**Student Name:** Andy Simpson

**Date:** 2/22/20

**Course:** Foundations Of Programming: Python

**Assignment:** Assignment06

**The “To Do List – Part 2” Python Script Using Functions and Classes**

**Introduction**

In this module and assignment, I learned about creating Python scripts that classes and functions. As my scripts increase in complexity it becomes necessary to use functions and groups of functions called classes. These coding methods can break up big programs into smaller more manageable chunks of code. I found both functions and classes to be super useful. In this chapter I learned how to write my own functions, accept values into my functions as parameters, ability to return information from my functions through return values. I also learned to work with global variables and constants. Whereas lists work with sequences of information, dictionaries work with pairs of data. Dictionaries like real life dictionaries allow you to look up values using another value.

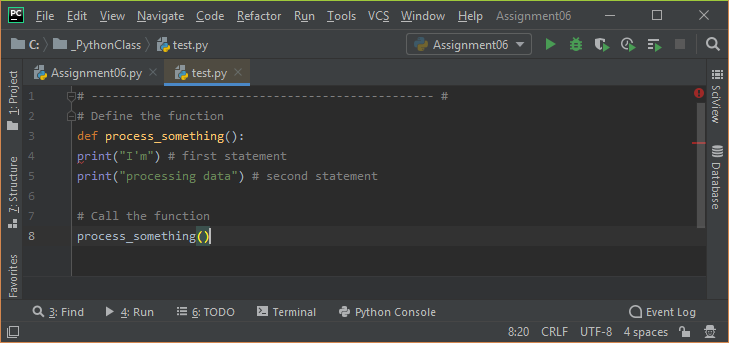
I also learned a bit about GitHub web pages, how to build simple classes and a little more about how to use the PyCharm Debugger.

**Understanding Abstraction:**

I learned on page 162 of Chapter 6, that by writing and calling functions, you practice what’s known as abstraction. Abstraction is a programming technique that lets you think about the big picture without worrying about the details.

**Functions:**

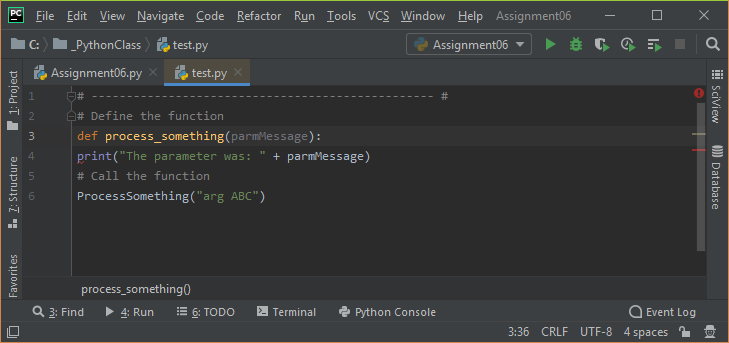
Functions are a way of grouping one or more statements. In Python, you must define a function before you can use code to call the function. Calling the function executes the statements in the function.



**Figure 1** *– Example of a simple function*

**Parameters:**

Optionally, functions can have parameters. These allow you to pass values into the function for processing. Values passed into parameters are officially called “arguments,” but it's common for people to call them parameters as too.



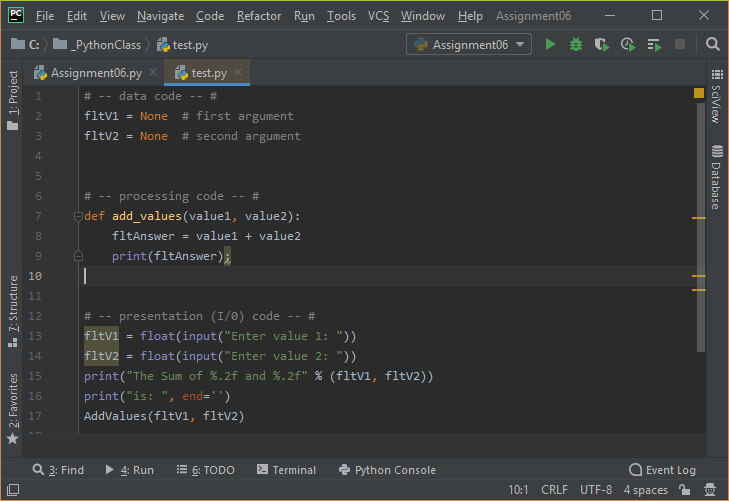
**Figure 2** *– Example of a function using an “Argument” and a “Parameter”*

There is no practical limit on how many parameters you can include. For example, you can easily pass two or more arguments to a function that calculates a sum.

***Note:*** *It is standard in the industry to parameter names without a prefix. As such, I leave them off in my function's parameters as well. I even use the common Python "snake casing" for some examples, just so you get used to seeing it.*

**Using Variables as Arguments:**

Use variables as arguments, which is a common pattern used in programming. Using argument variables is useful when you want access to these values multiple times in a script.



**Figure 3** *– Example of a function demonstrating the use of variables and Arguments*

**Return Values:**

Functions can return one or more values. You capture returning values of a function in variables.

**def MyFunction(): return 'data'**

**v1 = MyFunction()**

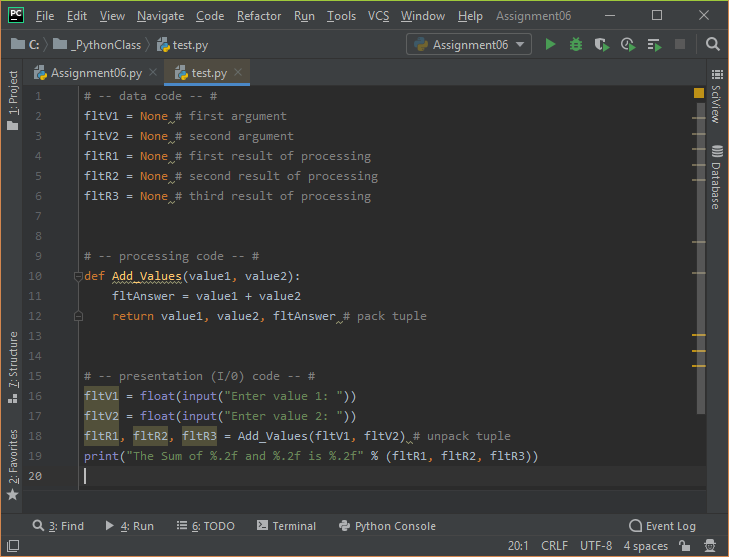
"Return values" make a function act as an expression. Evaluating a function as an expression means that you use the results of a function immediately without placing the result in a variable.

**print(MyFunction())**

Capturing the results in a variable allows you to use the variable of results multiple times without having to call the function again, but using it as an expression does not

***Important:*** *Notice that there is no presentation code in the function when using the return option! Functions with return values make code easier to divide into three layers of concern; data, processing, and presentation.*

Return values can be a single item of data or multiple items. If you return multiple values, you need to bundle them into a collection and return that collection. In Python, simplify this process using the tuple packing and unpacking feature (Listing 6).

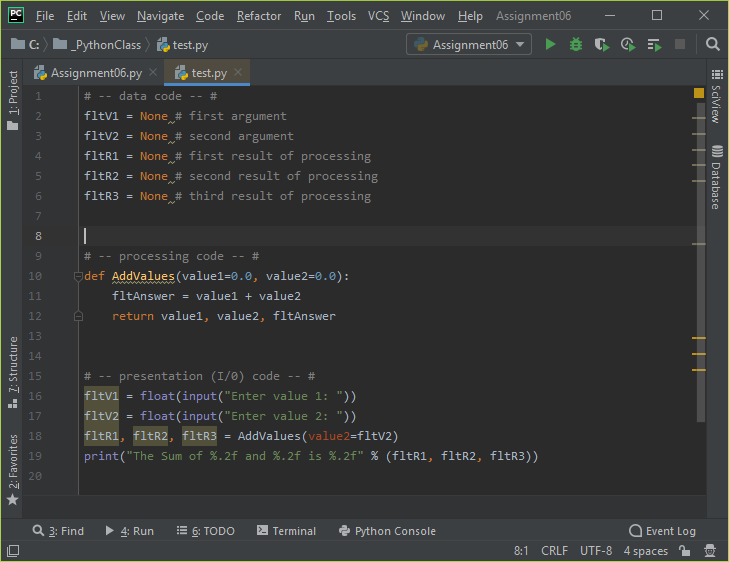


**Figure 4** *– Example of a function demonstrating the unpacking of features*

The packing and unpacking feature only applies when using Python tuples, but other languages do not let have this option. Instead, you would use code similar to Listing 7, which shows an example of using a List object instead of a tuple while still producing the same results.

**Default Parameter Values:**

You can set default values for a parameter. When you do, not supplying an argument for the parameter forces the function to use the parameter's default value



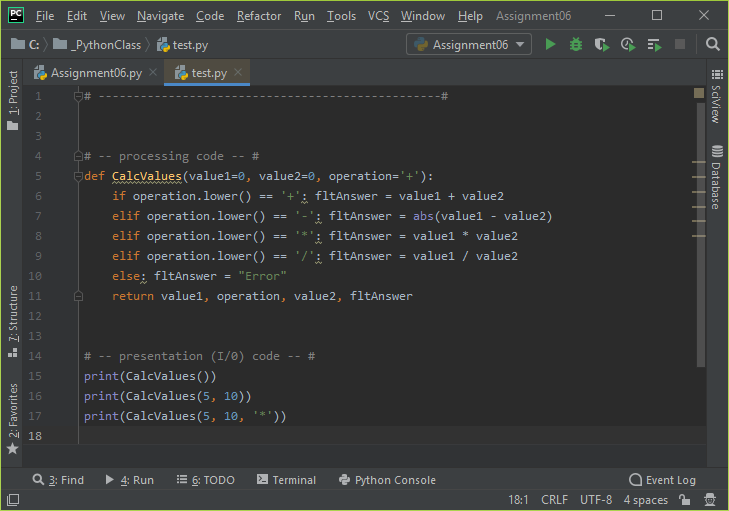
**Figure 5** *– Example of a function demonstrating the usage of Default Values*

***Note:*** *Notice how even though I asked the user to enter "value 1," it was never used, and as such, the script does not display what the user expects!*

**Overloaded Functions:**

Many modern languages allow you to create multiple versions of a function. Each version uses a different number of parameters or parameters with different data types.

Python cannot distinguish between versions based on data types due to its automatic data typing feature, nor does it allow you to have two functions with the same name. Instead, Python uses default values to accomplish something similar

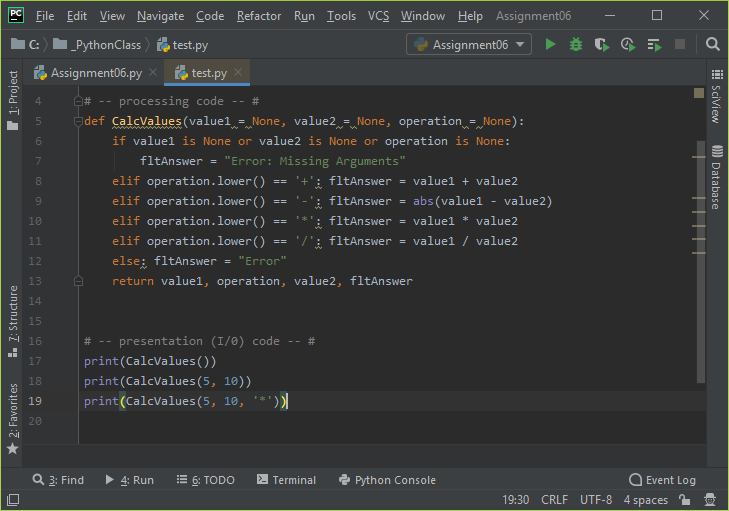


**Figure 6** *– Example demonstrating the usage of Overloaded Functions*

**The “None” Keyword:**

In Python, "None" is a special data type whose only value is "None." None is frequently used to indicate the absence of a parameter value.

You often see None used as a default argument. When you do check for the default by using the "is" operator instead of the "==" operator.



**Figure 7** *– Example demonstrating the usage of the “None” Keyword*

**Returning Data by Reference:**

In Python, some variables act like "value types" and others as "reference types." The code in listing 12 was discussed in module 02. Review it and see if it makes more sense to you at this point in the course.

In many modern languages, you can choose to pass arguments into a function as a reference to an address in memory or a value. You would indicate your choice using keywords next to the parameter's name.

**MyFunction(p1:float by val, p2:float by ref):**

**return 'data'**

Python automates this functionality based on data types. Simple arguments like Strings, Integers, and Floats pass to a function as a value. Complex arguments like Lists and Dictionaries pass to a function as a reference. Oddly, Tuples, although complex, also pass to a function as a value.

**Global vs. Local Variables:**

Variables in a script may be local or global. Variables declared in a function are considered local to the containing function and cannot be accessed outside of that function. Variables declared in a "body" of the script are considered global to the containing script and can be used anywhere in the script.

Any code inside of the same function can "see" the local variable because that variable would be "inside of its scope!".

def MyFunction(): v1 = 15 *# local* print(v1) *# This works!*

Any code outside of the function cannot "see" the local variable because that variable would be "outside of its scope!"

def MyFunction():

v1 = 15 *# local*

print(v1) *# This works!*

MyFunction()

print(v1) *# This causes a "NameError: name 'v1' is not defined"*

Some developers use global variables to access values inside of a function. Listing 14 shows how a global variable could be used to access function results alongside a local variable being used to return results. Using the return option is preferred. Note that the code uses the keyword "global" to indicate a variable is global since the default is local.

**Shadowing a Global Variable:**

If you do use global variables within a function, be careful to use the keyword "global," or your local variable will "shadow" the global one whenever you assign a value to a variable with the same name!

Conveniently, when you read a variable in a function with the same name as a global variable, the shadowing does not happen automatically.

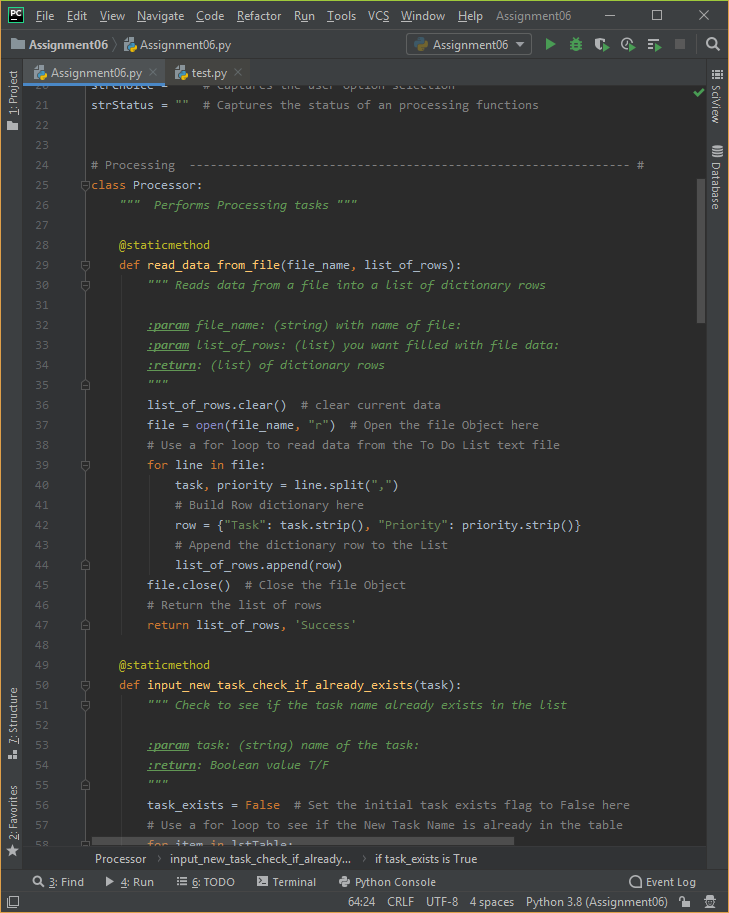
Using global variables within functions is discouraged in programming since it breaks the concept of "Encapsulation/Abstraction," which we look at in module 8.

***Note:*** *Unlike global variable which could appear in many of the hundreds of line in the main body of the script, it is common to name local variables without a prefix since these variables are easier to locate within the function. This also helps you not shadow a global variable by mistake!*

**Function Document Headers (Doc Strings):**

It is a common practice to include a header at the beginning of a function, which is known as *docstring* in python. It can be helpful when developers include additional notes in a docstring. When they do integrated development environments like PyCharm can display tooltips to show you a developer's notes (use ctrl + q to activate this option in PyCharm).

***Note****: PyCharm includes some basic tooltip text even without a docstring.*

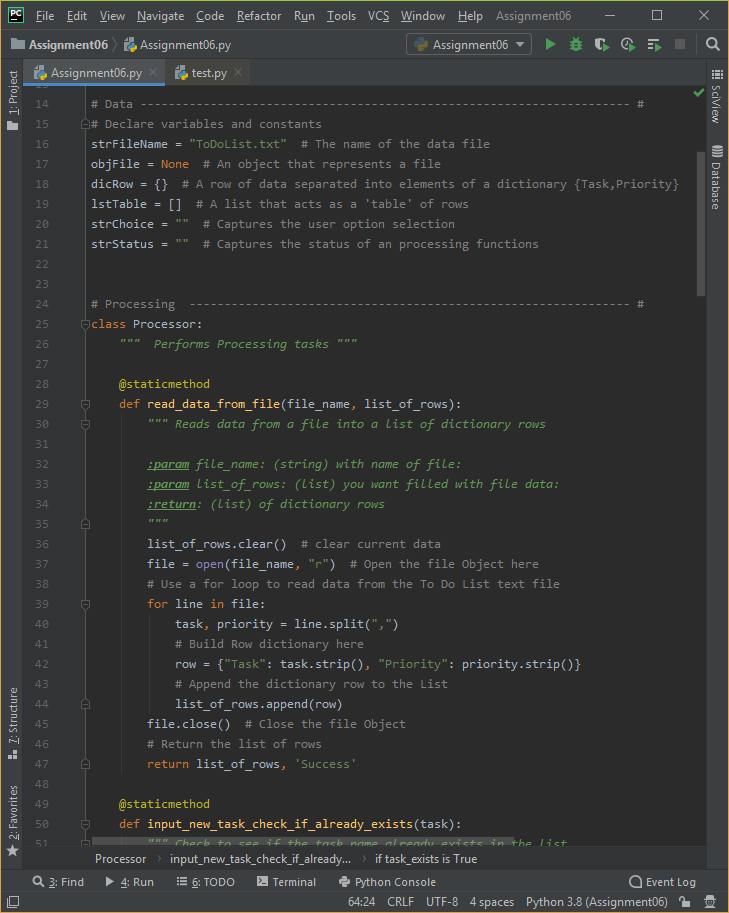


**Figure 8** *– Example demonstrating the usage of docstrings*

**Classes and Functions:**

Classes are a way of grouping functions, variables, and constants. We cover more about classes and why you might type the "@staticmethod" directive before the of the function's name in module 08.

Dictionary keys are a lot like columns in a spreadsheet or database. As such, it is helpful think for a dictionary as a row of data. These “rows” can be added to a List to form a collection of rows, which creates a table like two-dimensional collection of data.



**Figure 8** *– Example of functions within a Class*

**Using the PyCharm Debugger:**

Separating code into functions and classes make your code easier to read and to debug since smaller sets of statement are easier to test. Still, developers have found that using an Integrated Development Environment (IDE) debugging tool can make finding and fixing bugs much faster.

It is easy to start using PyCharm's debugging tools, even though the IDE includes many advanced options that can be confusing for new developers.

**Debug Mode:**

Frequently I run my code in PyCharm using the "Run" option from the context menu. You access this menu by right-clicking any location within the script file.

Notice that there is a "Debug" option in the same context menu. Using this option runs the code using the PyCharm debugging tools



**Figure 9** *– PyCharm Debugging Options*

When the code starts running you will notice that the results window looks different in run mode than it does in debug mode. In debug mode you can switch between the console display or a set of debugging windows using the Debugger | > Console tab.

**Breakpoints:**

To use the set of Debugging windows you need to set a "breakpoint" in your code. A breakpoint tells the debugger when to pause running the code so that you can exam the information in the debugging windows.

The next time you run your code in debug mode, the code pauses and lets you use the controls in debugging windows. Here is a description of the important controls:

1. The tab that lets you navigate between the debugging and console windows

2. The "Variable" window that shows which variables are being used and their values

3. The "Step Over" button that allows you to skip over seeing the code in a function when it is called.

4. The "Step Into My Code" button that allows you to step into the next line of your code, without showing you lines that the Python runtime user to run you code (Which can be confusing!)

5. The "Step Out" button that allows you to stop showing the code in a called function and return to the line of code that called it.

6. The "Stop" button that allows you to stop debugging and running the script.

**Walking Through Code:**

The control you use the most is the "Step Into My Code" button. Using it, allows you to "walk" through each line of code that performs an actions. Code that does not perform actions, such as comments are skipped.

When a line of code is reached that contains variables, you can either hover over that variable to see what is current held in memory or look at the "Variables" window

Note that both the value and the data type are shown in PyCharm. This is useful since Python automatically select a variables datatype for you, which can cause some hard to find bugs!

***Important:*** *The only way to learn how to use the debugging tools is to just start using them! So, try to use the debugger at least once for each assignment going forward!*

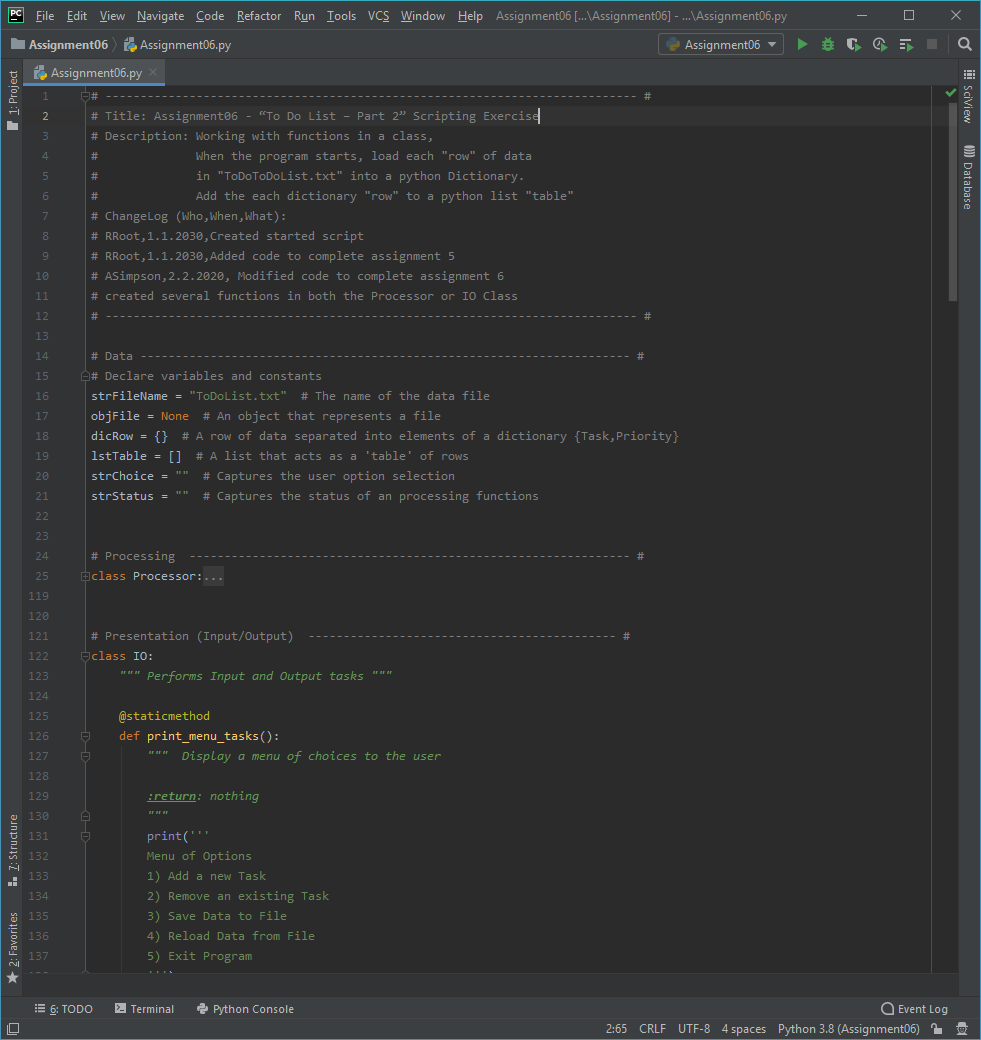
Once a template is created, you use it by right-clicking your project icon in the Project Explorer tree and using the New > “Name of your Template” option in the context menu as shown in figure 8.

**GitHub:**

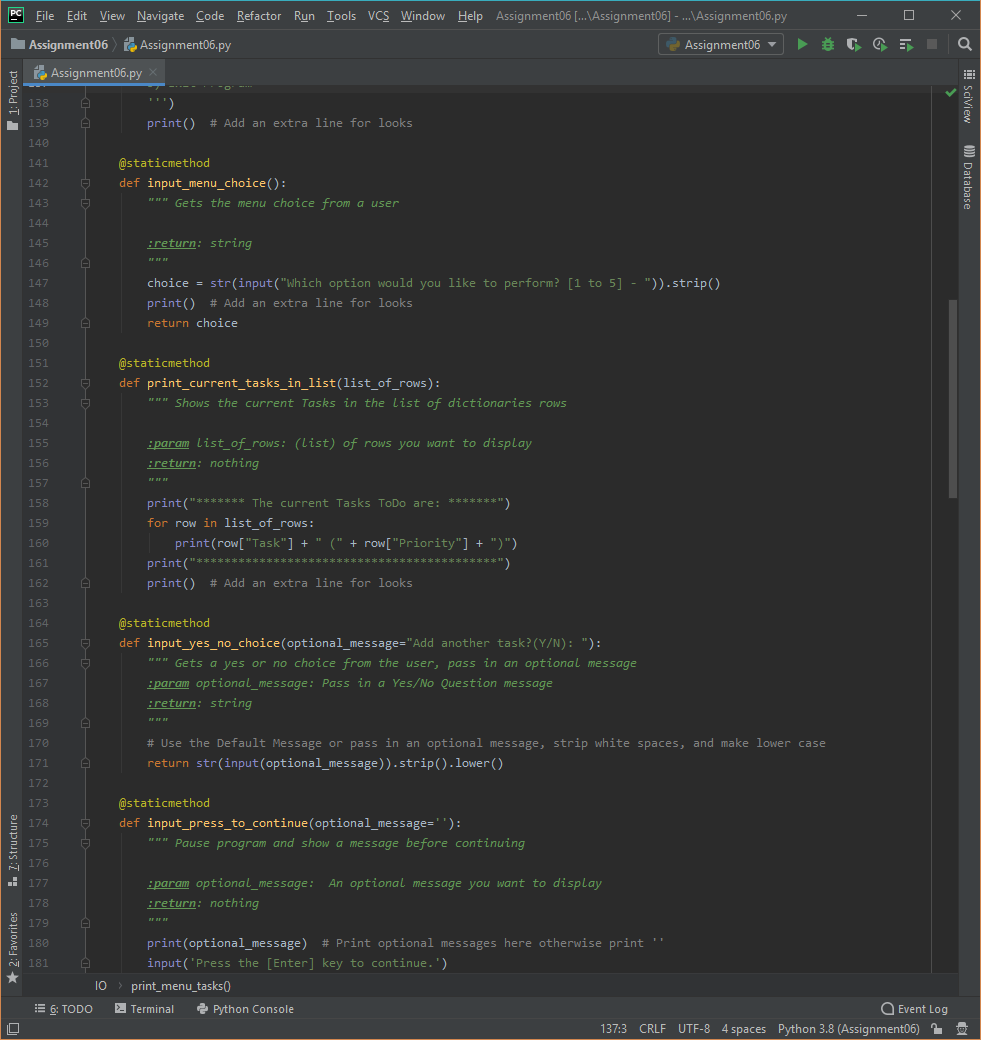
It has always been a good idea to make backups of your code files and to make them available for others to access. Traditional this has been accomplished using a network share on an organization’s server. While this is still a common practice, more and more organizations have embraced storing this code on the Internet via source control software.

**Assignment06 – The “To Do Task List – Part 2” Script:**

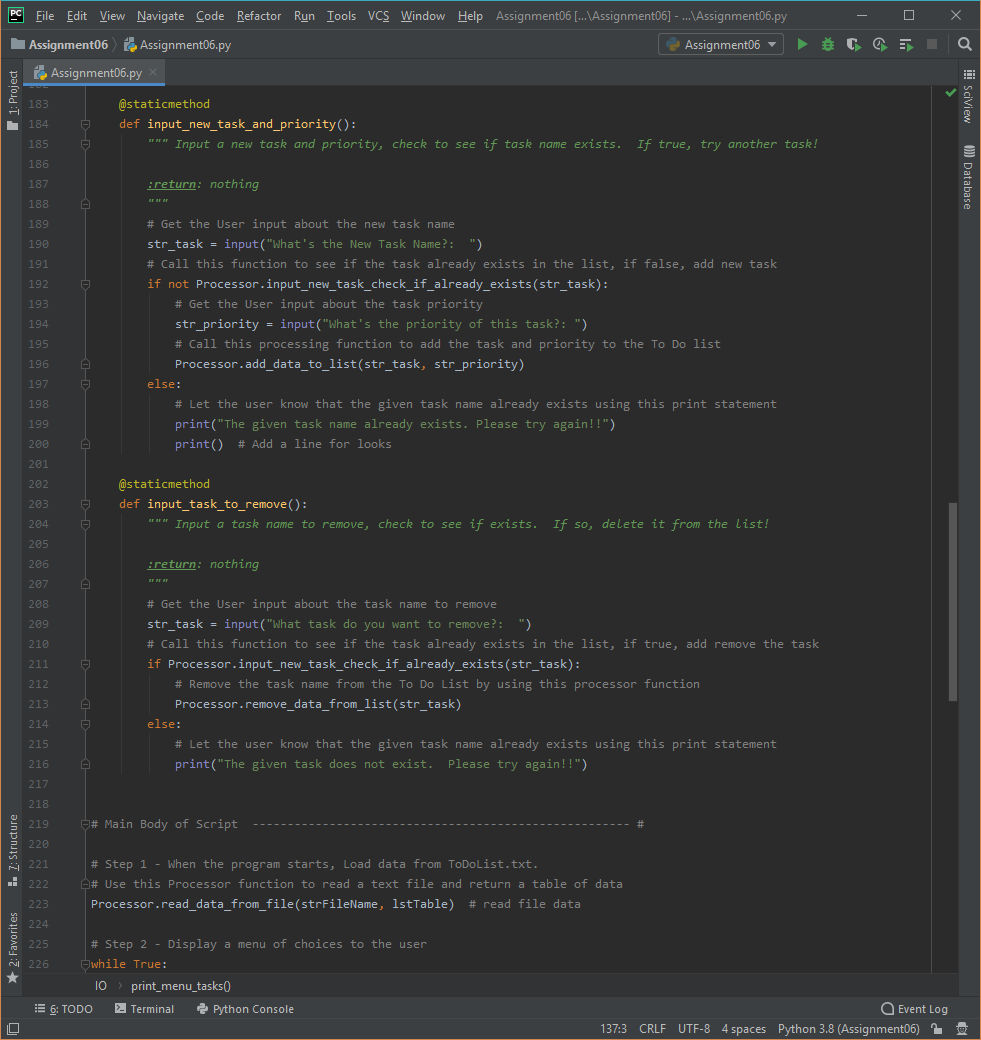
Figures 10, 11, 12, 13 and 14 show the code from the “To Do” Task List Python Script. Figures 15, 16, 17, 18 and 19 show this script running from within the PyCharm IDE. And then Figures 20-24 show the script running from the Command Console. Finally, I included the text file output as Figure 25 for your review.



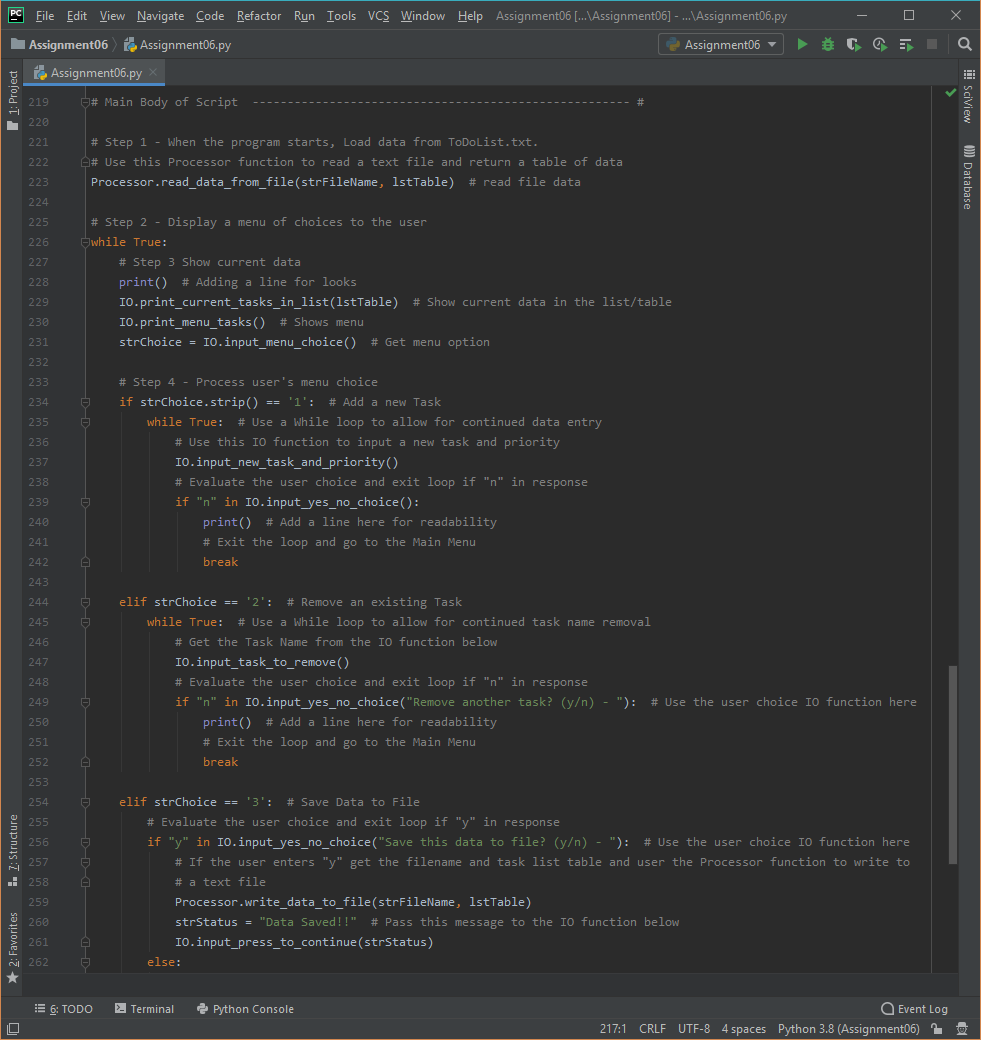
***Figure******10*** *– “To Do List – Part 2” Python Script (Assignment 6) using PyCharm IDE*



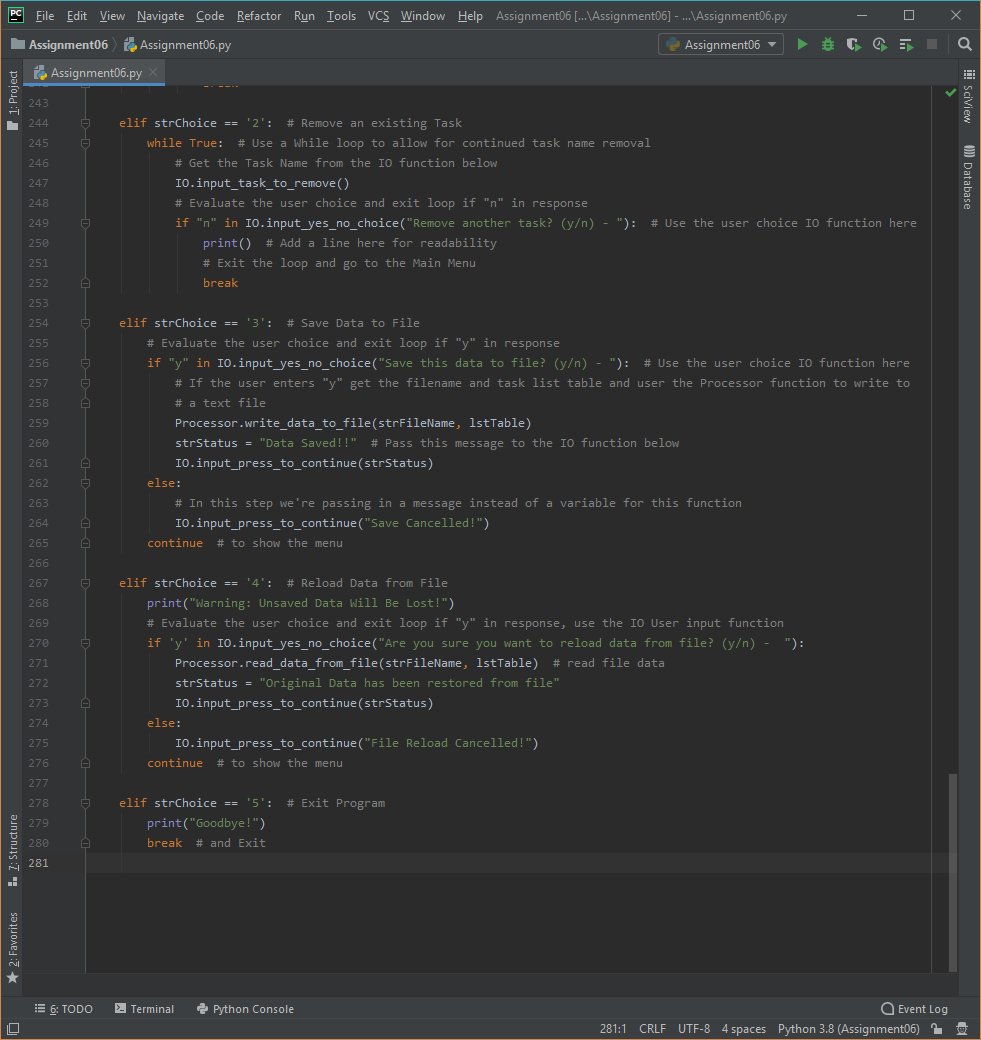
***Figure******11*** *– “To Do List – Part 2” Python Script (Assignment 6) PyCharm IDE - Continued*



***Figure******12*** *– “To Do List – Part 2” Python Script (Assignment 6) using PyCharm IDE – Continued*

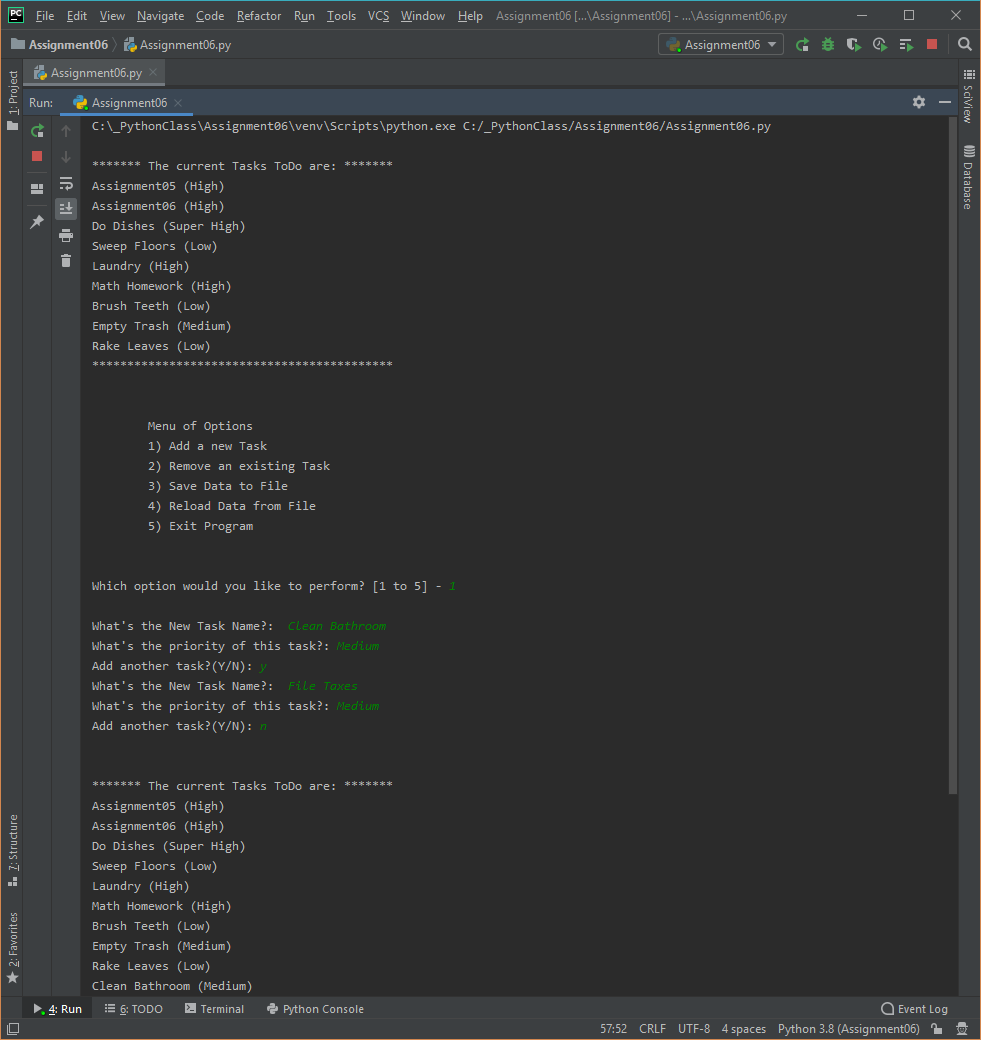


***Figure******13*** *– “To Do List – Part 2” Python Script (Assignment 6) using PyCharm IDE – Continued*

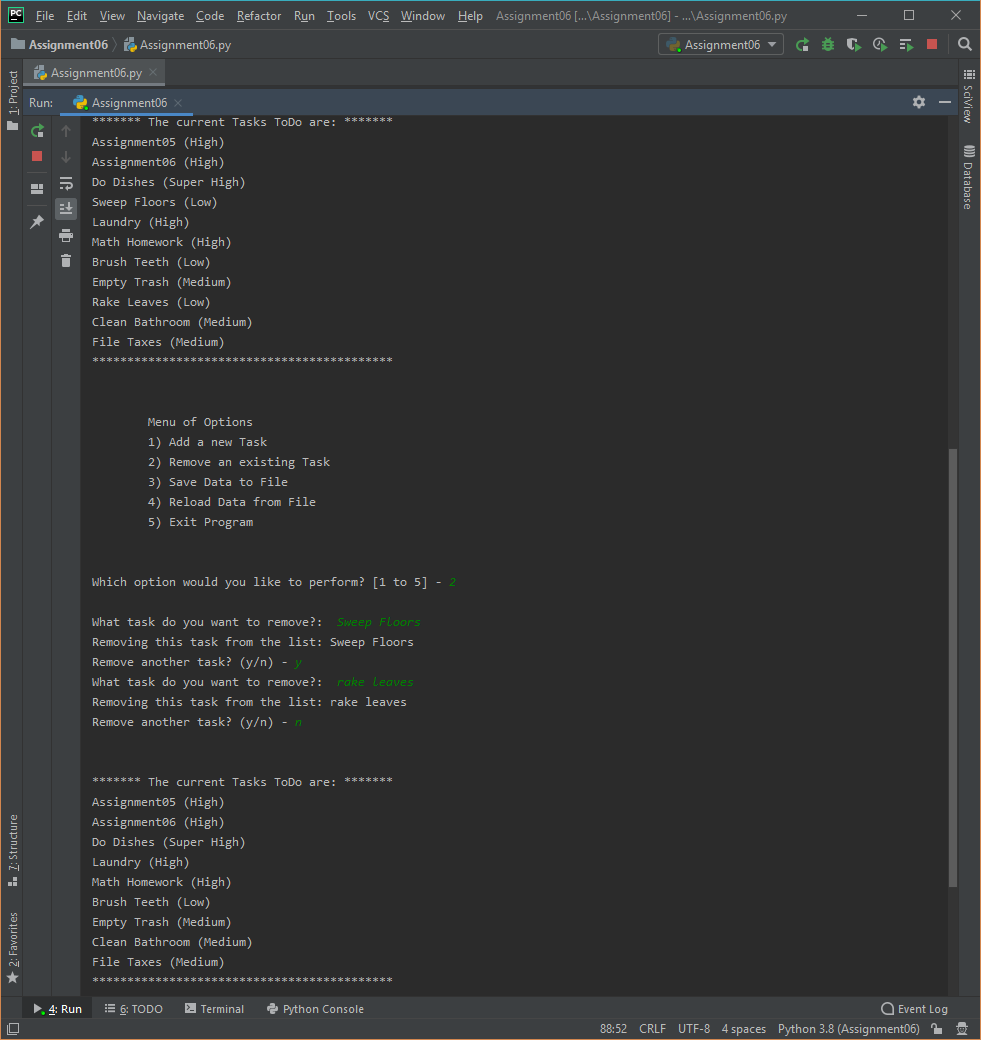


***Figure******14*** *– “To Do List – Part 2” Python Script (Assignment 6) using PyCharm IDE - Continued*

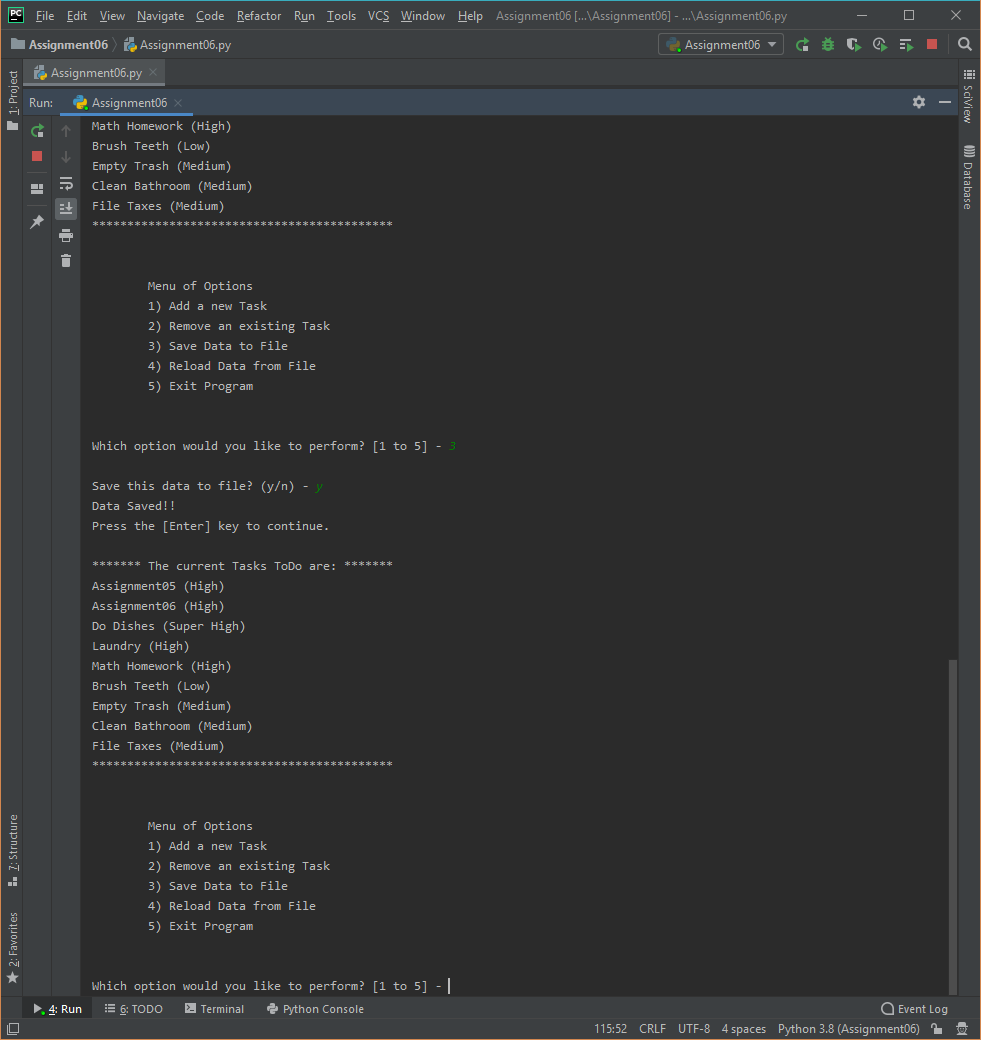
The following screenshots, Figures 14, 15 and 16 shows the Assignment 4 “Home Inventory” script running from within the PyCharm IDE. Please note that I added an introduction to the script as well as a message informing the user that the script has closed.



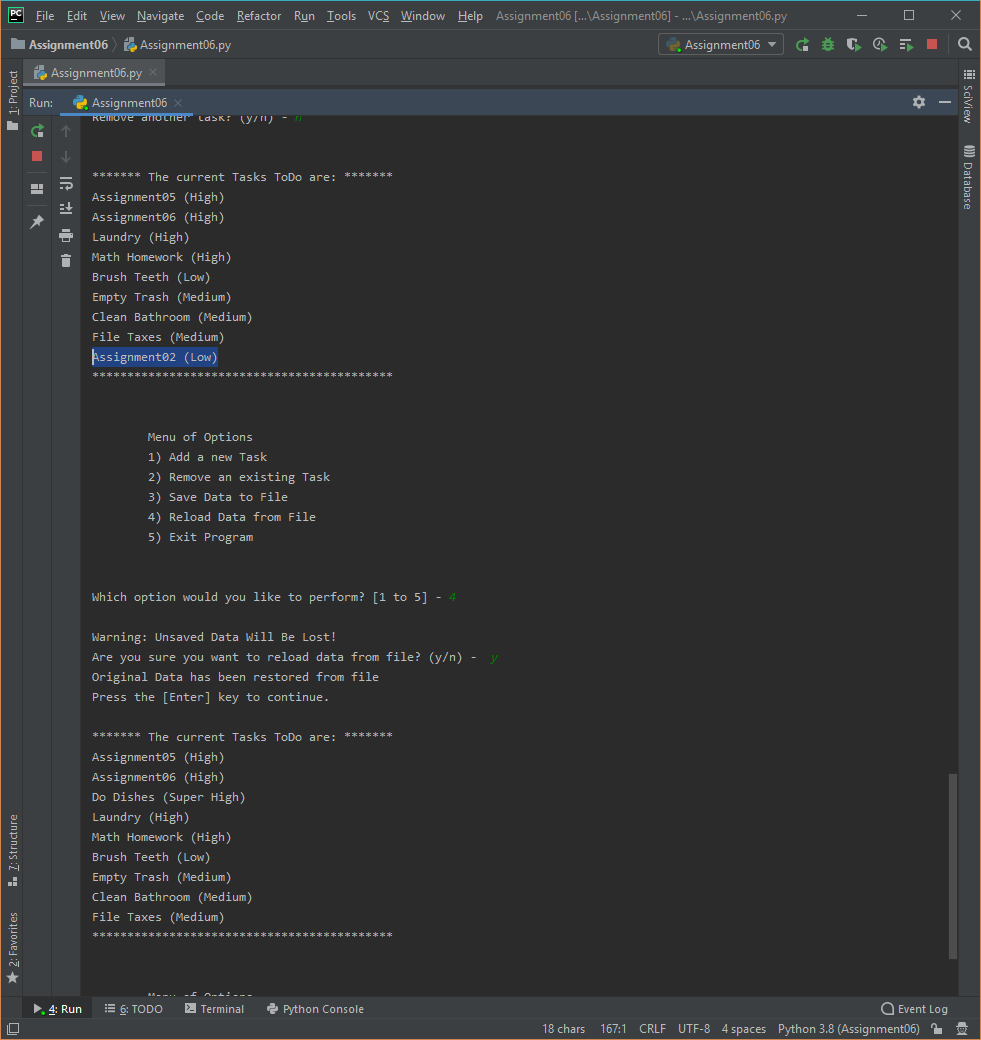
***Figure******15*** *– Running Option #1 from the “To Do List” Python Script using PyCharm IDE (Continued)*



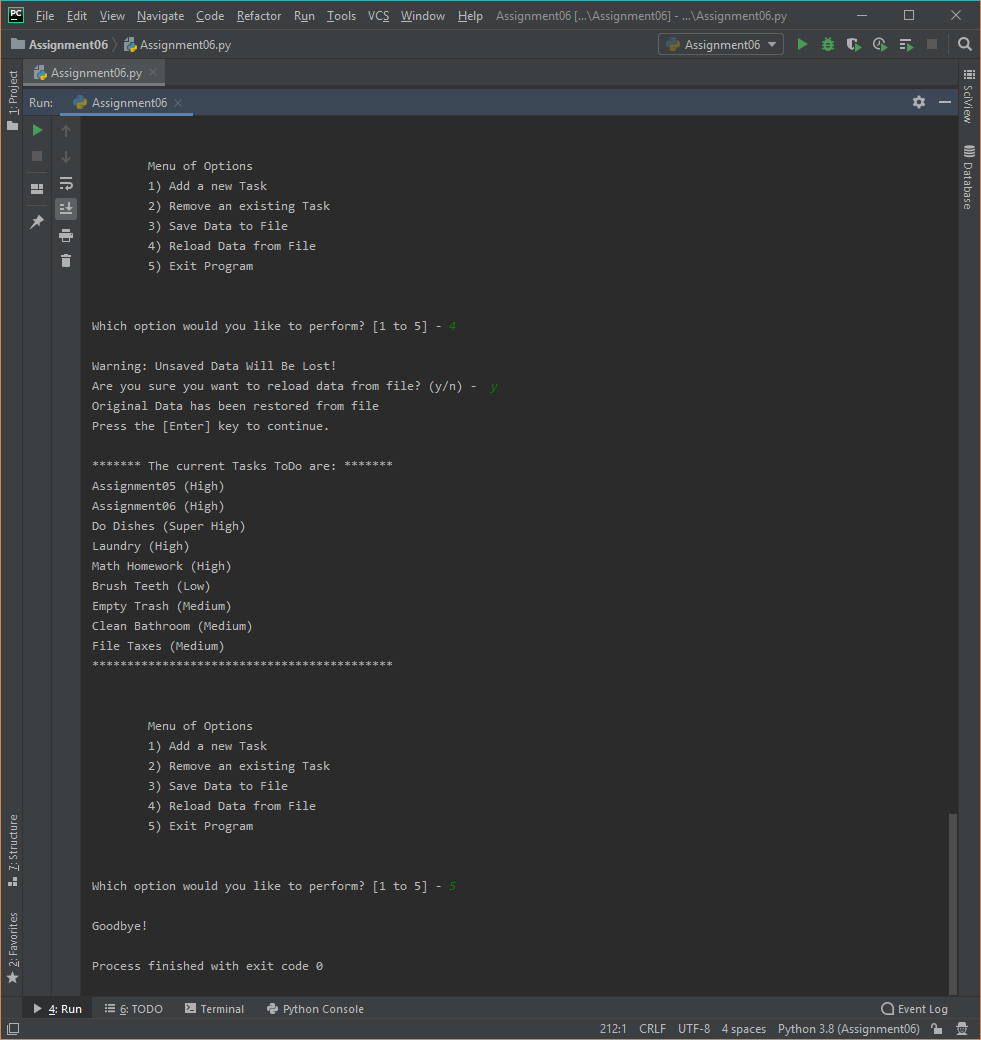
***Figure******16*** *– Running Option #2 from the “To Do List” Python Script using PyCharm IDE (Continued)*



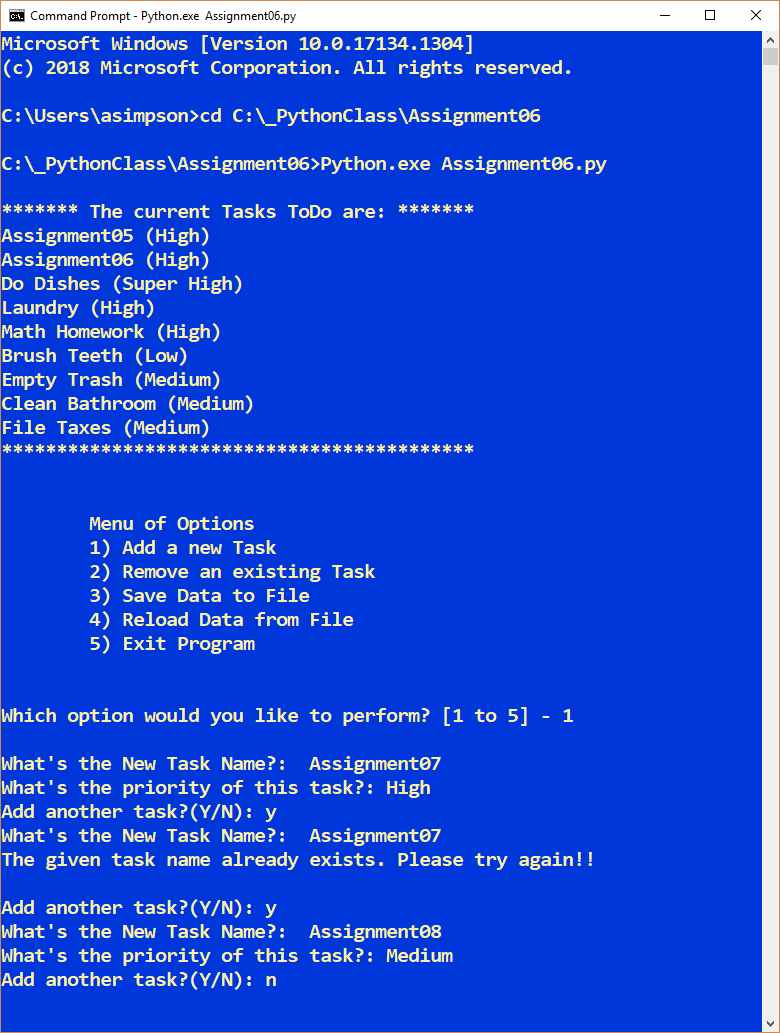
***Figure******17*** *– Running Option #3 from the “To Do List” Python Script using PyCharm IDE (Continued)*



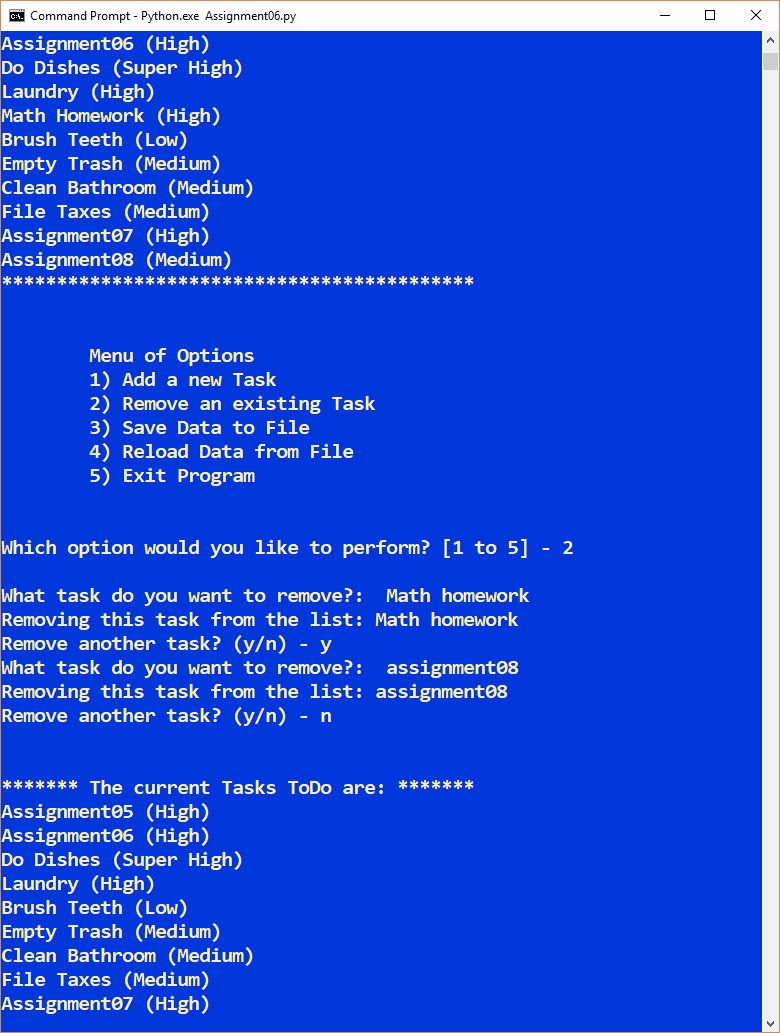
***Figure******18*** *– Running Option #4 from the “To Do List” Python Script using PyCharm IDE (Continued)*



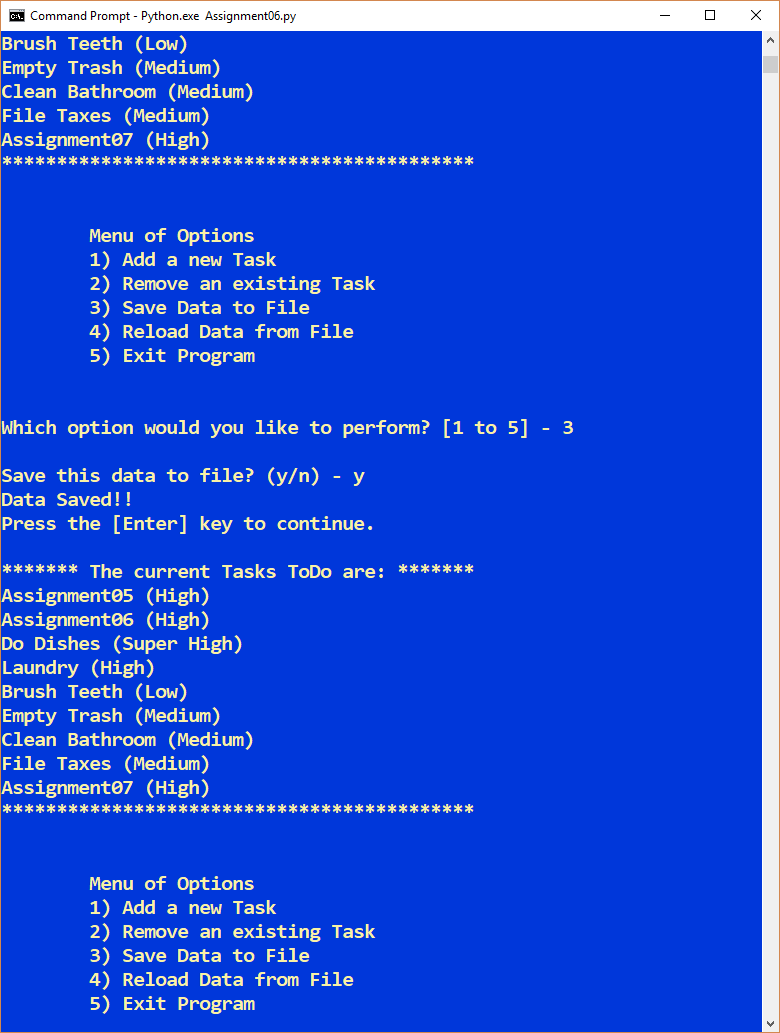
***Figure******19*** *– Running Option #5 from the “To Do List” Python Script using PyCharm IDE (Continued)*



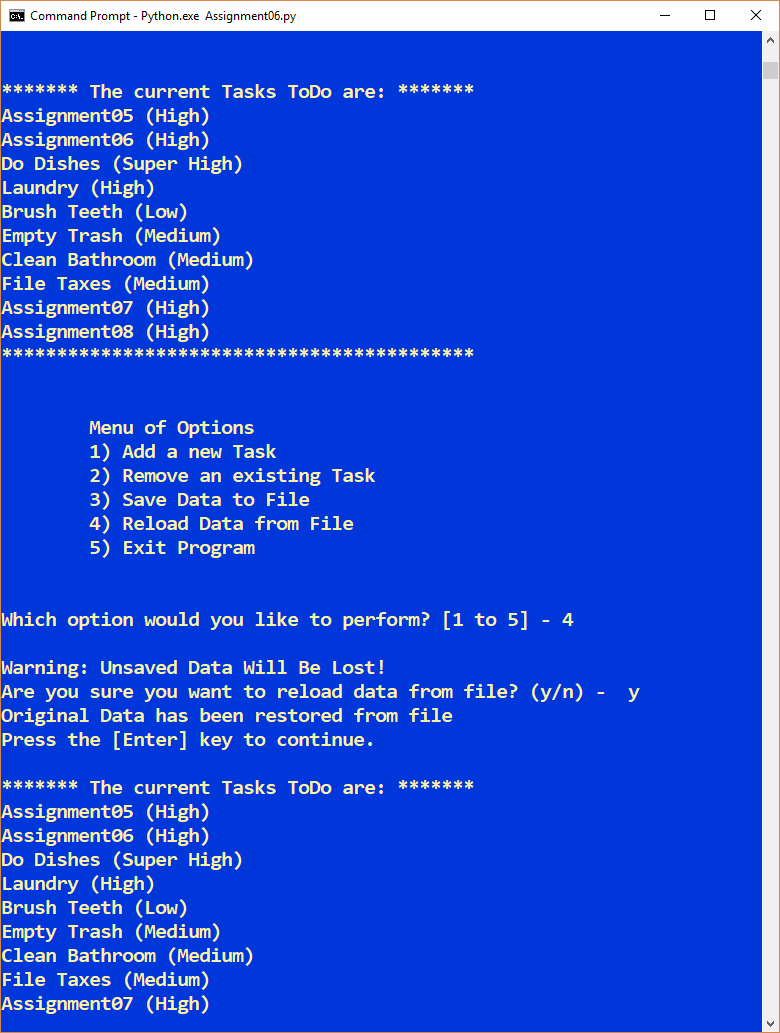
***Figure******20*** *– Option #1 Add a new task, the “To Do List – Part 2” Python Script running from the Command Console or Command Prompt*



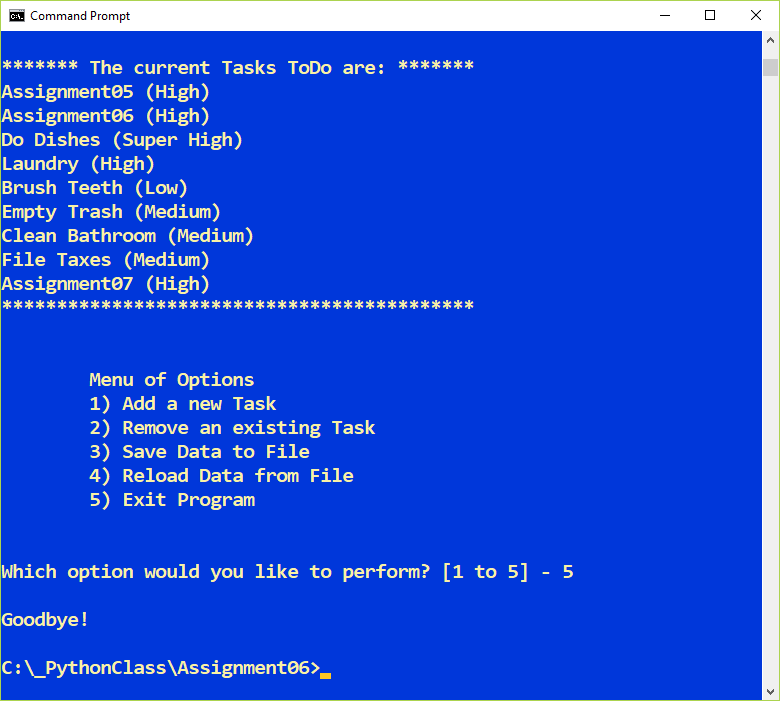
***Figure******21*** *– Option #2 Remove a task “To Do List – Part 2” Python Script running from the Command Console or Command Prompt*



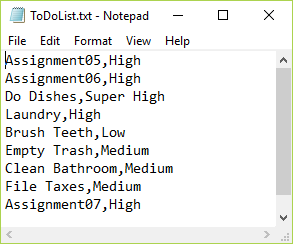
***Figure******22*** *– Option #3, save to a text file of the “To Do List - Part 2” Python Script running from the Command Console or Command Prompt*



***Figure******23*** *– Option #4, Reload Data, Don’t Save Changes to a text file of the “To Do List - Part 2” Python Script running from the Command Console or Command Prompt*



***Figure******24*** *– Option #5, Exit the Program, of the “To Do List - Part 2” Python Script running from the Command Console or Command Prompt*



***Figure******23*** *– The “To Do List” text file of user input values as written by the “To Do List” Script*

**Summary**

In summary, as part of module 6, I learned about how to use functions and classes. I also learned to write my own functions. I also learned how to accept and return values with functions I created. I also learned about scopes and saw how global variables can be accessed and changed from within functions. I also learned to limit my usage of global variables within functions.

In assignment 6, I used functions to add new tasks to a “To Do” list and then remove existing tasks from that list. As shown in Figure 23, I’ve successfully wrote data to a text file and later removed a value all using functions. Even though it wasn’t part of the homework, I wanted to add small enhancements like checking for existing task names before removing or adding a given task. I also added some functionality to compare the values equally in case the user miss-typed a given task name.

All of these “abstraction” programming concepts will help me improve my scripting by staying organized. Finally, I put many of the concepts mentioned above into the “To Do List – Part 2” python script. I’m ready to take on chapter 7 where we’ll learn about files and exceptions.