Aaron Lanphear

February 21 2021

IT FDN 110 B Wi 21: Foundations of Programming: Python

Assignment 6

<https://aaroncl13.github.io/IntroToProg-Python-Mod06/>

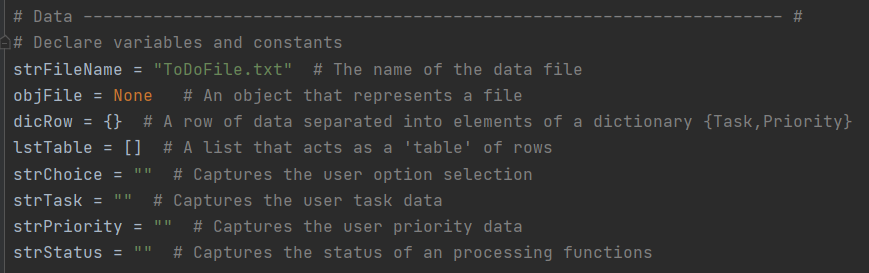
# To-Do List – Classes and Functions

## Introduction

This week we learned about classes and functions. We learned how they can be used to complete repetitive tasks, simplify your code and organize your script. For this assignment, we continued working with our to-do list, but updated our code to include these classes and functions.

## The Start of the Script

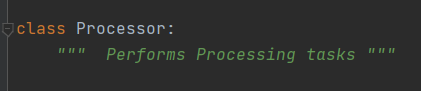
This week we, again, started off the script by declaring all of our variables (Figure 1). While not a requirement with python, it *can* be for other programming languages, so this was a good way for us to try it out. It can also make reading code, or editing it, much easier down the road. Especially for another programmer.



***Figure 1: Declaring the script’s variables***

## Creating the First Class

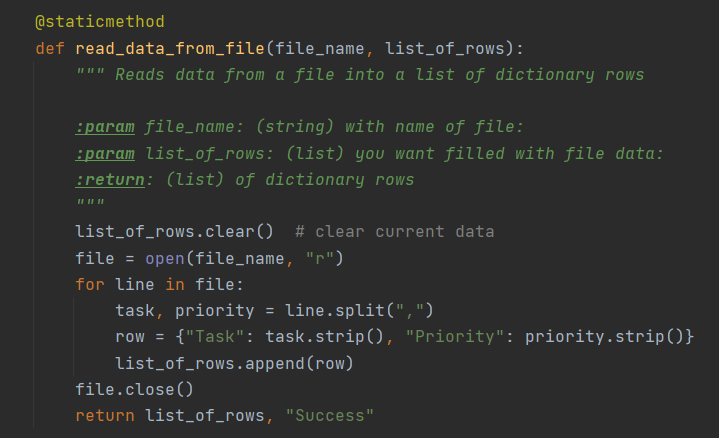
Instead of moving on to script that begins processing data, like we did last week, this week we begin by creating our first class. A class contains functions that will be used for a similar purpose. In this case, the class is called *Processor*. The functions within it will be used to be *process* data. Creating the class is simple. You just type “class” followed by the name (Figure 2).



***Figure 2: Creating the Processor class***

### The Processor Functions – Parameters and Returns

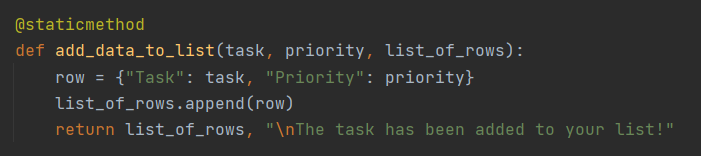
Once this has been done, you can begin creating your functions within it. Doing this is simple as well. You begin by typing “def” to define your function. You give it a name and then provide any inputs (parameters) the function may have. If none are needed, then the parenthesis can be left empty. In this first example, the function will read data from the file. To work, it will need to know the file name and the list it will be putting the data from the file in to (Figure 3). So, these are provided as parameters.



***Figure 3: Creating our first function. This function will read data from the file.***

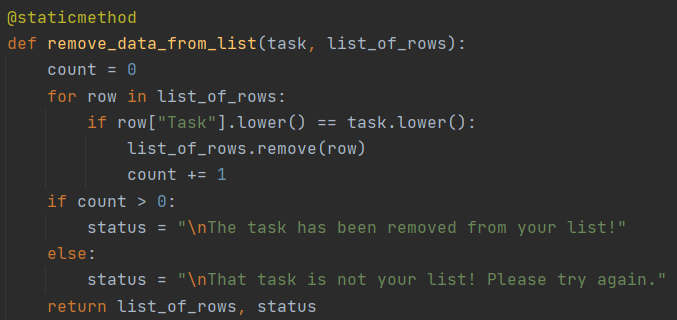
This code is similar to what we used last week, but instead of being part of the main script, it’s contained within this function, to be called later. One other part of the function is the output. This is defined with the code “return”. As this function processes, it takes the list that was inputted, updates it, then returns it to the program, along with a message.

The next processing function adds data inputted by a user to the list. Its parameters are those three things. The two inputs by the user and the list they will be added to. It returns the newly updated list and a message that the task has been added (Figure 4).



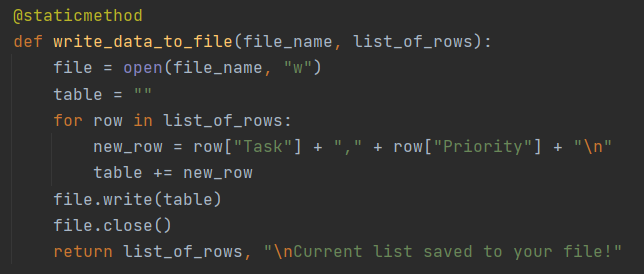
***Figure 4: Adding data to the list function***

The third function the processor class removes data from the list. Similar to the previous functions, this takes the parameter of the task the user inputs and the list that needs to be updated. It includes a conditional return statement that is dependent on whether the function found a matching value to remove. This is done by using a counter that tracks whether a match was found or not (Figure 5).



***Figure 5: Removing data from the list function, which includes a conditional status.***

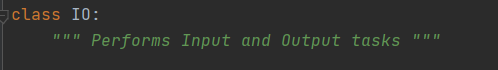
The final function in the Processor class writes the data to the file. Similar to the read function, it takes the file and list parameters. The difference here is that this function opens the file in write mode instead of read. It also, like the read function, returns the updated list and a user-friendly message (Figure 6).



***Figure 6: Writing data to file function***

## The Presentation Class

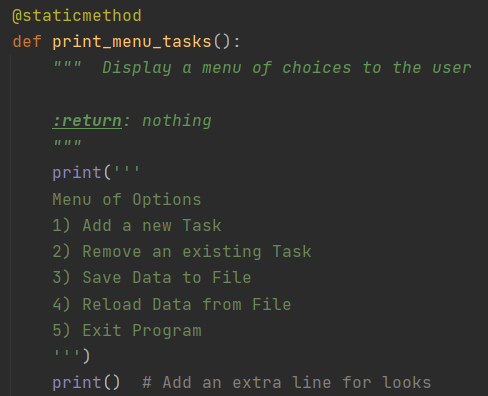
The second class we create is for input, allowing the user to enter data, and output, allowing the program to display something back to the user. We call this “IO”, for input/output (Figure 7).



***Figure 7: Creating the IO class***

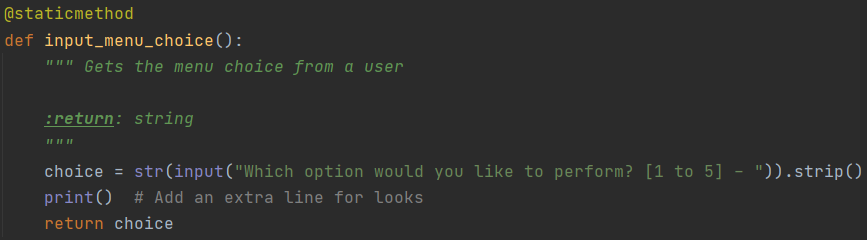
### The Input/Output Functions

All of the functions in this class allow us to display something back to the user or accept input from them. These inputs will be used later for providing parameters to our Processer functions or during conditional statements in the main code. The first function is a simple menu display (Figure 8).



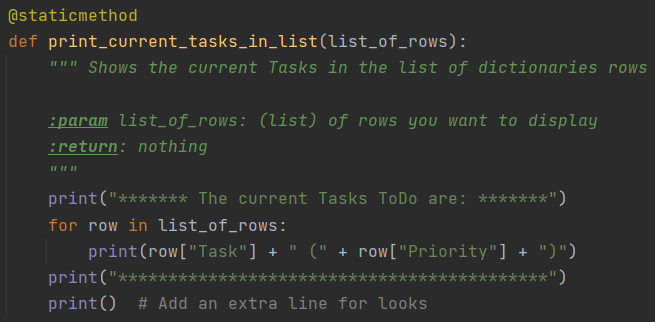
***Figure 8: The print menu task***

The next function is for accepting user input for their choice from the menu. The choice is returned by the function to the program (Figure 9).



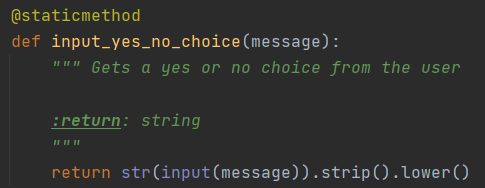
***Figure 9: The choice function***

The third function in this class prints the current list of data. This is not the list from the file as-is, rather it is the current list in our list table. This is the file data + - any changes the user has made. It has one parameter, the list table, and no returns (Figure 10).



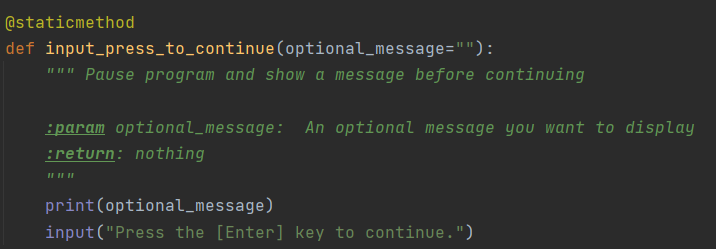
***Figure 10: The print current tasks list function***

The next function is pretty simple. It just takes the parameter of a message and uses it to create an input for the user that is then returned the program. For our purposes, we can use this for any time we need to receive a y/n response from the user (Figure 11).



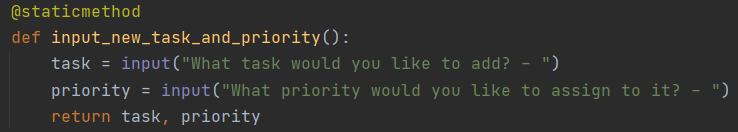
***Figure 11: Yes/No input function***

The fifth function in this class displays an input to the user, along with an optional message. The input is for the user to hit enter to continue (Figure 12). The optional message to the user can be something like a status, that is returned from one our processor functions. We’ll see this later in action.



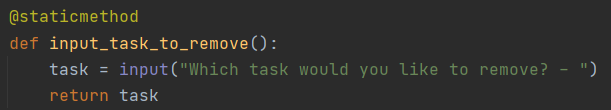
***Figure 12: The input press to continue function, with optional message***

This next function accepts user input for a new task and priority. It has no parameter, but returns the inputs from the user, to be used in one of the processing functions (Figure 13).



***Figure 13: The new task and priority input function***

The final function for the IO class asks the user for another input. This input is the task they would like removed. The input is returned as “task” and will be used as a parameter in another function (Figure 14).



***Figure 14: Task removal input function***

## The Main Body of Code

Now that all of our classes and their functions have been created, we can use them in our main body of code. Unlike last week’s main code that contained all of the processing and presentation code directly within it, this week’s code contains mostly the functions themselves. Last week’s main code is now contained within those functions, split apart by processing and presentation attributes.

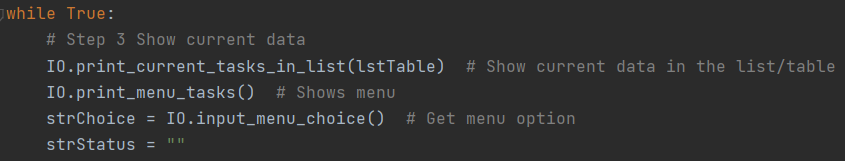
As the program begins, it calls our first function. We do this by first indicating the class it belongs to, followed by a period and the function name. This reads the data in the file, which is the basis for the rest of the program (Figure 15). Again, the inputs are the file name and the list table we are using to store the data. Both are variables we defined at the beginning of our script.



***Figure 15: The read file function being used***

With the data loaded into the list, we can begin our while loop (Figure 16). Before getting to the user options, we print the current tasks list. This is done by calling the IO function we created and providing the list table as the parameter. This is the same list table that was just updated by our read from data function.

We then print the menu and ask the user to input their choice. Both are done with the IO functions we created. The choice is stored in the *strChoice* variable, which is used in the conditional statements to follow. I set the *strStatus* to an empty string here so that it can be used in upcoming messages and be clear of any updates provided by our user options.

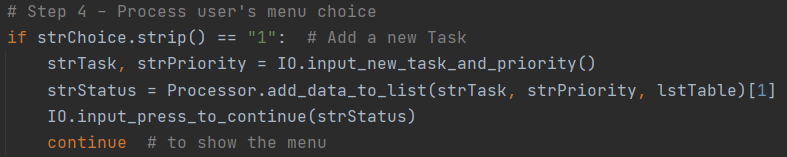


***Figure 16: The beginning of the while loop***

### Option 1

The first conditional statement checks if the user entered “1” (Figure 17). This is for entering a new task. The IO function is called to get the user inputs. As these are returned to the program, they are stored in two variables. They are then provided to the Processor add data function, along with list table we need updated.

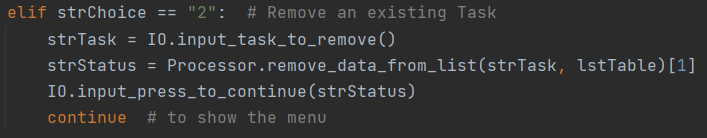
As this add-data-to-list function returns two values (the updated list and a message), we can store the message in the *strStatus* variable. This is done by indexing the returns, which are provided as a tuple. I don’t want to include the list as part of the message, so only ask for the message part of the tuple (the 1st index). This variable is then provided to the IO function for continuing, which allows for an optional message to be displayed. In this case the message that is returned from our Processor function.



***Figure 17: Option 1, allowing the user to add new data to the list***

### Option 2

If the user goes with option 2, to remove an existing task, this part of the code will run (Figure 18). Similar to the previous option, an IO function is provided to the user, with their input then being stored in a variable. The variable is provided to the Processor task, which removed an item from the list, along with the list to remove it from. We again store the message returned from the function by indexing it and provide it back to the user.

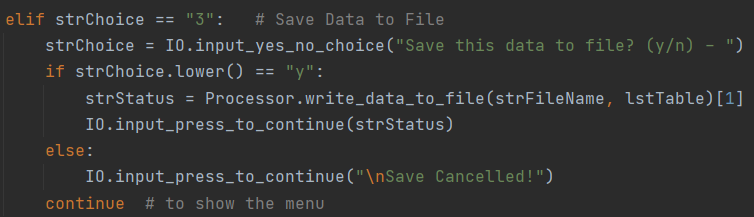


***Figure 18: Option 2, allowing the user to remove an item from the list***

### Option 3

To save data to the file, the user chooses option 3 (Figure 19). The first function called is the IO function for yes/no input. This function requires a message parameter to be provided, so we ask them if they’d like to save the data. Their answer is returned and stored in the variable. This allows the flexibility for them to cancel their save request.

If they choose “y”, then the program will call the Processor write data function. Just like our previous two functions, the message part of the return is stored in the status variable and displayed back to the user with the IO function for continuing. If they choose “n” then only the IO function is called, with the message hardcoded into the parameter.



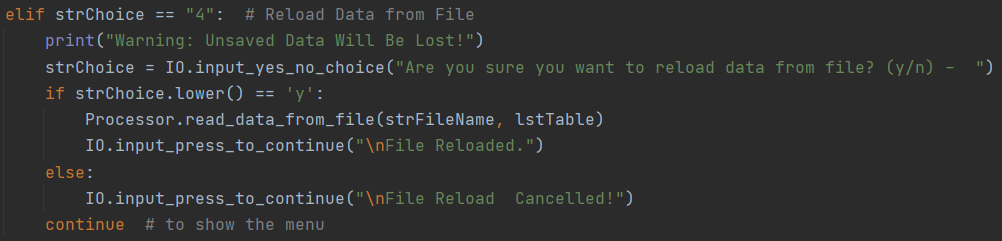
***Figure 19: Option 3, allowing the data save the list to the file. A cancel option is also provided.***

### Option 4

If the user decides that they want to reload the data from the file and ignore any changes that have not already been saved, they can do so by selecting option 4 (Figure 20).

Similar to the option 3, this gives the user a chance to cancel. A slightly different message is inputted into the parameter, but their input that is returned is stored the same way.

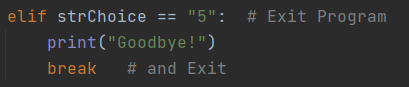
If they enter “y”, then the program will call the read data function. This resets the list table to what is currently in the file. Entering “n” will allow the user to cancel.



***Figure 20: Option 4 allows the user to reload the list from the file. A cancel option is provided.***

### Option 5

The exit option is simple (Figure 21). If the user selects option 5, then the program will break out of the loop and end. A good bye message is printed.



***Figure 21: Option 5 to exit the program.***

### No Option

I also included an else statement at the end of the while loop to catch any instance of a user entering an option other than 1 – 5 (Figure 22). If they do, the program will let them know and will restart the loop. This is done using the IO input function we’ve used previous that allows for option messaging.



***Figure 22: Statement to catch a user entering an invalid menu option***

## Summary

This week was all about classes and functions. The code within them is essentially the same as last week, but the processing and input/output steps of each section are broken out into their own relatable functions. This makes for cleaner and scalable code. As you can see, we reused some functions for more than one purpose and/or with different messaging.