Revisiting the To Do List: Classes and Functions

## Introduction

This week’s assignment took our previous to-do list script and improved upon it using classes, functions, and the concept of “separation of concerns”. At first glance, this makes the script seem much more complex. However, it ultimately makes the process the script is following much clearer to understand as each section of the script accomplishes a specific task. It also allows the programmer to repeat tasks quickly by calling the function rather than rewriting the same code.

## Topic 1 – Classes

The first thing I noticed when opening the starter script is that everything had already been organized into the “Separation of Concerns” discussed in class. Further, each group of functions (both those that had been provided and those that were yet to be written) were grouped into *classes*. Classes organize functions and when using an IDE allow the programmer to quickly call various functions as the IDE will suggest available functions in that class (“Python Classes/Objects”, <https://www.w3schools.com/python/python_classes.asp>, 2020)(External Site).

The two classes included were Processor and IO. There was no class for the main body of the script because that script simply relied on calling functions from the other classes. The classes helped clarify what each portion of the main script was doing. For example, if the user chose option 1 (add new task to the list), you can see that action requires some I/O work, then processing that data before continuing (Figure 1)

if strChoice.strip() == '1': # Add a new Task  
 task, priority = IO.input\_new\_task\_and\_priority() # get user input and

assign to task and priority variables  
 Processor.add\_data\_to\_list(task, priority, lstTable) # add new task and

priority to list

**Figure 1: Option one uses two functions, an IO function and a Processor function**

## Topic 2 – Functions

Within each class, there were a variety of functions that performed various tasks in the main body of my script. Functions only run when they are called and can be reused indefinitely (“Python Functions”, <https://www.w3schools.com/python/python_functions.asp>, 2020)(External Site). The action that is performed and/or data returned will depend on what data is passed in as arguments into the function. This assignment had us write several functions in both the Processor and IO classes.

One example of a function from the Processor class was adding a new task to the list. This function has three parameters: user\_task, user\_priority, and list\_of\_rows (Figure 2). User\_task and user\_priority are both pieces of data that are input by the end user. Initially, I thought I needed to include the input statements within this particular function. However, I later realized that the only purpose this function was meant to serve was to add data to the list**.** The process of getting user input for a new task and priority would actually be achieved by a *separate* function in the IO class!

def add\_data\_to\_list(user\_task, user\_priority, list\_of\_rows):  
 row = {"Task": user\_task, "Priority": user\_priority}  
 list\_of\_rows.append(row)  
  
 return list\_of\_rows, 'Success'

**Figure 2: Parameters of my function and the actual code performed by the function**

This function creates a new dictionary to be added as a new item to the list defined in the function’s third parameter, list\_of\_rows. Eventually, I would be using an IO function to capture user input that would be used as the arguments for the user\_task and user\_priority parameters.

Clearly, this function needed to take in some data as the user\_task and user\_priority parameters. I could simply get that data using the input() function every time I needed it, but instead we created a new function to do that for us. This was shown as a new function in the IO class: input\_new\_task\_and\_priority().

This function did not have any parameters, it just returned the two inputs as a tuple. This is useful because my final script only had one line of code to capture these user inputs, making things much cleaner (Figure 3, 4)

task, priority = IO.input\_new\_task\_and\_priority() # get user input and assign to task and priority variables

**Figure 3: One line of code to get user input, rather than the three lines we see in the code block of the function (Figure 4)**

def input\_new\_task\_and\_priority():  
 *""" Get user input for new task and priority  
  
 :return: string, string  
 """* user\_task = input("Please input task: ")  
 user\_priority = input("Please input priority: ")  
 return user\_task, user\_priority

**Figure 4: Code block of the function**

One line vs three lines doesn’t seem like much in this context, but it is clear that for larger, more complex programs these differences would quickly add up, or even be much larger (e.g. one line vs 10, 20, many lines!).

## Topic 3 – Using classes and functions in the main body of the script

By using all these classes and functions, the main body of the script was able to rely almost entirely on them as the program looped through each of the users’ choices. Essentially, the program ran until the user chose to exit, and while it was running gave the user the option to choose various actions.

Option 1 is a great example of using the various functions created earlier in the script (Figure 5).

if strChoice.strip() == '1': # Add a new Task  
 task, priority = IO.input\_new\_task\_and\_priority() # get user input and

assign to task and priority variables  
 Processor.add\_data\_to\_list(task, priority, lstTable) # add new task and

priority to list  
 IO.input\_press\_to\_continue(strStatus)  
 continue # to show the menu

**Figure 5: Body of the script for if the user chooses option 1.**

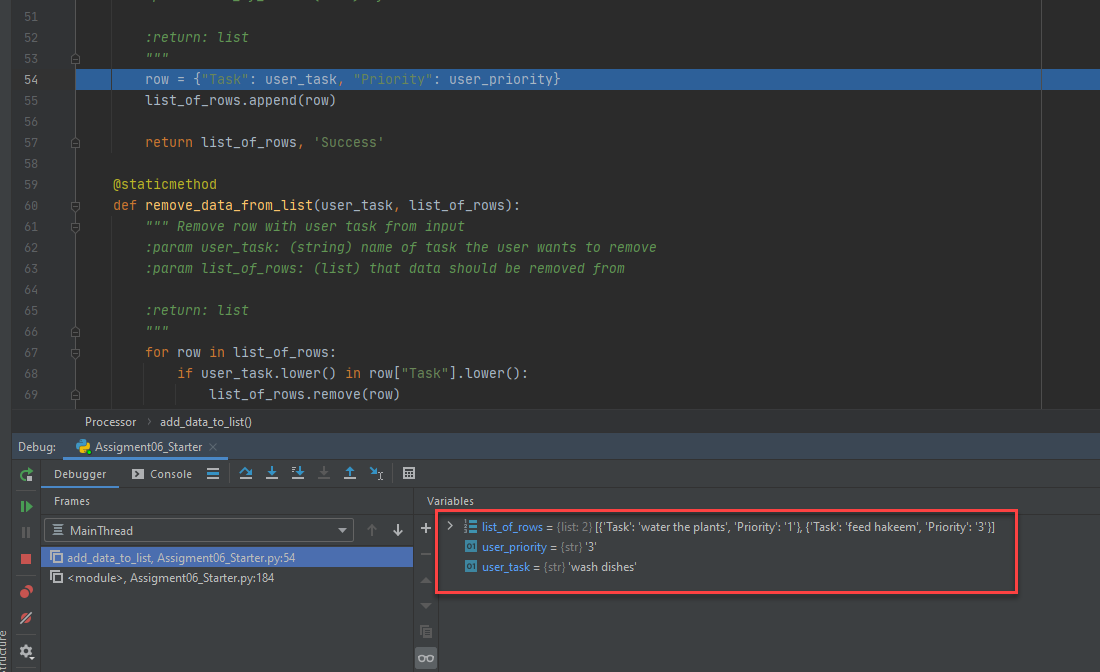
First, the script called my IO function, input\_new\_task\_and\_priority. This function asks the user to input a new task and function, then returns their choices in a tuple. I assigned the response to two variables: task and priority.

Next, I was able to pass those variables as arguments into the Processor function add\_data\_to\_list. This function also requires an argument stating what list the data will be added to – this was the list that had been initialized at the very beginning of my script, lstTable.

This portion of my script, including some other lines for looks and user messages, was only 5 lines. Comparing to a similar process in Assignment05, this was 3 lines shorter. Again, it’s only a small difference, but would quickly add up over thousands of lines of code in a more complex script.

## Topic 4 – Debugging

This was the most complex script we’ve written so far, and I naturally encountered a few issues over the course of development. The debugger tool in PyCharm was very useful for identifying and fixing various issues as I was able to “step into” my code to see how variables were being assigned and ensuring all of my functions were working correctly (Figure 6).



**Figure 6: Adding a stop point after calling the add\_data\_to\_list function to check my variables were being assigned correctly.**

This helped me make sure that my functions didn’t have any variables that were shadowing other variables and that everything was being appended to my list correctly (“Debug your first Python Application”, <https://www.jetbrains.com/help/pycharm/debugging-your-first-python-application.html#where-is-the-problem>, 2020)(External Site).

## Conclusion

This script was much more challenging than previous versions, but once I better understood how to work with functions and classes the benefits of doing so were clear. Functions make each task much more abstract, but that abstraction allows them to be much more versatile than simply writing processes multiples times throughout the script. Additionally, as we venture into more complex programs, it was helpful to explore PyCharm’s debugging tools.