Laura Truong

April 24, 2020

Introduction to Programming

Assignment06

ToDo list using Functions

# Introduction

Last week, students were asked to modify a Python script file that manages a “ToDo list”. This “ToDo” file contains two columns of data “Task” and “Priority” that will be loaded into a Python Dictionary object. The rows of data will be added to a Python list object to create a table of data. This week, the main goal of this assignment is to add functions to organize the code. This paper will outline the steps I took to create the code to add a new item to the list, remove an item from the list, save tasks to the “ToDoList.txt” file, reload data from the file, and exit the program. I will then run the script in a command shell from the Windows OS using Python version 3.

# Process data and user’s menu choice

When the program starts, we will load any data we have in a text file called “ToDoList.txt” into a python list of dictionary rows. Since there is no data yet, I had to create a txt file in my folder location "C:\\_PythonClass\Assignment06\ToDoList.txt". In **figure 1**, we create a function called read\_data\_from\_file that contains the instructions to open a file we will write in. In the txt file, each line has a Task and Priority separated by a comma. The code reads the lines using lstRow and the split() function separating the items in the list based on the commas in each line. dicRow shows the keys “Task” and “Priority” being indexed with the strip() function to remove spaces at the beginning and end of each string. dicRow is then appended to a list table. **Figure 2** shows us activating our “controller.” We call to the class the function is located in “Processor” followed with the function name. In order to create a program that loops depending on the end user’s choice, we use a while loop. Outside of the loop, we provided a status that shows the current data, as shown in **Figure 2** step 2, we call to the function print\_current\_Tasks\_in\_list that contains the information to show the list of rows and adds the new row beneath the current row. Since this current data status is outside of the loop, it will always show the latest update in the rows so that the end user can always refer to it rather than having to click on that particular menu option to see it.

Figure 1

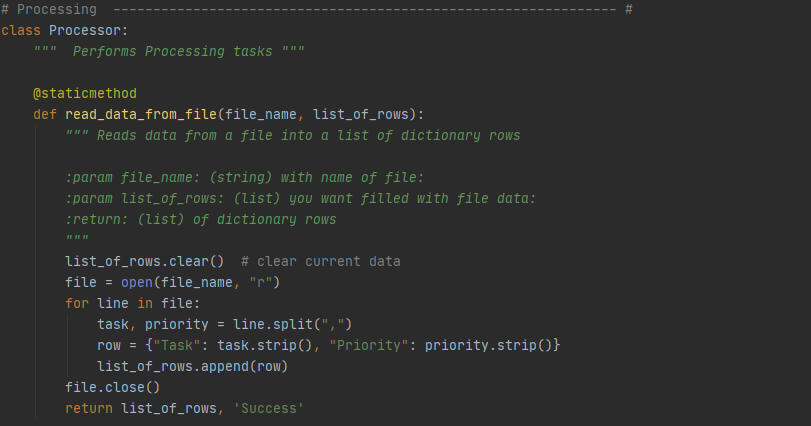
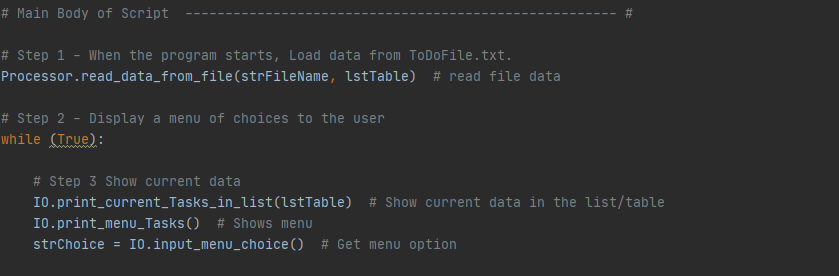


Figure 2



# Adding a new item to the list

To add a new item to the Table lstTable, we ask the user for two inputs, strTask which contains the input for a task, and strPriority, which contains the input for the priority level. strTask is assigned to the function AskUserForTask while strPriority is assigned to the function AskUserForPriority as shown in **Figure 3.** Adding a new item to the list is assigned to the value 1, so if strChoice which is the input of the user equals 2, we will begin this task. **Figure 4** shows the controller calling the functions using the class and function name, Io.AskUserForTask and AskUserForPriority in this scenario. DicRow will then create a dictionary with the the inputs of the user from strTask to “Task” and strPriority to “Priority”. dicRiow is then added to the end of the table lstTable using the append() method.

Figure 3

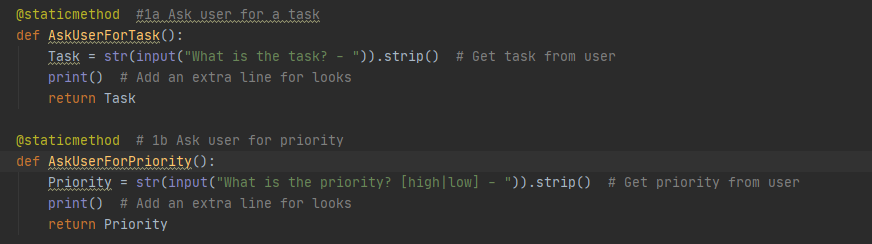
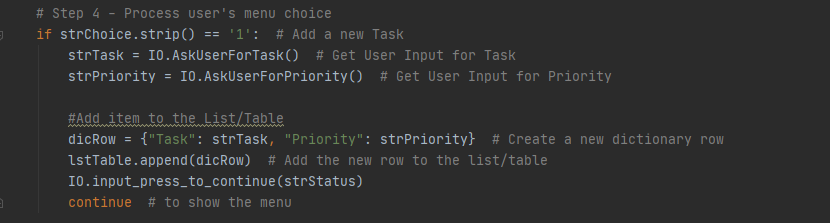


Figure 4



# Removing data from the list

The function input\_task\_remove has a variable called RemoveTask that asks the end user what task they would like removed and returns that value as shown in **Figure5.** Removing an item to the list is assigned to the value 2, so if strChoice which is the input of the user equals 2 we will begin this task. We use the controller in the main body to call the class IO and function input\_task\_to\_remove which asks the user which task they would like removed. **Figure 6** shows the loop used to go through the table to see if the value that the user input typed equals the strKeyToRemove. If it equals the input, the table row will be removed. However, if it is not, a statement will print telling the user that the task could not be found.

Figure 5

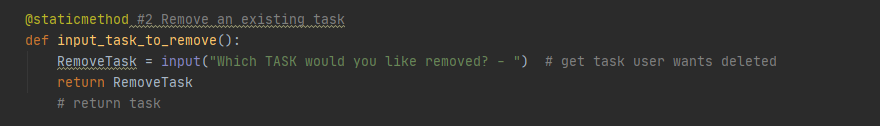
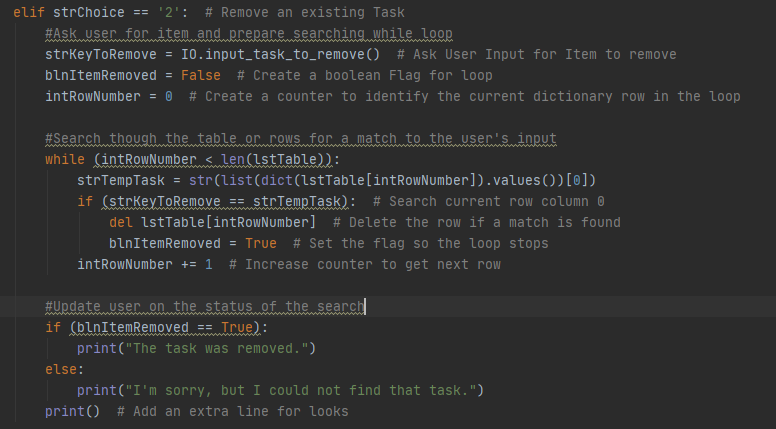


Figure 6

****

# Saving data to .txt file

The function write\_data\_to\_file contains an objfile that opens the file in write mode and goes through each row of data in the file and writes/saves it, It writes the data in the format of Task, Priority in the dictionary as shown in **Figure 7.** After writing to the file, it closes the file as specified by Professor Root as good practice. Saving data to a .txt file is assigned to the value 3, so if strChoice which is the input of the user equals 3, we will begin this task. In the body of the program, we call and pass the values written in the function into the while loop by calling it. We call it using the class Processor.write\_data\_to\_file as seen in **Figure 8.** We also call in the function Io.Input\_yes\_no\_choice which gives the end user the option to hit yes or no if they would like to save the data. If they say yes, the data is written per the instructions in the function. If the end user entered no, they are taken back to the menu page.

Figure 7

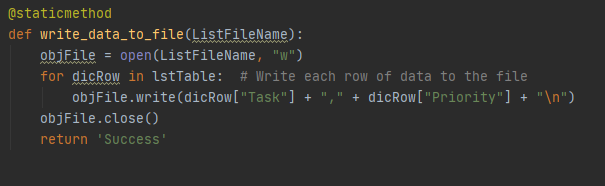
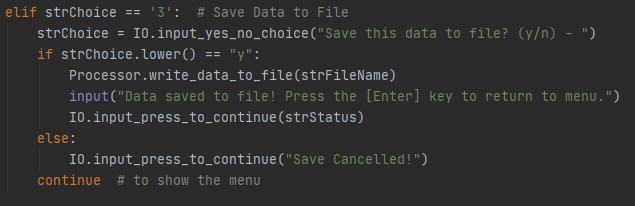


Figure 8

****

# Reload data from file

The function read\_data\_from\_file reads data from a file into a list of dictionary rows as shown in **Figure 9**. It opens the file in read mode and pulls the data in the format of Task, Priority. If new rows are added, they are appended in the table. The string choice with the input of 4 will reload the data from the file. In the main body of the program shown in **Figure 10**, the function input\_yes\_no\_choice and read\_data\_from\_file are both called. The end user is given the option to enter an input asking if they would like to save the data. If they respond yes, the function read\_data\_from\_file is called and the data is loaded. If they say no, they will be taken back to the main menu of options.

Figure 9

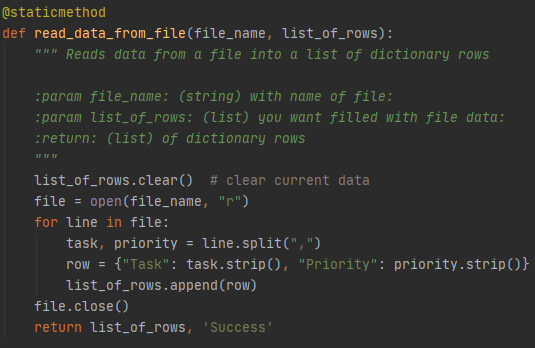
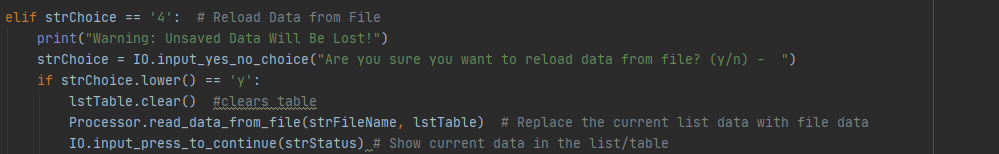


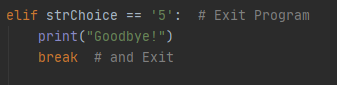
Figure 10



# Exiting the program

Exiting a program is assigned to the value 5, so if strChoice which is the input of the user equals 5, we will begin this task. Exiting a program is fairly simple, I add a break to the end of the program to exit. A break statement terminates the nearest enclosing loop.

Figure 11



# Running the script

Now that I have successfully created my script, I can hit the “run” tab located at the top of the file to execute it. Before running the script, the script will need to be saved. I saved my assignment as “Assignment06.py” to this location "C:\\_PythonClass\Assignment06\Assignment06.py" in the c drive in a folder called \_PythonClass and a subfolder called Assignment06. **Figure 12** shows the result of running the script from Pycharm Community edition 2020.1 version 201.6668.115 in debug mode. I added the task “bb” and later removed it by choosing menu option 2. **Figure 13** shows the data being saved and reloaded. As this processing occurs in the background, the end user just sees the prompts asking them for their input on a yes or no selection.

Figure 12

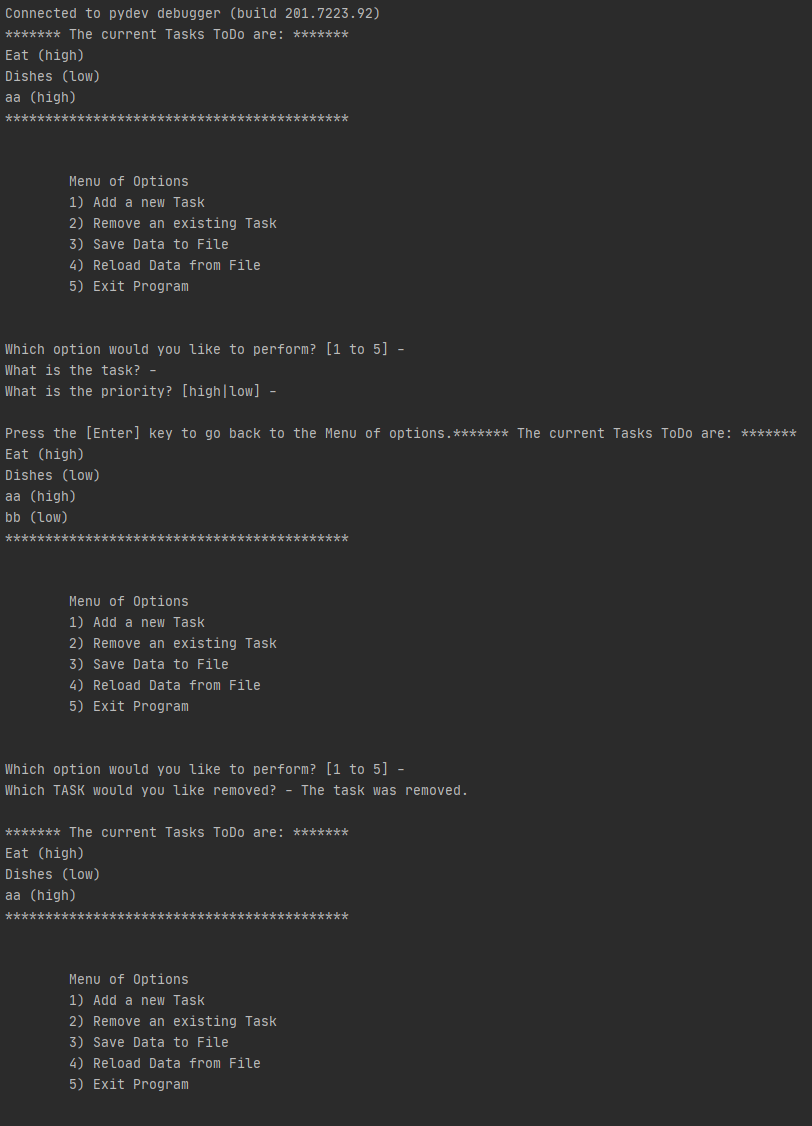
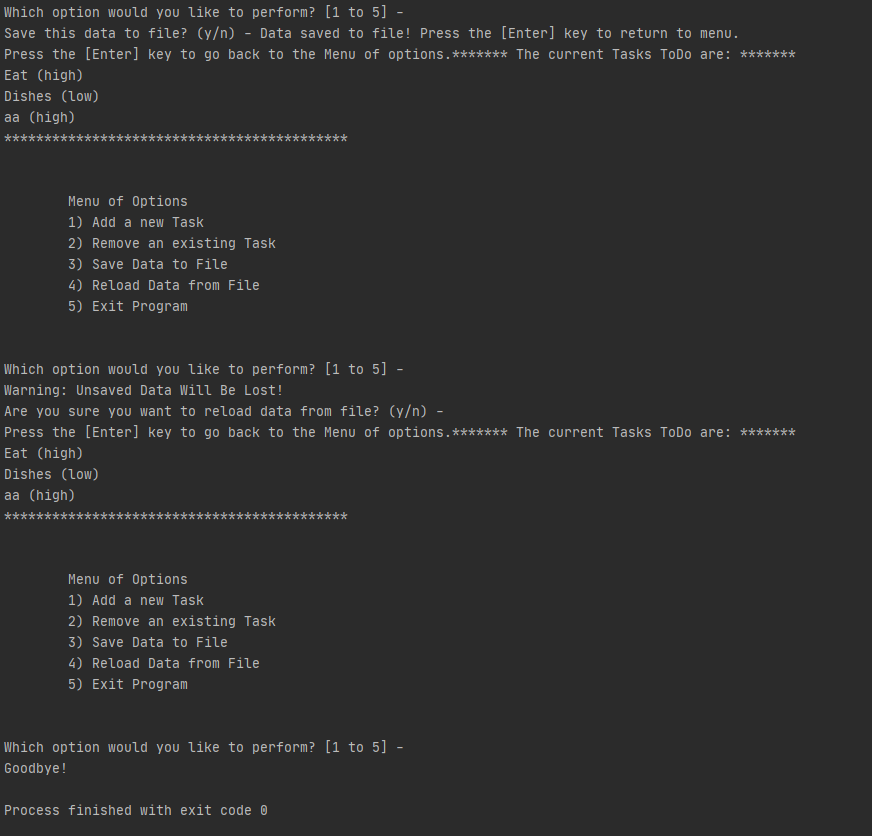


Figure 13



**Figure 14** shows the saved text file in the path the file was opened at: "C:\\\_PythonClass\Assignment06\ToDoList.txt". **Figure 15** shows the data that was written into the text document. The data is formatted to have a Task name and Priority level on each line and start the next Task on a new line.

Figure 14

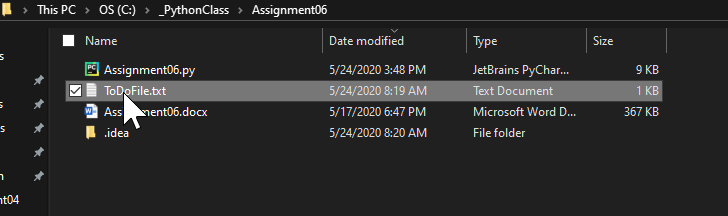
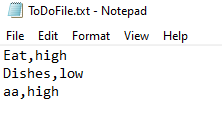


Figure 15



To run the script in an OS command, I opened a command prompt by typing “cmd” into the search bar of my desktop. I typed “Python "C:\\_PythonClass\Assignment06\Assignment06.py" into the command prompt providing the file location to open up the file. **Figure 16** shows the result of my program being run in the cmd line. I ran all 5 items in the menu of options.

Figure 16

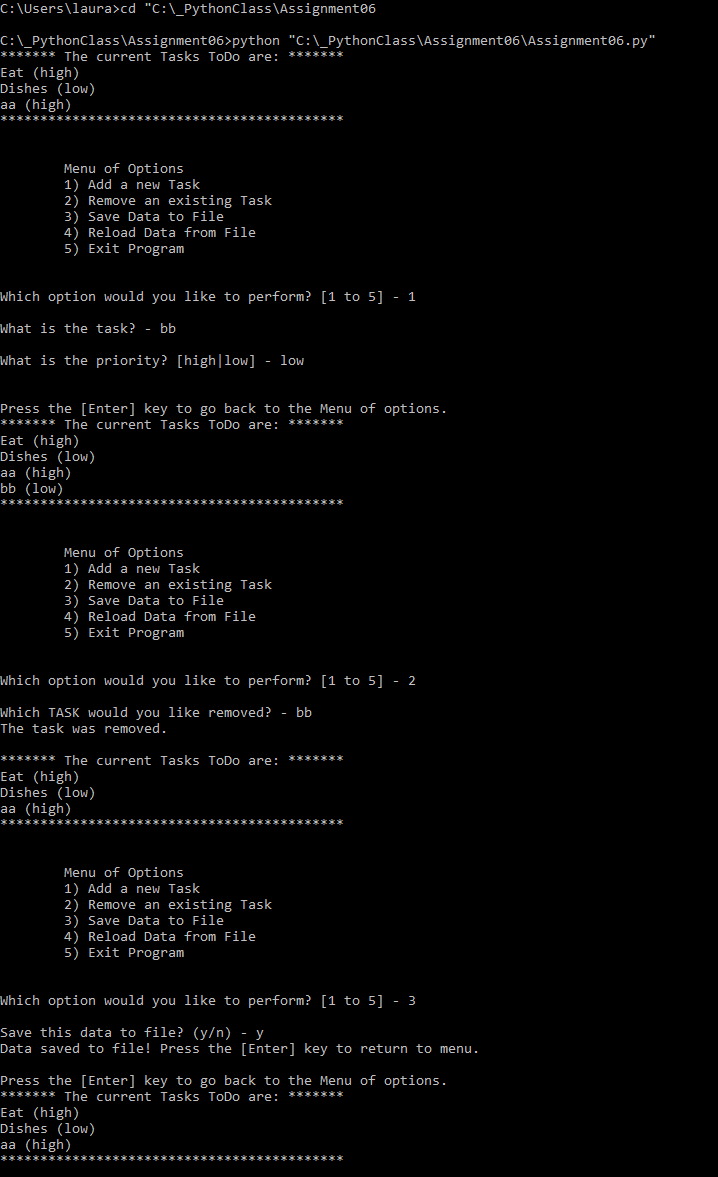
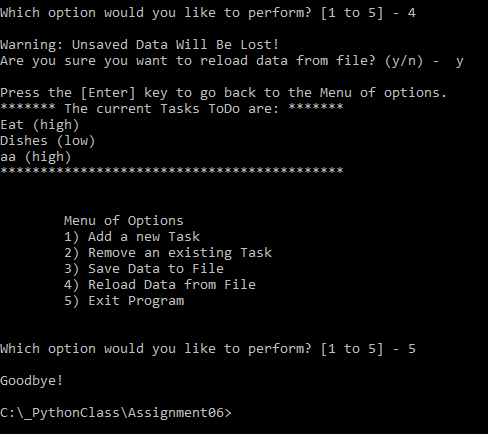


Figure 17



# Summary

The previous week, we learned how to add code to a Python script template and use dictionary objects to edit data in memory and in a file to manage a “To Do List”. This week, we learned how to add functions to our code to act as a brain with data where we can call from the main body of the script (the remote). Using functions is a great way to organize code and return data. This paper outlined the steps I took to create the code to add a new item to the list, remove an item from the list, save tasks to the “ToDoList.txt” file, reload data from the file, and exit the program. I will then run the script in a command shell from the Windows OS using Python version 3.