**Kate Pollock**

**November 19, 2021**

**Foundations of Programming: Python**

**Assignment06**

**GitHubURL <** [**https://github.com/katepollock/IntroToProg-Python-Mod06/upload/main**](https://github.com/katepollock/IntroToProg-Python-Mod06/upload/main) **>**

# **Functions – To Do List**

# Introduction

Module 6 is mainly about functions. I learned how to define and call functions as well as the definitions of parameters and arguments. I learned how to return the values of a functions and how to store them in variables to use later. I reviewed the concepts of local and global variables which I believe was helpful to my overall understanding of functions. I began to review the concept of classes and how similar functions may be grouped into them. In addition to functions, I also explored the PyCharm debugging tool. I used all of the above concepts when completing assignments for week 6.

# Planning my “To Do ” Script

This assignment was very similar to the previous assignment in that it creates a To Do list with tasks and priorities. However, the goal was to use functions in place of the repetitive code in Assignment 05. I used the Assignment 06 started file which provided a template for the basics of the code. The code was divided into 3 main sections – Processing, Input/Output and Main Body. Both Processing and Input/Output are the names of classes with multiple functions in each while Main Body walks through the menu choices with if/elif statements similar to Assignment 05. Much less code was required in the Main Body section as functions calls were utilized throughout it.

# Processing Class ***(See Figure 1)***

The processing class was composed of the following functions: read\_data\_from\_file, add\_data\_to\_list, remove\_data\_from\_list, and write\_data\_to\_file. The code within each of these functions was fairly similar to Assignment 05. The following is how I structured each of the functions:

* Function “read\_data\_from\_file” has 1 parameter(file\_name) and returns a list\_of\_rows. The function opens and reads the file, processes the data using a loop and splitting each line per the file into 2 variables (task and priority). These variables are added to a list as a dictionary row and the file is closed. Because we are writing from the file each time the code runs, it is necessary to ensure that the list is empty so additional data isn’t added to the list, therefore, I set list\_of\_rows equal to an empty list.
* Function “add\_data\_to\_list” has 3 parameters (task, priority, and list\_of\_rows) and returns list\_of\_rows. I appended a dictionary row composed of task and priority as keys. The arguments passed in for task and priority are added to the row as values. The dictionary row is appended to the list. I was able to call this function in the “read\_data\_from\_file”.
* Function “remove\_data\_from\_list” has 2 parameters (task, list\_of\_rows) and returns list\_of\_rows. I utilized a loop to check if there is a task in the row that is equivalent to the argument ‘task’. If True, I used the remove method to delete that dictionary row from the list.
* Function ”write\_data\_to\_file” has 2 parameters (file\_name and list\_of\_rows) and returns list of rows. I opened the file and looped through the dictionary rows, adding the task and priority separated by a comma and a carriage return for each row.

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

***Figure 1: Processing Class code***

# IO Class ***(See Figure 2)***

The IO class was composed of the following functions: print\_menu\_tasks, input\_menu\_choice, print\_current\_tasks\_in\_list, input\_yes\_or\_no\_choice, input\_new\_task\_and\_priority, and input\_task\_to\_remove.

* Function print\_menu\_tasks executes a print statement which gives the user a list of menu options.
* Function input\_menu\_choice prompts the user to choose a menu option. I utilized a local variable choice for the input which is returned when the function is called.
* Function print\_current\_tasks\_in\_list has 1 parameter (list\_of\_rows) and executes a print statement which loops through the list and displays the tasks and priorities.
* Function yes\_or\_no\_choice has one parameter (message) and returns a Boolean. The user is prompted to input yes or no and those answers are set to True and False.
* Function input\_new\_task\_and\_priority prompts the user to input a task and a priority. If a user would inadvertently add a comma, this would cause problems when writing to the text file and loading from the file. Therefore, I used the replace method to remove any commas. Both task and priority are returned.
* Function task\_to\_remove prompts the user to input a task to remove. That task is returned.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

***Figure 2: Input/Output Class code***

# Main Body of Script ***(Figure 3)***

The main body of my code is very similar to Assignment 05, but it was much more concise and non-repetitive due to the ability to call the functions that I defined in the Processing and IO Class sections.

I began by using the function read\_data\_from\_file to populate my list variable (lstTable). In the case that the file did not yet exist, I used a try except statement for the File Not Found error. I used a while True statement to keep looping through the code until the user exits. I print the menu for the user by calling the print\_menu\_tasks function. I then prompt the user to enter his or her choice by using the input\_menu\_choice and set this equal to the strChoice variable.

Once the user has made their choice, I use if-elif statements to process the code accordingly.

* Option 1 (adding a new task/priority) calls input\_new\_task\_and\_priority and unpacks the returned tuple into 2 variables (strTask and strPriority). It then calls add\_data\_to\_list by passing these two variables and lstTable to return the updated list. I’m using a Boolean to identify changes to the list which will be utilized in Options 4 and 5. Print\_current\_tasks\_in\_list is called to display the new list to the user.
* Option 2 (removing a task/priority) calls print\_current\_tasks\_in\_list so the user can see what tasks are available to remove. StrTask is set equal to the function call task\_to\_remove. Remove\_data\_from\_list is called and returns back an updated list. I’m using a Boolean to identify changes to the list which will be utilized in Options 4 and 5. This list is then printed so the user can validate that the task and priority have been removed.
* Option 3 (adding data to file) calls the function input\_yes\_or\_no\_choice and if True, the function write\_data\_to\_file is called by passing the file name (strFile) and the updated list (lstTable). The ‘unsaved changes’ is set to False as the data has been saved. Then the current list is printed for the user.
* Option 4 (reloading to file without saving) prints a warning message that data will not be saved to file. If the Boolean value from unsaved changes is false indicating that data has been saved or the user indicates that they want to reload the data, read\_data\_from\_file is called and the list (lstTable) is updated to what was in the file. Then the function print\_current\_tasks\_in\_list is called and it displays the list. If the user elected not to reload, an else statement states that the file wasn’t reloaded.
* Option 5 (exit program) evaluates if the Boolean is true (there are unsaved changes) and the user does not want to exit without saving, in which case, the program is continued. If data has been saved or the user wants to exit without saving, the program breaks and exits.
* The Else statement provides the user feedback should they have selected an option outside of 1 to 5.

*# Main Body of Script ------------------------------------------------------ #  
  
# Step 1 - When the program starts, Load data from ToDoFile.txt.***try**:  
 lstTable = Processor.read\_data\_from\_file(strFileName)  
**except** FileNotFoundError:  
 **pass  
  
while True**:  
 *# Display a menu of choices to the user* IO.print\_menu\_tasks()  
 strChoice = IO.input\_menu\_choice()  
 print() *# adding a new line for looks* **if** strChoice == **'1'**: *# Add a new task and priority* strTask, strPriority = IO.input\_new\_task\_and\_priority()  
 Processor.add\_data\_to\_list(strTask, strPriority, lstTable)  
 unsaved\_changes = **True** print(**'Task added!'**)  
 IO.print\_current\_tasks\_in\_list(lstTable)  
  
 **elif** strChoice == **'2'**: *# Remove a task* IO.print\_current\_tasks\_in\_list(lstTable)  
 strTask = IO.input\_task\_to\_remove()  
 Processor.remove\_data\_from\_list(strTask, lstTable)  
 unsaved\_changes = **True** print(**'Task removed!'**)  
 IO.print\_current\_tasks\_in\_list(lstTable)  
  
 **elif** strChoice == **'3'**: *# Save Data to File* response = IO.input\_yes\_no\_choice(**'Would you like to save your data to the file [yes or no]? '**)  
 **if** response:  
 Processor.write\_data\_to\_file(strFileName, lstTable)  
 unsaved\_changes = **False** print(**'Data saved!'**)  
 IO.print\_current\_tasks\_in\_list(lstTable)  
 **else**:  
 print(**'Data has not been saved.'**)  
  
 **elif** strChoice == **'4'**: *# Reload Data from File* **if not** (unsaved\_changes **and not** IO.input\_yes\_no\_choice(  
 **"Are you sure you want to reload data from file w/o saving [yes or no]? "**)):  
 lstTable = Processor.read\_data\_from\_file(strFileName)  
 print(**'Data reloaded'**)  
 IO.print\_current\_tasks\_in\_list(lstTable)  
 **else**:  
 print(**"File data was not reloaded"**)  
  
 **elif** strChoice == **'5'**: *# Exit Program* **if** unsaved\_changes **and not** IO.input\_yes\_no\_choice(**"Are you sure you want to exit without saving [yes or no]? "**):  
 print(**'Enter option 3 to save data.'**)  
 **continue** print(**"Goodbye!"**)  
 **break  
  
 else**:  
 print(strChoice, **"is not an option. Please select [1 to 5]."**)

***Figure 3: Main Body of Script Code***

# Results of Script *(Figure 4)*

Below is an example of the output run in the command prompt:

Text

Description automatically generated

***Figure 4: Output in command prompt***

# Summary

I have written the Python program above by utilizing the new concepts learned in Module 6 of this course. This Module was heavily focused on learning about functions. This week has been very challenging, as It’s been tricky to keep things working when functions call functions. The advice given regarding breaking the code up into smaller pieces and getting that working prior to moving on was helpful. Overall, it was satisfying to get my code working.