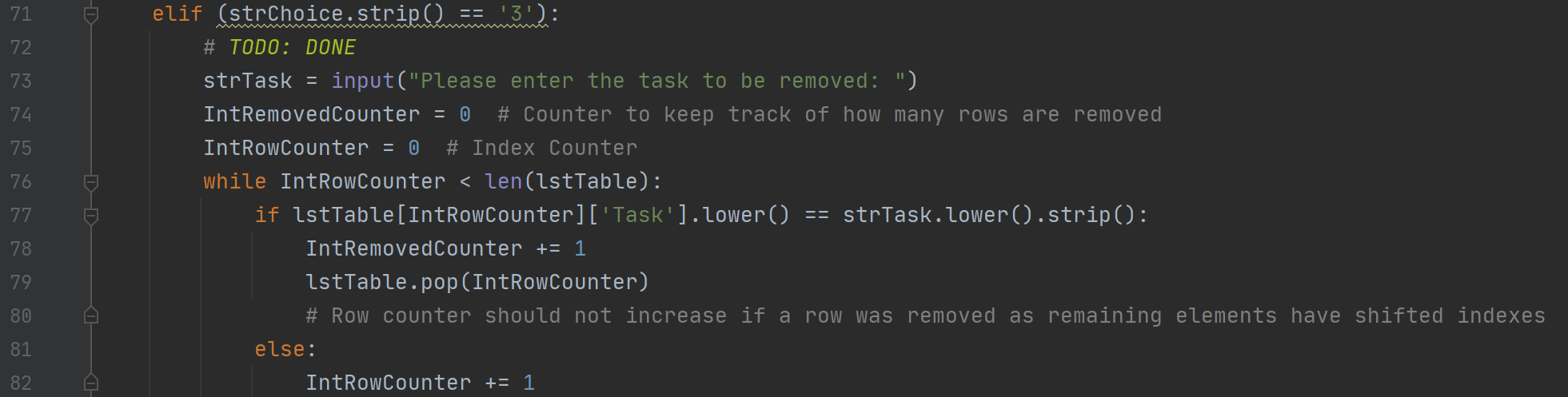
  
Figure 3: Adding a new task stored as a dictionary to the list

## **Option 3: Removing an Existing Item**

### Identifying and removing the item

If the user wants to remove a task from the to do list, then option 3 is entered into the menu prompt. This option prompts the user to enter the task they would like removed which is then stored in the variable strTask. A while loop is used to run through each element of the current list stored in lstTable, using an integer counter variable, IntRowCounter, to access the elements through their index. The individual elements are dictionaries and the value associated with the dictionary key ‘Task’ is retrieved and then compared to the value of strTask. If the value stored in the task key matches strTask then a second integer count called IntRemovedCounter is incrementally increased and the dictionary is removed by using the pop() method. The pop method removes elements of a list based on the index. This process is shown in lines 76-79 of Figure 4.

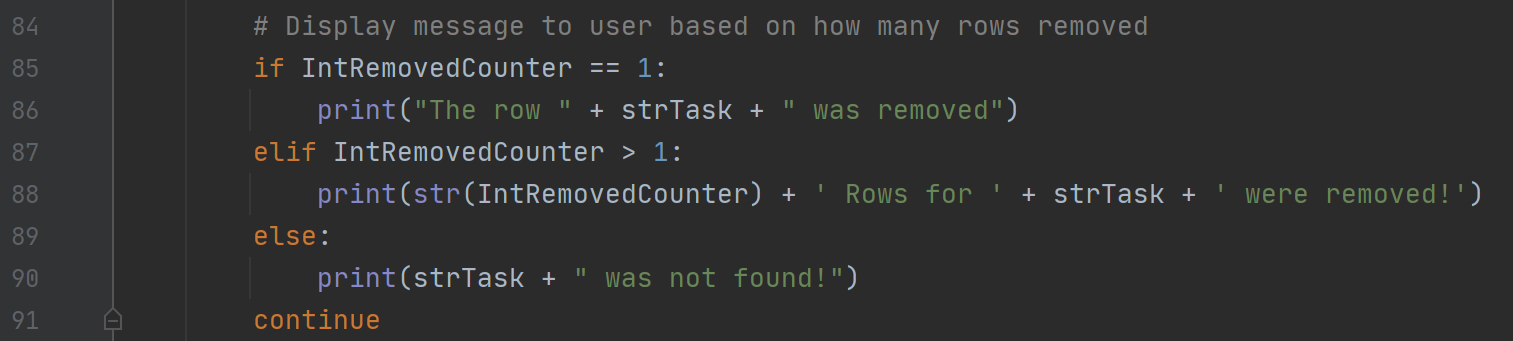
If the value of the task for a particular dictionary element of lstTable does not match the strTask entered by the user then the IntRowCounter is increased, and the loop continues to the next element of lstTable until the IntRowCounter is no longer less than the length of lstTable.

  
Figure 4: Removing an element from the list based on user input

A while loop with the integer counter is used to loop through lstTable instead of a for loop to ensure that if the list contains multiple elements with a task equal to the strTask that all instances are removed. When an element is removed from lstTable the length of lstTable is reduced by one, shifting the index locations of the remaining elements lower by one. The loop does not increase the variable intRowCounter when an element is removed due to this shifting of index locations. Using a for loop would automatically increase the index location even when a row was removed, thereby jumping over the element in the next index location prior to the removal.

### Displaying a removal message to the user

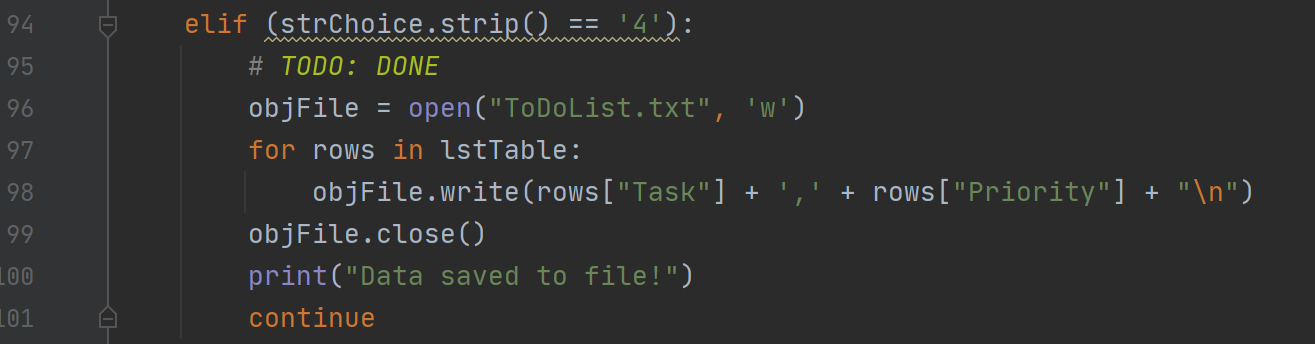
Once the script has identified and removed all elements with a Task matching the user input of strTask a message is printed letting the user know the removal was completed. The variable intRemovedCounter is used to provide more descriptive information to the user regarding the data removal by keeping track of how many elements were removed. An if, elif, and else statement are used to tell the user how many rows were removed, if any at all as shown in Figure 5.

  
Figure 5: Printing a message to the user based on how many elements from the list were removed

## Option 4: Saving the Data to the Text File

When the user chooses option 4, the current data in the lstTable variable is saved to the ToDoList.txt file. The ToDoList.txt file is opened in write mode, therefore any existing data in the file will be overwritten. No data will be unintentionally lost in this process though as the contents of the file were originally loaded to the lstTable variable when the program launched. So all data less the rows that the user removed will be saved back to the ToDoList.txt file, plus any additional tasks added.

To save the data, a for loop is used to run through the rows of the list lstTable. A write command is then used to extract the Task and Priority from each dictionary row as shown in line 98 of Figure 6. The file is then closed and a message is displayed to the user.

  
Figure 6: Saving the data to the ToDoList.txt file

## Option 5: Exiting the Program

To exit out of the menu and the program the user enters 5 into the menu prompt. When this occurs the script first checks to see if the user has saved the data to the text file. This is done by opening the text file in read mode and adding the contents of the file to the variable lstFile by following the exact same code as described earlier in Figure 1.

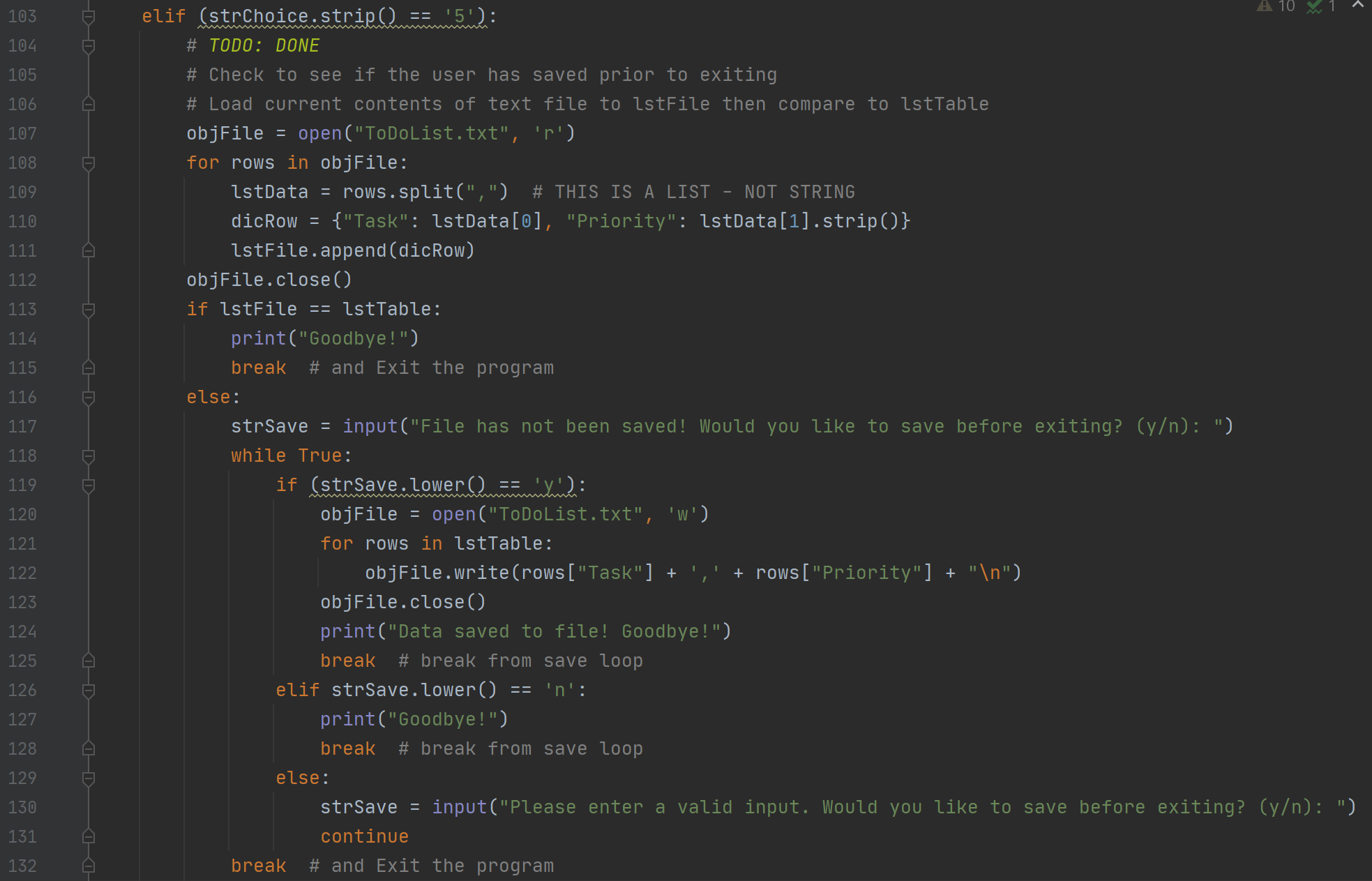
Once lstFile is loaded, it is compared to the lstTable variable. If the two lists are equivalent then the program prints “Goodbye!” and a break is used to exit out of the menu while loop (lines 113-115 of Figure 7).

If the lstFile and lstTable are not equivalent, then that means that lstTable has been modified since the ToDoList.txt file was last saved. If this is the case, then the user is prompted to enter y or n if they would like to save the file before exiting. A while loop is used to loop through the user response. If the user enters ‘y’, then the script runs through the same block of code used to save the lstTable list to the text file as used in menu option 4 above before hitting a break statement to exit the while loop (lines 119-135 of Figure 7).

If the user enters ‘n’ then a message of “goodbye” is displayed to the user and the program exits without saving (lines 126-128 of Figure 7).

Otherwise, the user has entered an invalid response and is prompted again if they would like to save (y/n) then the while loop repeats (lines 129-131 of Figure 7).

Once the user breaks out of the saving prompt while loop the next line of code is a break statement for the menu prompt while loop. This breaks the user out of menu while loop and the program exits.

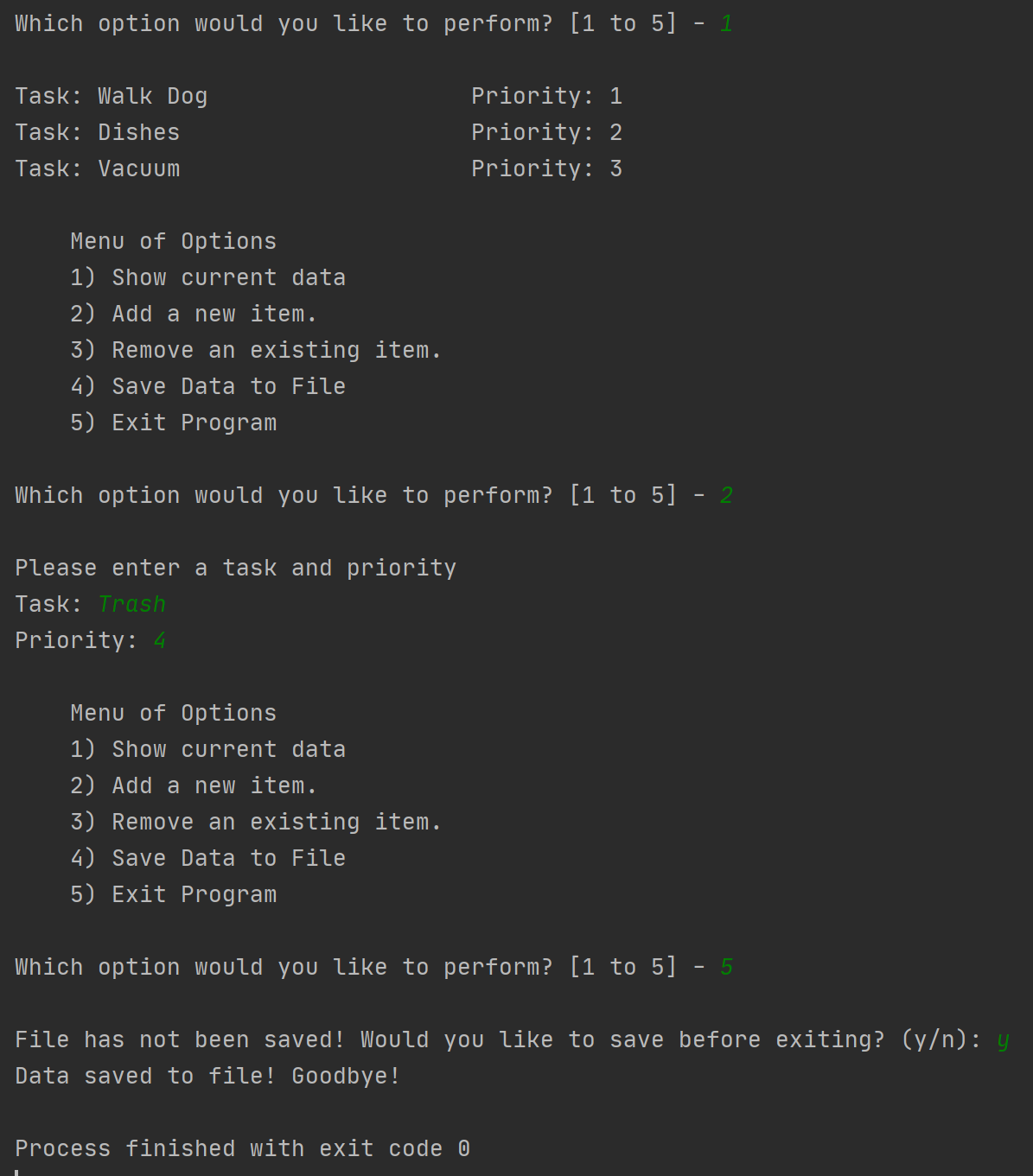
  
Figure 7: Checking to see if the user has made changes before exiting the program

# Running the To Do List Script

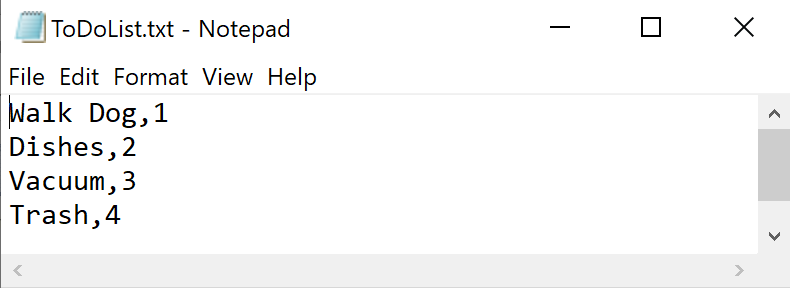
The home inventory script can be ran using PyCharm or from a command window. The following provide examples of successful completions of the script using both methods

## Output using PyCharm

Figure 8 contains a successful run of the script using PyCharm. In this example the initial data in the ToDoList.txt file was displayed by entering menu option 1. There were initially three tasks in the list. Then by selecting menu option 2, a new task, trash, was added to the list. The user then selected menu option 5 to exit, the program noticed the user had not saved the file prior to selecting exit and therefore asked if the user wanted to save. The user entered y, the ToDoList.txt file was saved, and the program exited.

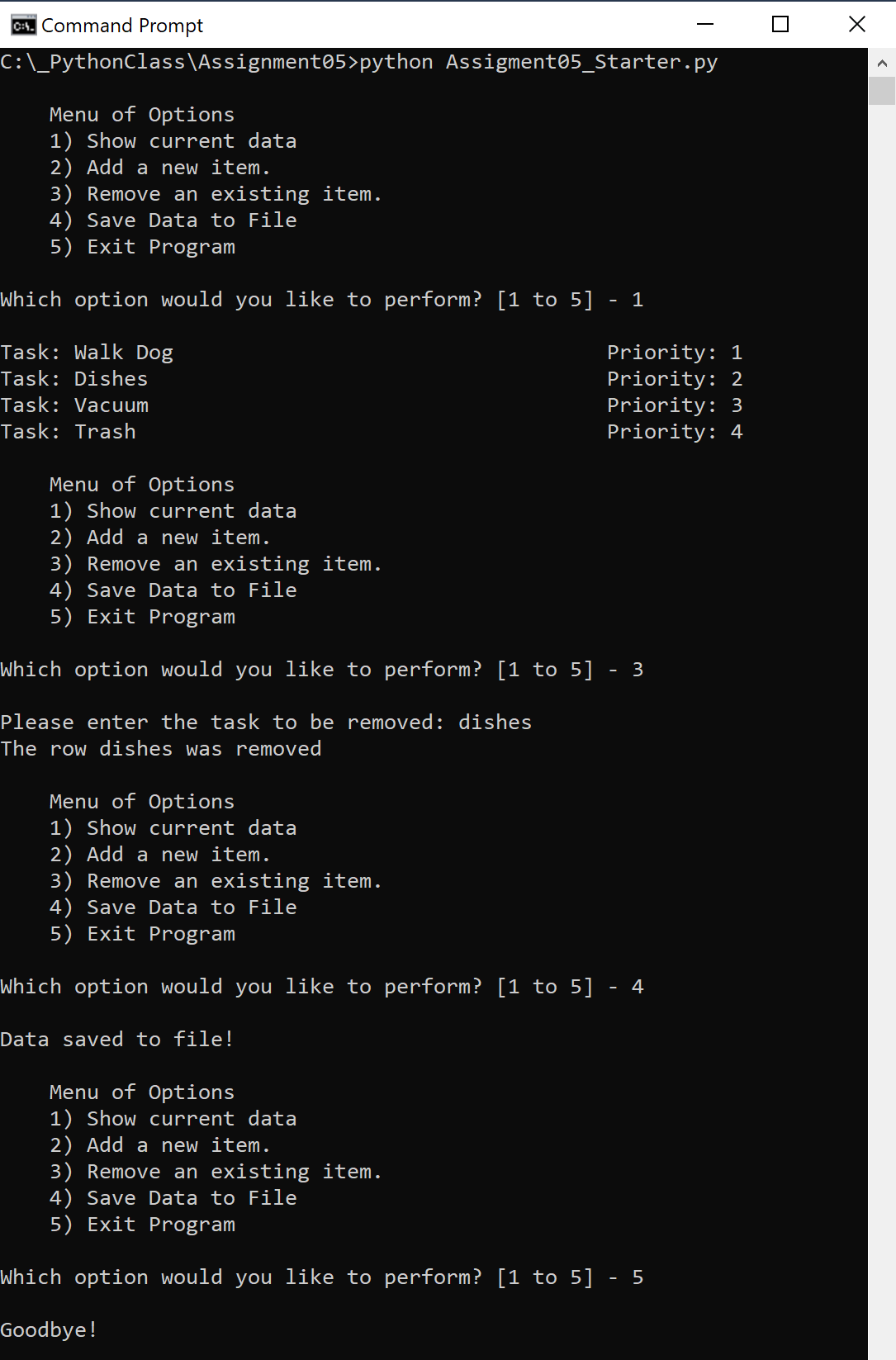
  
Figure 8: A successful run using PyCharm

After the above run in PyCharm the ToDoList.txt text file contains the initial three tasks plus the additional task of Trash added during the running of the script as shown in Figure 9.

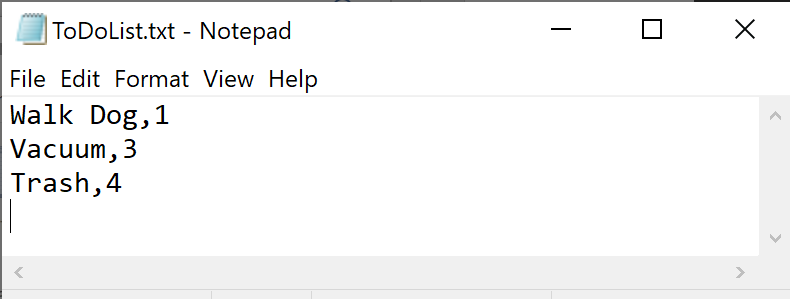
  
Figure 9: ToDoList.txt file after the PyCharm run

## **Output using a command prompt**

The following output as displayed in Figure 10 is an example of a successful run using a command prompt to launch the script. Upon initial load, the user selects menu option1 to display the current data. The current displays the four items in the ToDoList.txt file as shown above in Figure 9. The user then selects to remove a row and enters dishes; dishes was found in the list and removed. The next menu item selected is to save the data to the file, then the user selects exit.

  
Figure 10: A successful run in a command prompt

After running the script in the command prompt the ToDoList.txt file contains three rows as shown in Figure 11. Dishes, which was removed during the running of the script is no longer included in the file.

  
Figure 11: ToDoList.txt file after the run in the command prompt

# Summary

Python can be used to create a basic to do list application. The ToDoList.txt file is initially loaded to list of dictionaries in Python, if the file does not exist then this list begins as an empty list. From there, a series of while loops, for loops, and if statements can be created to run the user through a menu prompt, giving them options to display the list, add to the list or remove tasks from the list. These modifications are stored in memory in the list of dictionaries until the user tells the program to save the list back to the text file. The script can be ran using either PyCharm or a command prompt.