

Creating Custom Functions

Introduction

Before diving into the scripting, it was important to review the resources recommended by Prof. Root. This included his Module 6 material video and class notes. Additionally, he provided links to external webpages and YouTube videos detailing the same information but presented in a different way. This variety of material was useful in understanding the use of custom functions and their implementation in the main portion of a script. This document provides an outline of the steps required to write a script necessary to allow a user to write and interact with data stored in a file using Python.

The goal of this assignment was to practice editing a script created by another developer and to utilize custom functions to improve the flow of the program. Additionally, the goal of the assignment was similar to the previous two: interact with data stored in a text file and handle the user's inputs in interacting with the data. To complete the assignment, dictionaries were used along with lists to create a table of values, which acted as the temporary storage mode between the user's inputs and the data stored in the text file. The processes were modularized by utilizing functions.

Processing Class Functions

Within the Processing Class of the script, there were several functions which could be added. The first function created was the `WriteListDataToFile()` function, which takes a file name and a list of rows as the arguments, and saves the data temporarily stored in the list to a file. This code was copied and pasted from the main portion of the starter code into the function. This section of the code can be seen in Figure 1.

```
@staticmethod
def WriteListDataToFile(file_name, list_of_rows):
    """
    Desc - This function saves the data stored in the LstTable to a .txt file
    :param file_name: Type String, name of file to write data to
    :param list_of_rows: Type List with dictionary objects, temporary storage of task/priority data
    :return: Nothing
    """
    objFile = open(file_name, "w")
    for dicRow in list_of_rows: # Write each row of data to the file
        objFile.write(dicRow["Task"] + "," + dicRow["Priority"] + "\n")
    objFile.close()
    input("Data saved to file! Press the [Enter] key to return to menu.")
```

Figure 1: Code for function to write the list data to a file.

To implement this function into the appropriate location within the code, the line shown in Figure 2 was added to the main body of the script. By implementing this code in the form of a function,

we're able to more concisely write the main body of the code.

```
elif(strChoice == '4'):
    #Step 3.4.a - Show the current items in the table
    IO.ShowCurrentItemsInList(lstTable) # Show current data in the list/table
    #Step 3.4.b - Ask if user if they want save that data
    if("y" == str(input("Save this data to file? (y/n) - ")).strip().lower()): # Double-check with user
        FileProcessor.WriteListDataToFile(strFileName, lstTable)
    else: # Let the user know the data was not saved
        input("New data was NOT Saved, but previous data still exists! Press the [Enter] key to return to menu.")
# Step 3.5 - Reload data from the ToDoFile.txt file (clears the current data from the list/table)
```

Figure 2 Implementation of WriteListDataToFile function in Main

The next function created for this script was a function to add a new task to the list of tasks saved in memory. This processing function takes the task and priority, which were obtained in the implementation of the GetNewTask() function as described in a subsequent section, as arguments.

```
@staticmethod
def AddTaskToList(strTask, strPriority):
    """
    Desc - This function places the argument task and priority into a dictionary;appends lstTable with the dictionary
    :return: Nothing
    """
    dicRow = {"Task": strTask, "Priority": strPriority} # Create a new dictionary row
    lstTable.append(dicRow) # Add the new row to the list/table
```

Figure 3: Code for function to add a new task to the list

This function was implemented into the program by adding a line of code to call the function in the proper location. This can be seen in Figure 4.

```
elif(strChoice.strip() == '2'):
    # Step 3.2.a - Ask user for new task and priority
    IO.GetNewTask()
    # Step 3.2.b Add item to the List/Table
    FileProcessor.AddTaskToList(lstNewTask[0], lstNewTask[1])
    IO.ShowCurrentItemsInList(lstTable) # Show current data in the list/table
# Step 3.3 - Remove a new item to the list/Table
```

Figure 4: Implementation of functions for getting a new task and adding a new task to the list.

The last function added to the Processing Class was a function used to remove a task from the list. This function was implemented by taking an argument, which was an input from the user as to which task they'd like to remove, and compares the value against the values stored in the list of dictionaries under the Task key. If the value input was existent in the list, the dictionary containing that value is deleted from the list and the function returns True. The code for this function can be seen in Figure 5.

```

@staticmethod
def RemoveTask(keytoremove):
    intRowNumber = 0 # Create a counter to identify the current dictionary row in the loop
    itemremoved = False # Create a boolean Flag for loop
    while(intRowNumber < len(lstTable)):
        if(keytoremove == str(list(dict(lstTable[intRowNumber]).values())[0]).lower()): # Search current row column 0
            del lstTable[intRowNumber] # Delete the row if a match is found
            itemremoved = True # Set the flag so the loop stops
            intRowNumber += 1 # Increase counter to get next row
    return itemremoved

```

Figure 5: Code for function used to remove a task from the list.

This function was implemented by adding a line in the appropriate section of the main part of the program, which can be seen in Figure 6.

```

elif(strChoice == '3'):
    # Step 3.3.a - Ask user for item and prepare searching while loop
    strKeyToRemove = input("Which TASK would you like removed? - ").lower() # get task user wants deleted
    # Step 3.3.b - Search though the table or rows for a match to the user's input
    blnItemRemoved = FileProcessor.RemoveTask(strKeyToRemove)
    # Step 3.3.c - Update user on the status of the search
    IO.CheckIfRemoved(blnItemRemoved)
    #Step 3.3.d - Show the current items in the table
    IO.ShowCurrentItemsInList(lstTable) # Show current data in the list/table

```

Figure 6: Implementation of RemoveTask() function.

IO Class Functions

The next Class we'll talk about is the IO Class. The functions added to this Class were much more simple, but it was valuable to put them into functions in order to improve the flow of the main portion of the program and to improve the readability of the code. The first function added to this Class was the

GetNewTask() function. This function takes no arguments and has no return. The function simple prompts the user for a new task and its priority and saves them to memory as the 0th and 1st indices of a list object. The code for this function can be found in Figure 7.

```

@staticmethod
def GetNewTask():
    """This function:
    -asks the user for a new task and for its priority
    -stores the values input in a List - lstNewTask- for later retrieval
    :return: Nothing
    """
    strTask = str(input("What is the task? - ")).strip() # Get task from user
    strPriority = str(input("What is the priority? [high|low] - ")).strip() # Get priority from user
    lstNewTask[0] = strTask
    lstNewTask[1] = strPriority
    print() # Add an extra line for looks

```

Figure 7: Code for function to create a new task.

The implementation of this function can be seen above in Figure 4.

Finally, the last function that was added to the IO Class was a function to check if an item was removed from the list successfully and prints the results to the user. The argument for the function is of type Boolean and is obtained as described in the section above. The code for this function can be seen in Figure 8.

```
def CheckIfRemoved(item_removed):
    """
    This function prints to the user whether the input task has been removed
    :param item_removed: Type boolean,
    :return: Nothing
    """
    if(item_removed == True):
        print("The task was removed.")
    else:
        print("I'm sorry, but I could not find that task.")
    print() # Add an extra line for looks
```

Figure 8: Code for function to check if a task has been removed from the list.

The implementation of this function can be seen above in Figure 6.

Putting it all Together

When running the program, the execution is done with a while loop so the user can perform any of the operations shown in the main menu until the user selects the “Exit” option, at which point the program breaks the loop and terminates. Sample executions of the program can be seen in Figures 5 and 6.

```
C:\Users\Josh\python.exe C:/_PythonClass/Assignment06/Assignment06.py

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

Which option would you like to perform? [1 to 6] - 1

***** The current items ToDo are: *****
Work (High)
Running (High)
Fantasy Football (Low)
*****

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
```

Figure 9a: Sample execution of program in PyCharm

```

Which option would you like to perform? [1 to 6] - 2

What is the task? - Homework
What is the priority? [high|low] - High

***** The current items ToDo are: *****
Work (High)
Running (High)
Fantasy Football (Low)
Homework (High)
*****

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

```

Figure 9b: Sample execution of program in PyCharm

```

Which option would you like to perform? [1 to 6] - 4

***** The current items ToDo are: *****
Work (High)
Running (High)
Fantasy Football (Low)
Homework (High)
*****

Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

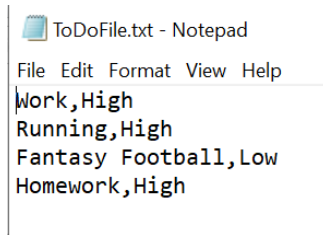
Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

Which option would you like to perform? [1 to 6] - 6

Process finished with exit code 0

```

Figure 9c: Sample execution of program in PyCharm



ToDoFile.txt - Notepad

File Edit Format View Help

Work,High
Running,High
Fantasy Football,Low
Homework,High

Figure 9d: Resultant text file after sample program execution.

```
C:\_PythonClass\Assignment06>python Assignment06.py

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

Which option would you like to perform? [1 to 6] - 3

Which TASK would you like removed? - Fantasy Football
The task was removed.

***** The current items ToDo are: *****
Work (High)
Running (High)
Homework (High)
*****

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

Which option would you like to perform? [1 to 6] - 4

***** The current items ToDo are: *****
Work (High)
Running (High)
Homework (High)
*****
```

Figure 10a: Sample execution of program in command line.

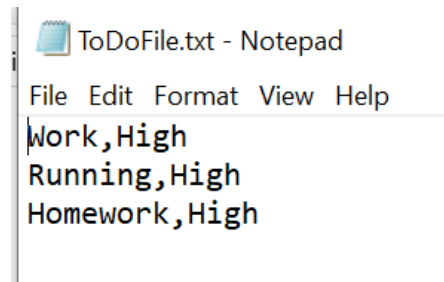
```
Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

Menu of Options
1) Show current data
2) Add a new item.
3) Remove an existing item.
4) Save Data to File
5) Reload Data from File
6) Exit Program

Which option would you like to perform? [1 to 6] - 6

C:\_PythonClass\Assignment06>
```

Figure 10b: Sample execution of program in command line.



```
ToDoFile.txt - Notepad
File Edit Format View Help
Work,High
Running,High
Homework,High
```

Figure 101: Resultant text file from sample execution.

Summary

This document demonstrated the process for editing a script to handle a user's modifications to a To-Do List stored in a text file. This was implemented using lists and dictionaries in order to efficiently store the data temporarily before ultimately writing the information to a text file. As a note, it was important to keep the code added to the .py file as organized as possible in order to facilitate good collaboration with the author of the script. To do so, meaningful comments were added, and code was broken up into logical sections within the script. In addition, by utilizing functions, further organization has been added, and redundancy of code has been reduced. The flow of the script was improved by using custom functions within the script and calling them, as necessary, in the main portion of the program.