**Introduction.**

In week 6 we do three labs that include working with functions, returning tuples, and creating classes. We practice doing these three methods by doing basic math problems. After we have a good understanding of the three methods, we move onto our assignment of creating a To Do list that allows the user to add, remove, save data, and exit the program.

***Week 6 Labs***

**Lab 6.1**

***Working with functions.***

Lab 6.1 gave me flash backs of assignment two, when we set up a script for solving basic math problems. Here we do the same thing but it’s entered into a string and is also used as floating data. You can see a sample of my code in figure 6.1 below.

A screenshot of a computer code

Description automatically generated

(Figure 6.1)

At first, I got the wrong answer for multiplying the values but quickly realized that I had the wrong symbol ( I had *value1 – value2*, instead of value1 \* value2). After an easy correction I was able to get the correct answers from my script in both IDE and OS. You can see my outcomes in figure 6.2.

A screenshot of a computer

Description automatically generated A black background with white text

Description automatically generated

(Figure 6.2)

**Lab 6.2**

***Returning Tuples.***

Lab 6.2 we return to using Tuples but with a twist. We make them “returning tuples” so we can pack and unpack them when needed. We achieve this by solving basic math problems and using arguments. To pack the Tuple, I had to create arguments, write the results of processing, and define *Calculate*. The sample for my code is in figure 6.3.

A screenshot of a computer code

Description automatically generated

(Figure 6.3)

Once my definition for calculate is set up I can pack my tuple.



Now that everything is defined, and the tuple is packed we are able to create the presentation for the user.

1. I created a couple of inputs for the user to select the two values.
2. I unpack the tuple by using calculate and my two arguments (value1 and value2).
3. Then simply have my code print the outcomes for each calculation. You might notice the %.2f , this is just a way to get your code to round up to two decimal places.

***A screenshot of a computer code

Description automatically generated  
(Figure 6.4)***

I was able to accomplish this script with relative ease, but it was good to compare lab 6.2 to 6.1 and see different methods to accomplish the same task. One using functions and one using tuples. You can see an example of my code in figure 6.4 above.

**Lab 6.3**

***Creating classes.***

Classes are simply a group of functions. Classes are great for simplifying your code and keeping your script neat. Once your script gets to crowded you are able to collapse the code that defines your functions under the processor, shown in figure 6.5.

A screenshot of a computer program

Description automatically generated

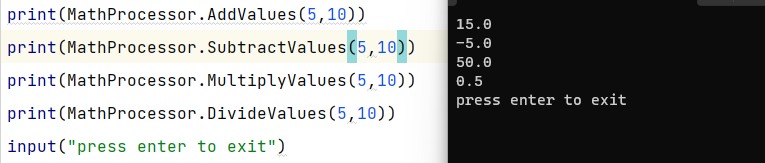
A close up of a code

Description automatically generated  
(Figure 6.5)

In lab 6.3 we continue to do some basic math but this time by using a class called math processor that have all the functions packed into the processor. Below is an example of my code with the functions defined. I was able to get the code to excute the functions in both the IDE and the command prompt, shown in figure 6.6 below.

A screenshot of a computer program

Description automatically generated



(Figure 6.6)

**Assignment 6**

Now that we are familiar with packing functions, creating arguments, and making classes. We are ready to put our skills to the test by doing a different task. Similar to assignment 5, we are creating a To Do list that allows the add, remove, and save task. As well as exiting the program. Easy enough to do since our instructor gave us a starter script that we had to simply add code where the script says *TODO: Add Code Here!* . We were also given an example for assignment five that already had the functions.

1. The first *TODO:* was under def add\_data\_to\_list. Shown in figure 6.7.

A computer screen shot of a program

Description automatically generated

(Figure 6.7)

I simply added list\_of\_rows.append(row)to be able to execute the function of adding a task with the priority into the list data. Shown below in figure 6.8.

A screenshot of a computer

Description automatically generated

(Figure 6.8)

1. The second *TODO:* was under def remove\_data\_from\_list

A screenshot of a computer

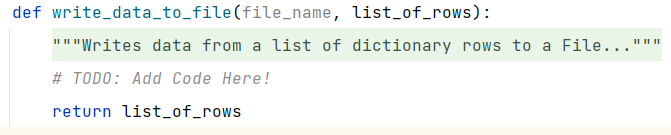
Description automatically generated

To get the script to remove the task, I wanted to create an updated list without the row that matches the specified task. Then create a returning tuple for the updated list. I was unable to get assignment 5 if statement to work here so I changed to a simple list code.

**A screen shot of a computer

Description automatically generated**

1. Third we are just making the data write to the txt file. The final *TODO:* under def write\_data\_to\_file. We have done this so many times, it was quite easy.



I took a piece of code from a prior assignment 5 and modified it to work on this script. Just had to put the file name for the txt file and put in the new string command. Shown in figure 6.9 below.

A screenshot of a computer program

Description automatically generated

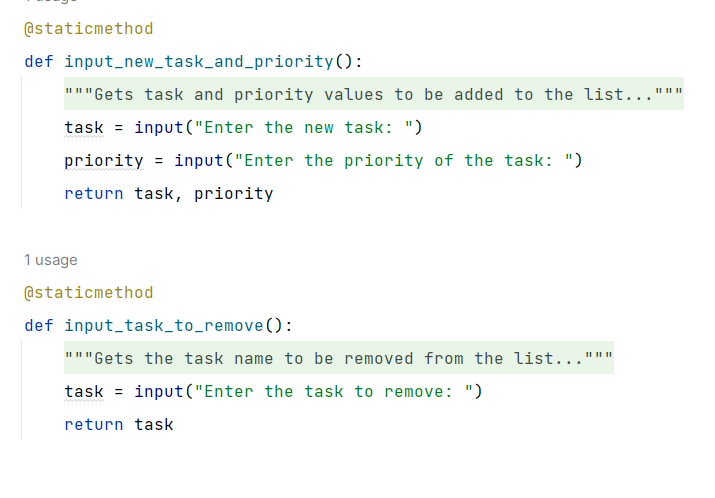
**(Figure 6.9)**

1. Now we will need to create class functions for the input/output tasks to complete the script.

**A screenshot of a computer code

Description automatically generated**

I just defined and created the inputs for the class processor to run. Once the user types in there task and priority, the script will then run the *if* statements to know how to continue. My functions for the input/output task are shown in figure 7.0 and my if statements are shown in figure 7.1.

****

(Figure 7.0)

A screenshot of a computer program

Description automatically generated

(Figure 7.1)

Now that we have the script written out, it’s time to test that it will run correctly.

Once I ran the program, I selected option 1) Add a new task.

Enter the new task: *Finish*

Enter the priority of the task: *High*

My program had no issue running the task, as you can see in figure 7.2

A screenshot of a computer program

Description automatically generated

(Figure 7.2)

Next, I selected option 2) Remove an existing task. Script ran without any issues and removed the task. Shown in figure 7.3, you can see that the “finish” task has been removed from the list and text.file.

A screenshot of a computer program

Description automatically generated

A screenshot of a computer screen

Description automatically generated

(Figure 7.3)

Next is to check option 3) Save Data to file. The information saved and wrote on the text.file. See figure 7.4 for example.

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

(Figure 7.4)

Final step was to make sure the script will run option 4) exit program and say goodbye to the user. Pretty self explanatory. See figure 7.5 below.

A white background with black text

Description automatically generated

(Figure 7.5)

Then run it again in the command prompt.

1.Add a task called finish. Figure 7.6

A screenshot of a computer program

Description automatically generated

(Figure7.6)

1. Remove the “finish” task from the to do list. Figure 7.7. You can see that the task has been removed from the list.

A screenshot of a computer program

Description automatically generated

(Figure 7.7)

1. Save the data. Figure 7.8

A screenshot of a computer program

Description automatically generated

(Figure 7.8)

1. Choosing option 4 automatically exits the script.

**Summary**

My script ran well but it took allot of trial and error before I could get it right and I’m sure there are things I couldve done better. It was good to learn how to organize my code in many different ways such as; creating arguments with functions, return tuples, and creating classes with a host of functions. It brought depth to my capabiilty as a coder. I can now attack a task with multiple approaches and decypher code much easier. I look forward to the many script I will apply these lessons to.