Alex Frain

February 21, 2021

IT FDN 110 B – Foundations of Programming, Python

Assignment06

{NEED TO UPDATE!!!!!} https://github.com/alexfrain/IntroToProg-Python

Working with Functions in a Class -“To Do List” Revised

# Introduction

The Python script created in this assignment is a revised version of the “To Do List” script originally created in Assignement05. It has been improved upon programmatically as it introduces Functions and Classes. The script has the same functionality in that it reads in a To Do List from a text file into a list of dictionary items, gives the user the ability to add and remove tasks, and write changes back to the text file but the code has been reorganized into functions which also helps address a Computer Science concept introduced last week called Separation of Concerns. A couple other topics covered by this script include the use of global variables vs local variables and the ability to pass and return data to and from function by reference.

# Creating the Script

We were provided with starter template that has the script divided up into four sections: Data, Processing, Presentation, and the Main Body. These sections better organize the code into units that each handle a specific purpose which is the concept of Separation of Concerns. I find this makes the code easier to read as you know what each section handles with regards to the script and it’s easier to troubleshoot issues since you’ll have a better idea of where to look in your code if an issue is encountered. I imagine it would be especially helpful with logic errors where your code runs but produces unexpected results. You won’t be given a line number to check like you would with a syntax error but with code divided up by functionality you’ll have a better idea of which area in your code to start looking.

In the Processing and Presentation sections of the template we were also provided with classes and class functions that would address all the tasks required in the script. “A function is a block of code which only runs when it is called.” (<https://www.w3schools.com/python/python_functions.asp>, 2021) (External Link). Values can be past into the function for processing through parameters which are enclosed in the parentheses in the function definition. Functions also have the ability to return one or more values which are generally placed inside a variable when returned. A class is used to group related functions and variables. Classes and functions need to be defined before they can be used by the script so they are physically located above the main body of the script.

The lines of code from the previous assignment that handle processing data is being moved into the Processor class which contains predefined functions for reading data from a file, adding data to a list, removing data from a list, and writing data to a file. The tasks that handle asking the user for data and displaying the data to the screen are addressed by predefined functions in the IO class that is contained within the Presentation section of the script.

The Data section listed at the top of the script is where all variables used by the main body of the script are declared and initialized. They are considered global variables as they can be used anywhere in the script. The predefined functions in this script also contain variables but they are considered local variables as they can only be used within the function or class in which they were declared. Local variables cannot be used by the main body of the script and would result in an error if this was attempted.

The main body of the script is where the functions are called to perform all the processing and input/output tasks. Just like in the original To Do List script from last week, the main body is composed of a while loop that lets the user continually loop through the script to continually perform these various tasks until they choose exit.

In order to complete the script I found the nine areas in the template where we were to add our code and I grouped them logically into four basic task sections that I would tackle one at a time: Adding Data to a List, Remove an Existing Task, Save Data to a File, and Reload Data from a File. These four sections, which are further detailed below, required adding code to different functions from the two classes previously mentioned along with adding function calls in the main body of the script.

## Adding Data to a List

Adding data to a list requires asking the user to enter a task and its priority which is handled by the function *input\_new\_task\_and\_priority()* (Figure 1) located in the Presentation section of the script. The task and priority are captured via separate input statements and assigned to variables with the same name, respectively. These two variables are then returned by the function to the main body of the script.

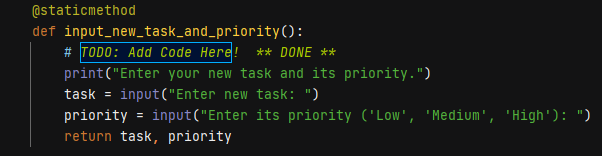


Figure 1: input\_new\_task\_and\_priority() function

The act of actually adding the data to a list is handled by the *add\_data\_to\_list()* function found in the Processing section (Figure 2). The task and priority which contain the values entered by the user and list containing all the tasks and their priorities are passed as arguments to the function. The task and priority are assigned to a dictionary item and that dictionary item is then added to the list using the list’s *append()* function. Finally, the list is returned by the function along with a literal ‘Success’ string that is used by another predefined function that indicates the block of code in the function completed.

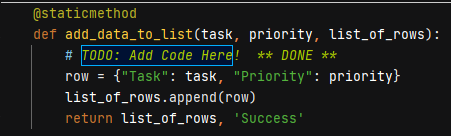


Figure 2: add\_data\_to\_list() function

There are a couple of additional things to make note of with regards to what’s happening in this function and others like it in this script. If multiple values are returned by a function they are packed into a tuple. This tuple can be unpacked where the function was called and the values can be assigned to other variables. The second note is that when passing lists to a function or returning lists from a function they are done so as a reference to a value that exists in memory. Any processing done on this reference affects the existing value in memory. The lstTable global variable and the list\_of\_rows local variable which are both list variables reference the same list value in memory when working with this script. This is different than passing item types like strings, integers, floats, etc that are passed by value or copies of existing values.

Finally, in the main body of the script, menu option 1 refers to the user’s choice to add a new item to their To Do list which is within an *if* statement. This is where the calls to the two functions are made (Figure 3). The call to the *input\_new\_task\_and\_priority()* is called first to get the user’s input for the task and priority to add and the tuple that is passed back by this function is unpacked into two global variables, *strTask* and *strPriority*. Then the *add\_data\_to\_list()* is called and is passed the two global variables as arguments that now containing the user’s input. That function returns a list and a string in a tuple which is unpacked into two more global variables, *lstTable* and *strStatus*. It should also be noted that since these functions are part of a class they need to be preceded by the class name when being called.

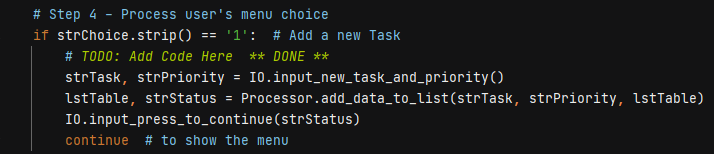


Figure 3: Main script body - Add a new Task

## Remove an Existing Task

The next section of the script I addressed was removing an existing task from the To Do list. There is a function in the Presentation section called *input\_task\_to\_remove()* that handles getting input from the user as to which task is to be removed which is assigned to a local string variable. This variable is then returned by the function (Figure 4).

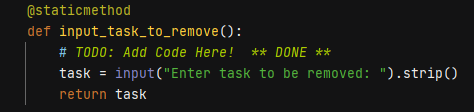


Figure 4: input\_task\_to\_remove() function

In the Processing section of the script is a function called *remove\_data\_from\_list()* that processes the removal of the task entered by the user from the list (Figure 5). The string variable containing the task along with the list of tasks is passed to this function as arguments. The function uses a for loop to go through the list to check if the task entered by the user is in the list. If it is then the dictionary item containing this task is removed from the list. I also declare a local Boolean variable, *task\_found*, and initialize it to false before the *for* loops runs. If the task to be removed is found in the list this variable is then set to True. After the list has been looped through I check if this variable is False which would indicate the task did not exist in the list and I print a message to the user indicating this. Finally, the function returns the list along with a literal ‘Success’ string in a tuple.

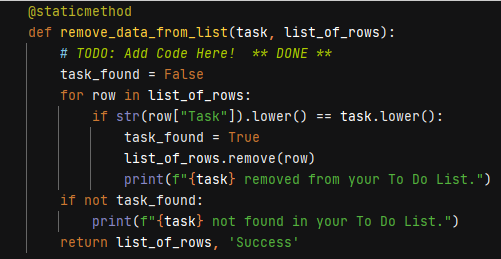


Figure 5: remove\_data\_from\_list() function

Then I addressed the section within the main body of the script that handles the menu choice for removing an existing task which is part of an *elif* statement. I first called the *input\_task\_to\_remove()* function and assign it to the *strTask* variable since the function returns the task input by the user. I then call the *remove\_data\_from\_list()* function passing in the *strTask* variable along with the list table, *lstTable*, in order to process the task removal and I unpack the returned tuple into the *lstTable* and *strStatus* variables (Figure 6).

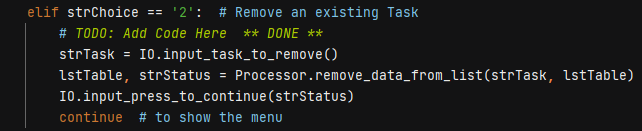


Figure 6: Main script body - Remove an existing Task

## Save Data to File

The third section that I addressed was saving the data to a file which had two areas where code needed to be added. Within the Processing section there is a function called *write\_data\_to\_file()* that takes in the name of the file and the list containing the tasks. The function then opens this file in write mode and then uses a *for* loop to loop through each row in the list and write it to the file. Once it’s done looping, the file object is closed and the list of tasks and a literal ‘Success’ string is returned packed in a tuple (Figure 7).

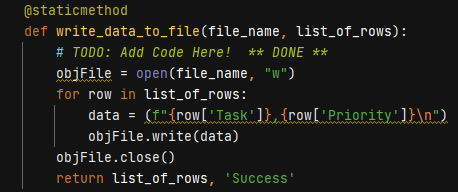


Figure 7: write\_data\_to\_file() function

In the main body of the script I located the menu choice for saving the data. I added a single line of code that calls the *write\_data\_to\_file()* function (Figure 8) passing in the global variables, *strFileName*, that contains the file name, and *lstTable* containing the task list and it saves the To Do list to the file. Since this function returns the list of tasks and a string I unpack them by assigning them to the *lstTable* and *strStatus* global variables respectively.

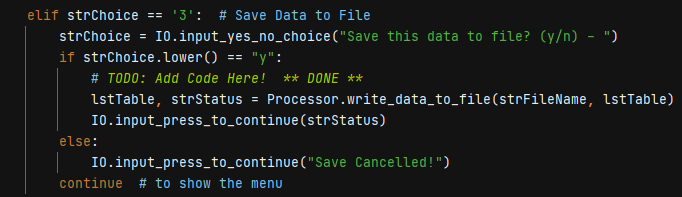


Figure 8: Main script body - Save data to File

## Reload Data from File

The final section requiring code to be added was reloading data from a file which overwrites the list of tasks in memory with list of tasks contained in the file on the computer. The function *read\_data\_from\_file()* was completely defined for us as part of the initial script template so I just needed to call the function in the main body of the script within the applicable menu option (Figure 9). I passed in the *strfileName* and *lstTable* global variables as arguments to this function which contained the name of the file to be read from and a reference to the list of tasks, respectively. The function handled opening the file in read mode, adding each task and associated priority to a dictionary item, and then adding that dictionary item to the list. It then closed the file object and returned the list of tasks along with a literal ‘Success’ string packed into a tuple. Back in the main body this returned tuple was unpacked and assigned to the *lstTable* and *strStatus* variables.

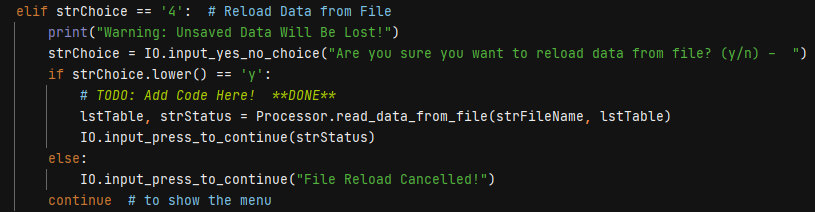


Figure 9: Main script body - Reload data from File

# Running the Script

I ran the script in both PyCharm and Windows Command line to confirm it worked in both. I went through the same verification in both environments by hitting each menu option to verify I would get the expected results.

I had an existing ToDoFile.txt file on my PC in the same directory as the .py file. I ran the script to confirm it would read in the existing tasks from the file into memory and then present the Menu of Options (Figure 10).

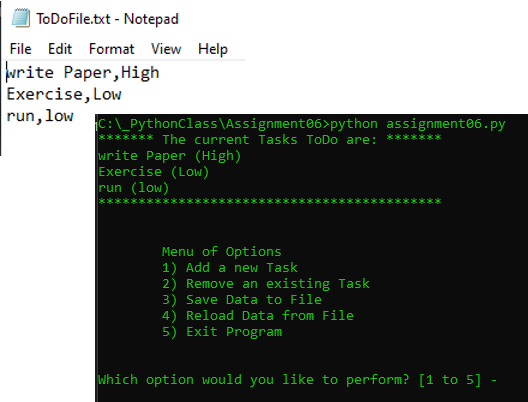


Figure 10: Read in tasks from text file

I then tested option 1, Add a new Task, confirming it successfully added a task to the list in memory (Figure 11).

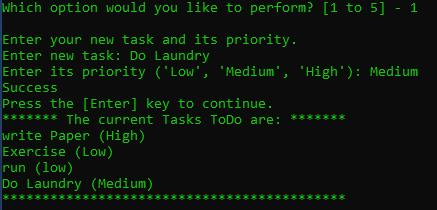


Figure 11: Add a new Task

I followed that up by testing option 2, Remove an existing Task. I entered a task from the list confirming it was removed and then entered a task that wasn’t in the list and confirmed I received a message stating the task was not found in the list (Figure 12).

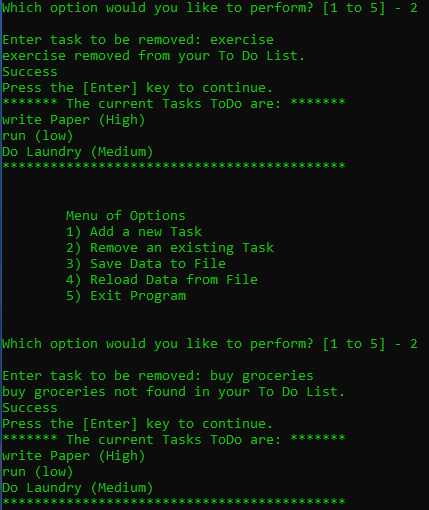


Figure 12: Remove an existing Task

Next, I tested menu option 3, Save Data to File, and confirmed the current list saved to the file on my PC (Figure 13).

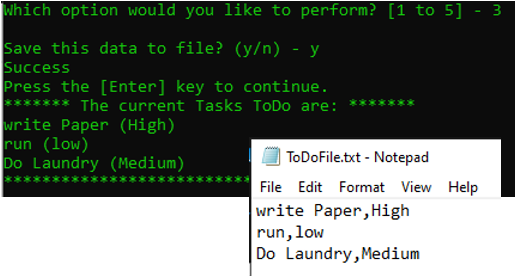


Figure 13: Save Data to File

I then tested menu option 4, Reload Data from File. I added some new items to the list that’s loaded in memory and then reloaded data from the file confirming the list in memory was overwritten with the list from the file (Figure 14).

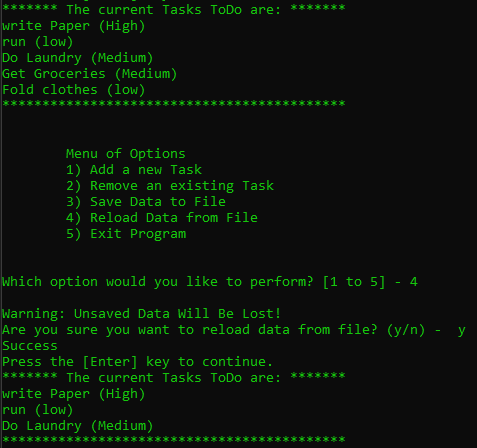


Figure 14: Reload Data from File

Lastly, I selected menu option 5, Exit Program, to confirm the script would exit successfully…which it did (Figure 15).

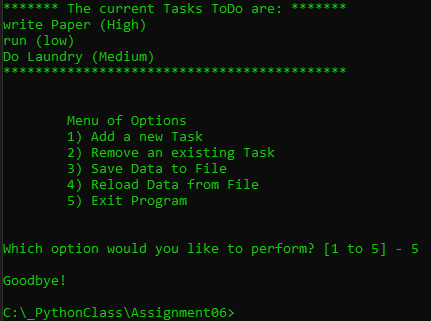


Figure 15: Exit Program

# Summary

In this assignment I revised the original To Do List script from Assignment05 moving existing code into various functions that belong to two classes. Using functions in a program has many benefits, one of which is that it helps address the programming concept of Separation of Concerns. A block of code that addresses a single specific task can be placed in a function which can be called by the main body of the program when needed. Related functions can be grouped together into a class like what was done in this script. A class was created for processing data and contained all the process related functions and a separate class was created for presentation and the functions that handled input/output tasks were contained here. Data can be passed into function as arguments that the function can then perform operations on this data and the function can return data back to the main body of the program using a return statement. Some other important topics covered in this assignment included creating global variables (available to the entire program) and local variables (available only within the function that the variable is declared). Another topic was the ability to pass and return data by reference which was done with a handful of functions in this script. Data types like a list can get passed into functions as a reference meaning that any changes made to the list in the function affect all variables that point to that list whether in the main body of the script or in the function. They all refer to the same list value in memory. This is different from other item types like strings, integers, and floats that would get passed or returned as a value which means basically means the value gets duplicated in memory.