**Student Name:** Andy Simpson

**Date:** 3/12/20

**Course:** Foundations Of Programming: Python

**Assignment:** Assignment08

**GitHub Link:** <https://github.com/Asimps2006/IntroToProg-Python-Mod08>

**The “Product Price Tracker Program” Python Script**

**Introduction**

In this module and assignment, I learned about Object-Oriented programming (OOP) which is a different way of thinking about programming. It’s a modern methodology that’s been embraced by the software industry and is used in the creation of the majority of new commercial software. The basic building block in OOP is the software object – often just called an “object”.

Specifically, I learned about:

* How to create classes to define objects
* How to write methods and create attributes for objects
* How to instantiate objects form classes
* How to restrict access to an object’s attributes

I also learned a bit about GitHub Desktop, how to use this tool on my laptop to create repositories and publish them to GitHub.

**Classes:**

Classes are a way of grouping data and functions, and most classes are designed to focus on either data or processing. For example, a developer might create one class for processing data to and from a file, called something like "File Manager," and another for managing the data to be processed called something like "Customer." The focus of the "File Manager" class would be to perform a set of actions, while the focus of the "Customer" class would be to organize data about a customer.

Data in a class is defined using variables or constants. However, when these are in a class, they are called Fields. Any functions you have in a class are called Methods.

**Objects vs. Classes:**

When the class's code loads into memory, you either use that code directly or indirectly. To use the class's code directly, you use commands like the following pseudo-code:

Customer.Id = 100

Customer.Name = "Bob Smith"

To use the code indirectly, you create an object instance of the class and use the object's variable with commands like the following pseudo-code:

objC = Customer()

objC.Id = 100

objC.Name = "Bob Smith"

One advantage of using the code indirectly is that you can have multiple object instances, each with a different address in memory. The data for each instance is kept separate for each object, and each object would hold data about a different customer.

In general, you use a class directly if its focus is on processing data and indirectly if its focus is on storing data. This generalization may not always be true, but often enough that it provides a good starting point.

**Standard Class Pattern:**

Classes typically have Fields, Constructors, Properties, and Methods. Like scripts, class code follows a general design pattern in most of the languages. Here is a pseudo-code example the different area that make-up a Python class:

class MyClassName(MyBaseClassName):

# -- Fields –

# -- Constructor –

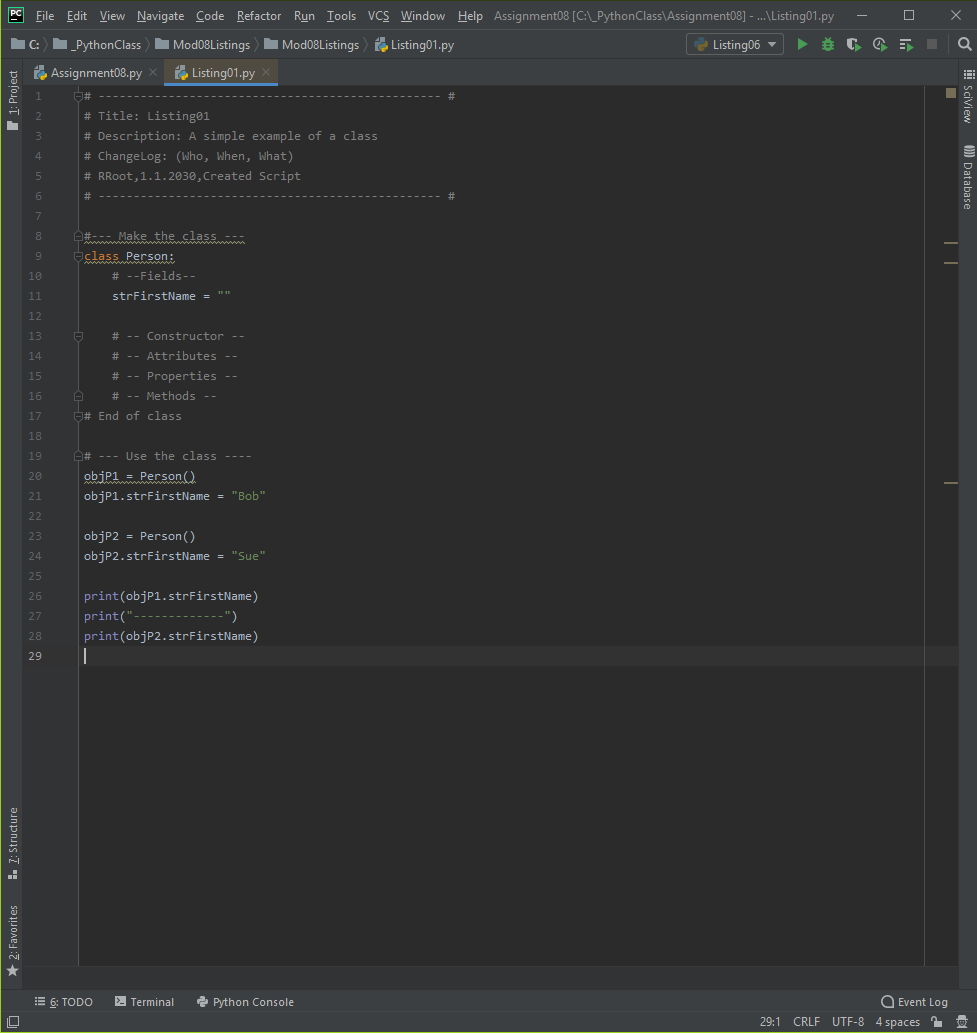
# -- Attributes –

# -- Properties –

# -- Methods --

**Fields:**

Fields are the data members of a class. Fields are created using variables and constants. Listing 1 shows an example of creating two object instances using a class called Person. Each object instance can hold first name data unique to each person.



**Figure 1** *– Example of how different elements of a class are used (from Assignment08)*

**Constructors:**

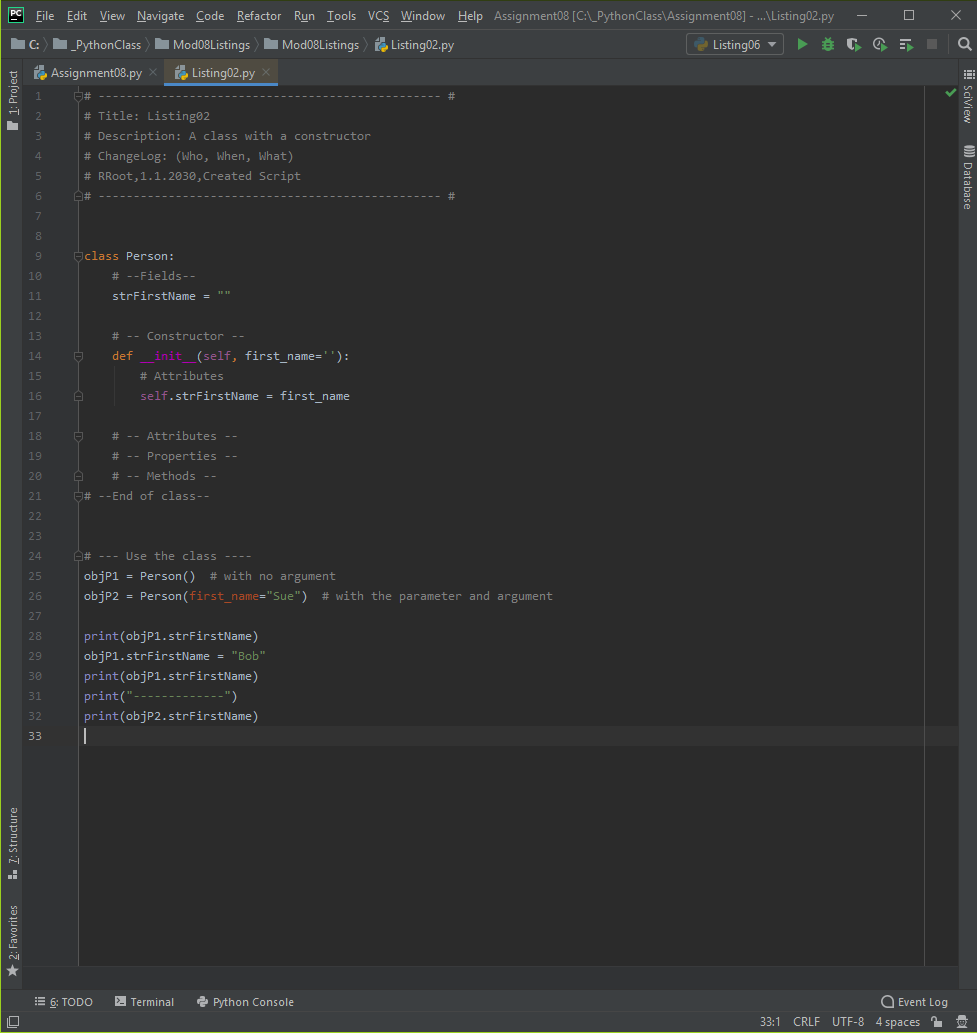
Constructors are special methods (functions) that automatically runs when you create an object from the class. Constructors are often used to set the initial values of Field data.

Python Constructor's use the double underscore ("duder") name of "\_\_init\_\_" but in many languages, it is the same name as the class.

When you create an object instance from a class, you use the class's name as if it were a function.

objP1 = Person("Bob")

Python automatically calls the "\_\_init\_\_()" method and passes any arguments you provide to the "\_\_init\_\_()" method each time you make a new object.



**Figure 2** *– Example of how to use Constructors (from Module08 Notes)*

***Note:*** *Since constructors are a specialized function, so you use them as a function by passing arguments into the parameters. However, remember, they only run once; when a new object instance of a class is created!*

**Destructors:**

Another special method is the "Destructor." These automatically run when an object instance goes is removed from memory. They are used to "clean up" any resources that are not needed once the object is gone. In Python, most of the resources are "self-cleaning," and so you do not often see these in classes like you do Constructors.

Destructors are considered an advanced feature and should be used with care. We do not go into them in this course, but their code looks like this.

**The “Self” Keyword:**

You probably noticed the use of the keyword "self" in the constructor method. This keyword is used to refer to data or functions found in an object instance, but and not directly in the class. Many other languages use the word "this" instead of "self."

To understand the "self" keyword, start by remembering that the code of a class always loads into memory when your script starts running. There the class code sits, waiting, even if it never gets used.

class Customer():

ID = 0

Name = ''

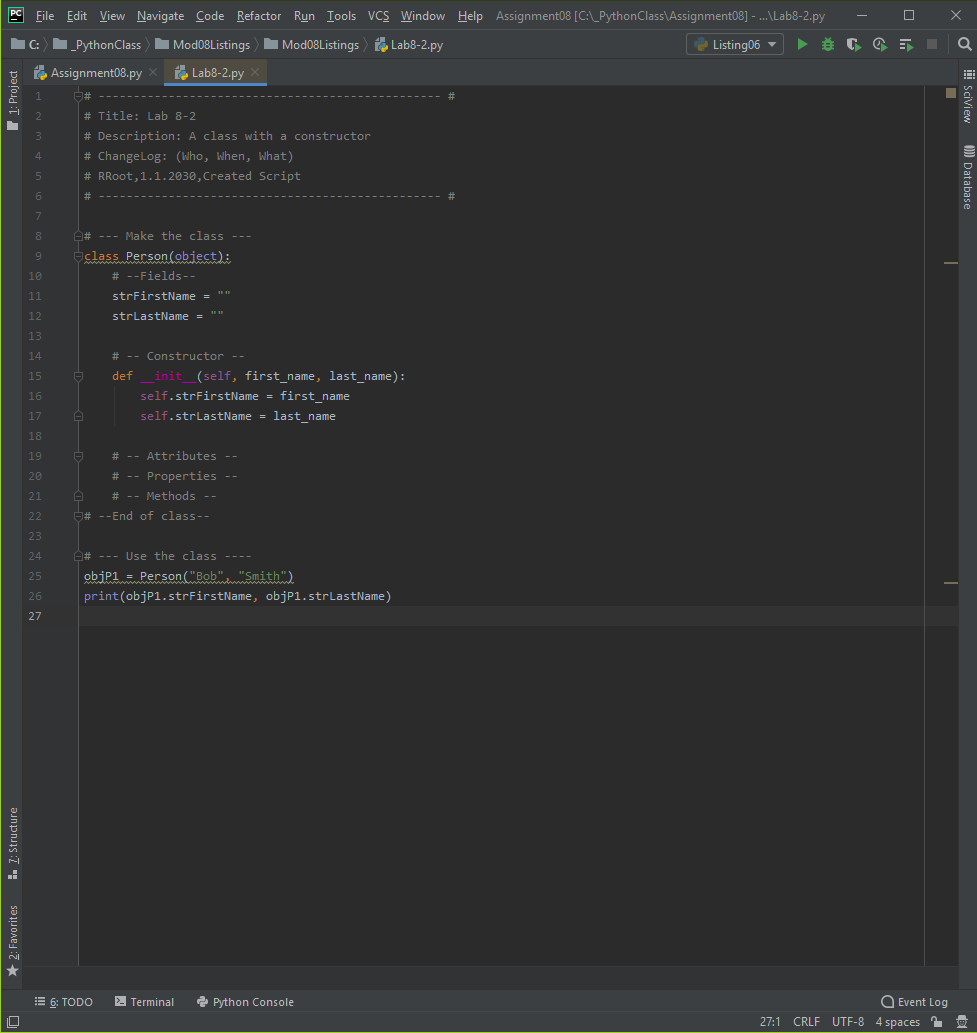
On the other hand, you must explicitly create an object instance in your script.

objC1 = Customer()

You can only load the class's code in memory once, but you can have multiple instances of a class, each representing a "copy" of the classes code! In Python, you identify which copy is referenced using the pronoun "self." like how two people conversing might each refer to themselves!

Because of this, the people who made Python made a rule that you include a parameter called "self" in each method meant to be used from an object instance.

Oddly, you do not pass arguments into this parameter. Nor is it not automatically assumed to be there if you forget to type it in.



**Figure 3** *– Example of how “Self” keyword works (from Module08 Notes)*

**Attributes:**

In Python, Attributes are "virtual" fields that hold internal data. An invisible field is created for you when you use the following syntax in the constructor.

***Note:*** This feature is not typical in most other languages, but it does keep to Python's mostly "automatic" nature!

**Properties:**

Properties are functions used to manage field or attribute data. You typically create two properties for each field/attribute, one for "getting" data and one for "setting data. In fact, you may hear them called "Getters" and "Setters" or "Accessors" and "Mutators."

Setter Properties let you add code for both validation and error handling. If a value passed into the Properties parameter is valid, then it is assigned to the field or attribute. You create a setter like any other function, but it must include the @name\_of\_method.setter directive, and the directive and function name must match!

@first\_name.setter # (setter or mutator)

def first\_name(self, value): # The name must match the attribute!

if str(value).isnumeric() == False:

self.FirstName = value

else:

raise Exception("Names cannot be numbers")

Getter Properties let you add code to format a field's or attribute's data. Often, a Getter is included in a class, even if there is no formatting code. Inconsistently, Python use the @property directive to indicate a getter function.

@property # (getter or accessor)

def first\_name(self): # The name must match the attribute!

return str(self.\_\_first\_name).title() # Title case

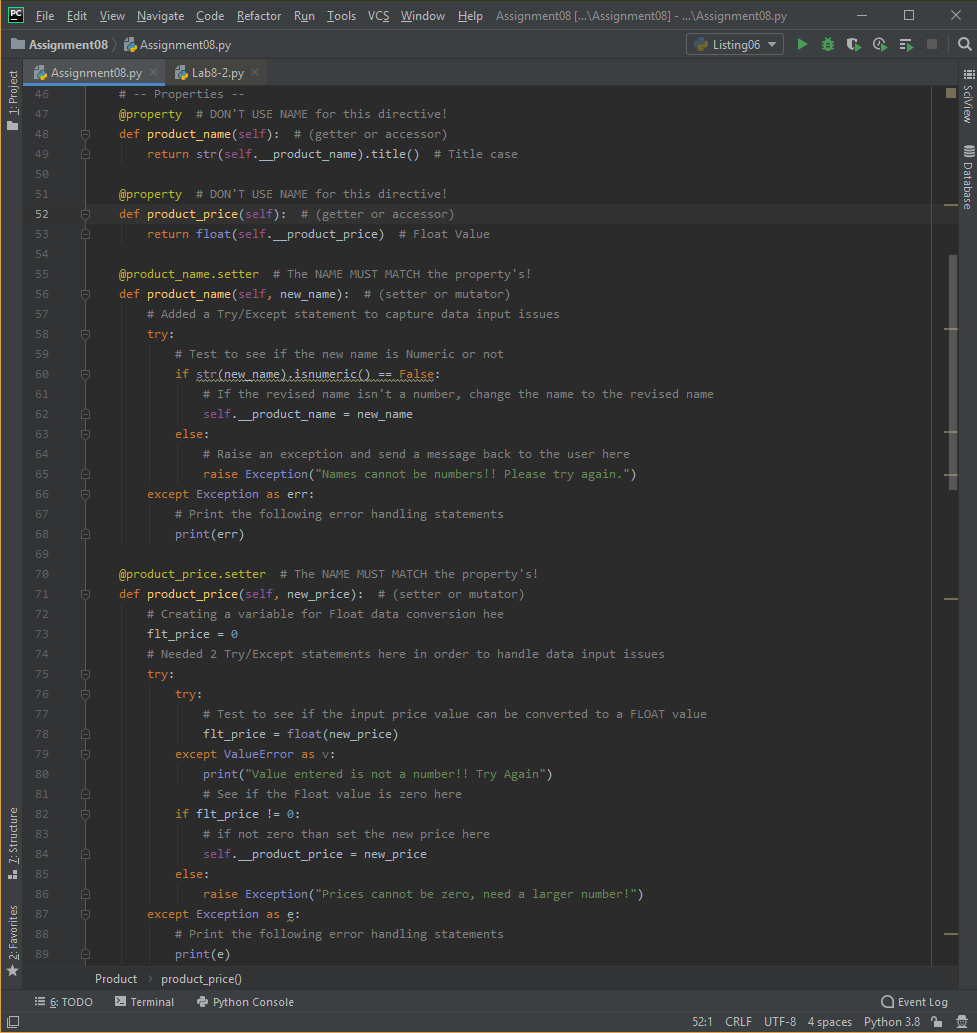
It is considered a best practice to only work with the data in a class through a Method or Property. This practice creates a layer of "Abstraction" and protects software using your class from internal changes to the Fields or Attributes. One of the reasons for this is Abstraction.

***Note:*** Abstraction is an advanced topic that we come back to later in the course.

Forcing Property Use:

When using Properties, you "hide" the attribute using (2) underscores before the attribute's name, which makes the attribute "private" and indicates not to use it directly!

Let me rename the attribute to be "private," and more Pythonic, by changing it to "\_\_first\_name."



**Figure 4** *– Example of how “getter” and “setter” properties (from Assignment08)*

By convention, programmers should respect hidden attributes as being private. However, Python does not vigorously enforce this privacy, unlike most other languages.

**Methods:**

While functions that manage attribute data are called properties, other functions inside of a class are called Methods. Methods allow you to organize your processing statements into named groups, just like functions in scripts do!

**The "\_\_str\_\_()" Method:**

Most classes, in most languages, include a method that returns some or all the class's data as a string. Python has a built-in method that performs this task called the "\_\_str\_\_()" method, but in many languages, it's called something like "ToString()."

Python includes an invisible "\_\_str\_\_()" method if you do not add one to a class. This default invisible method only returns the name of class and an address identifier, and you may have seen this going through the course.

**Static Methods:**

If you want to include methods called directly from the class, without making an object first, you add the @staticmethod directive like this:

class Math(object):

@staticmethod

def Add(Value1, Value2): # You do not need the self keyword

return Value1 + Value2

Now, you can call the method by using the name of the class and the name of the method.

Sum = Math.Add(5, 6)

print(Sum)

***Important:*** You do not need the "self" keyword with a static method since you are not calling the function from an object instance, and because the class is only loaded one, there is no need for a confusing pronoun!

Classes can have both instance methods and static methods, but most will not. In general, when a class focuses on processing data, use "static" methods. However, when a class focuses on storing data, use "instance" methods (the ones with self). For example, a class that holds a customer's name, email address, phone number, and other data about the customer, so use mostly instance methods because you would want to make multiple copies of the class as you program runs, each with its own unique customer data. On the other hand, a class that processes data from a list object into a file and from a file to a list object would be most about performing actions that would not require multiple copies of the class with unique data. This generalization may not always be true, but often enough that it provides a good starting point.

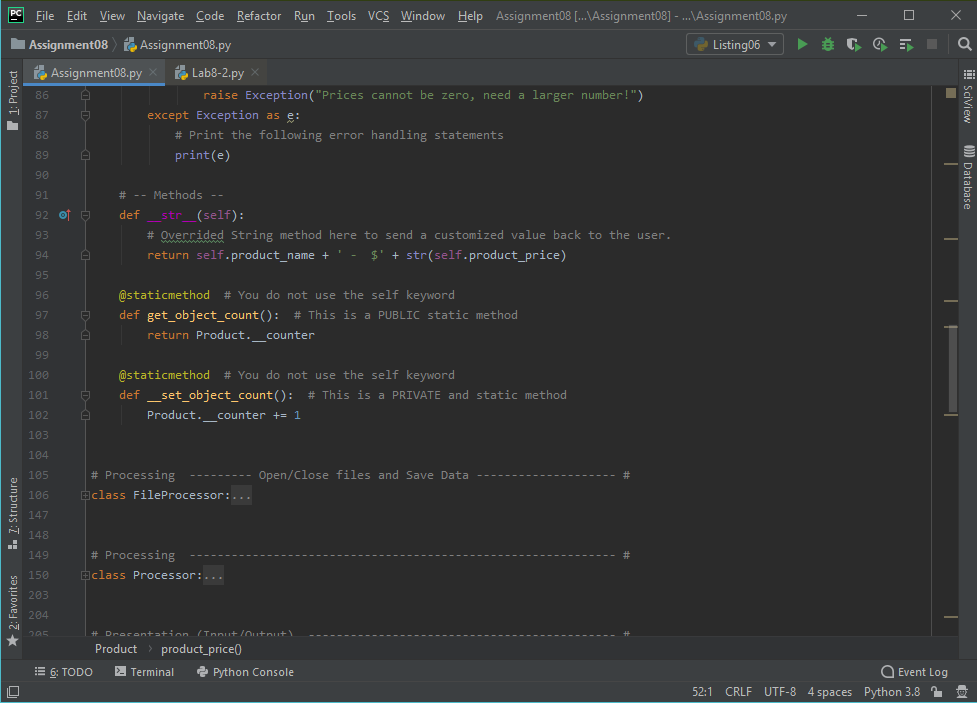
**Private methods:**

If you want a method to be for internal processing only, in other words, "private" to the object or class, you can name it with (2) underscores. These can be occasionally useful for things like tracking how many objects are currently created from a class

Counting objects is a critical task in older languages because you needed to make sure that objects were all removed from memory before the application closed. If you did not do so, they stay in memory taking up valuable space and making the computer run slow. To clear them from memory, you reboot the computer, a task that many of us do regularly.

Python self-cleans object to avoid this, but let's look at an example anyway, since it is something that might come up in other languages.

***Note:*** You do not need to include an object counter in this module's assignment.



**Figure 5** *– Example of what Methods look like (from Assignment08)*

**Type Hints:**

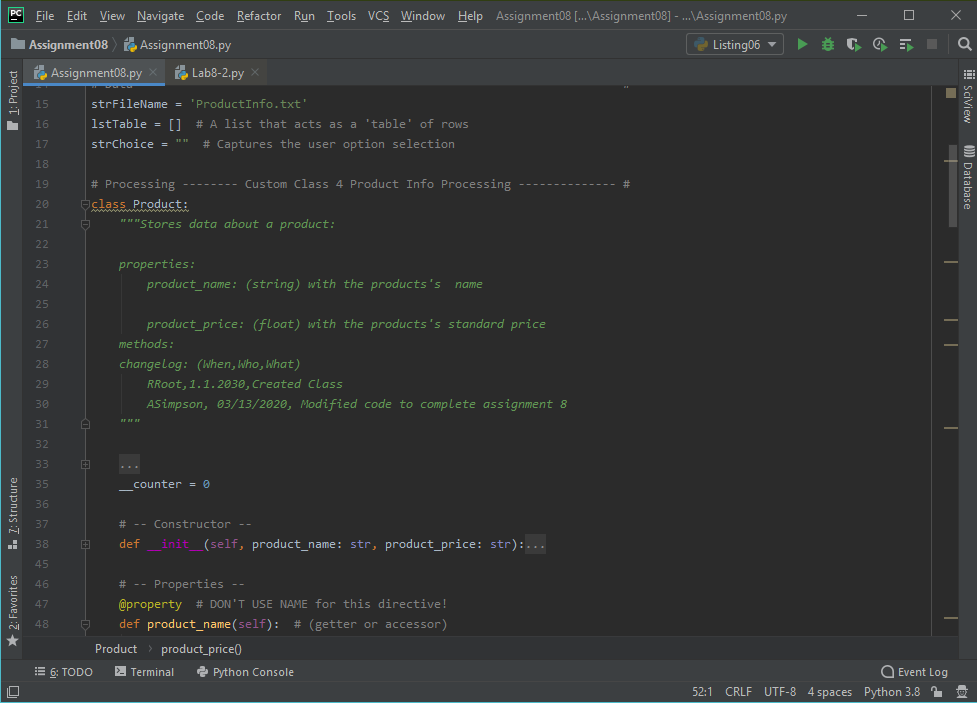
Another documentation feature that was added to Python is "type hints." These are be used to what type of data is expected in a variable. Unlike other languages, the Python runtime does not enforce parameter and variable types, so incorrect argument types can be passed into to a parameter. Still, type hints can help other developers understand what is wanted and some IDEs provided tips to reduce mis-typing. Type hints are commonly used for function parameter and return types using the following syntax:



**Figure 6** *– Example of Type Hints look like (from Module08 Notes)*

**DocStrings:**

Just as we did with functions, we should include a docstring for our classes. It can be helpful if 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). You can also show the "DocString" using the built-in and inherited \_\_doc\_\_ property.



**Figure 7** *– Example of how to create DocStrings (from Assignment08)*

**Using Classes:**

While processing classes are used directly by calling their static methods, data classes are mostly used indirectly through a set of one or more objects.

objP1 = Person("Bob", "Smith")

objP2 = Person("Sue", "Jones")

The data in these objects are like a row of data. These "rows" are often collected into a list object.

lstPersons = [objP1, objP2]

You can think of this collection like a table of data in a spreadsheet or database.

**GitHub Desktop:**

In this course, we've used GitHub's website through a web browser, but often developers work with GitHub differently. Instead of browser, they work with GitHub on their local computers using either with a command prompt application or a desktop application. In both cases, the communication between the GitHub website and the local computer is handled by a program called "Git."

**GIT:**

The Git software manages versions of one or more files. It allows you to make a clone (copy) of a file, then make changes to clone, and save it as a new version of the same file. All while maintaining a copy of the original version. In addition to managing the cloned file on your computer, by default, Git uses the GitHub website to store backup files in the "Cloud."



**Figure 8** *– Example of how Git and GitHub work (from Module08 notes)*

**GitHub Desktop:**

In most organizations, developers use command shell to interact with the website from their local computers, but in this module, we start with something more visual; GitHub Desktop. GitHub Desktop is a free application you can install on both Windows and Mac OS.

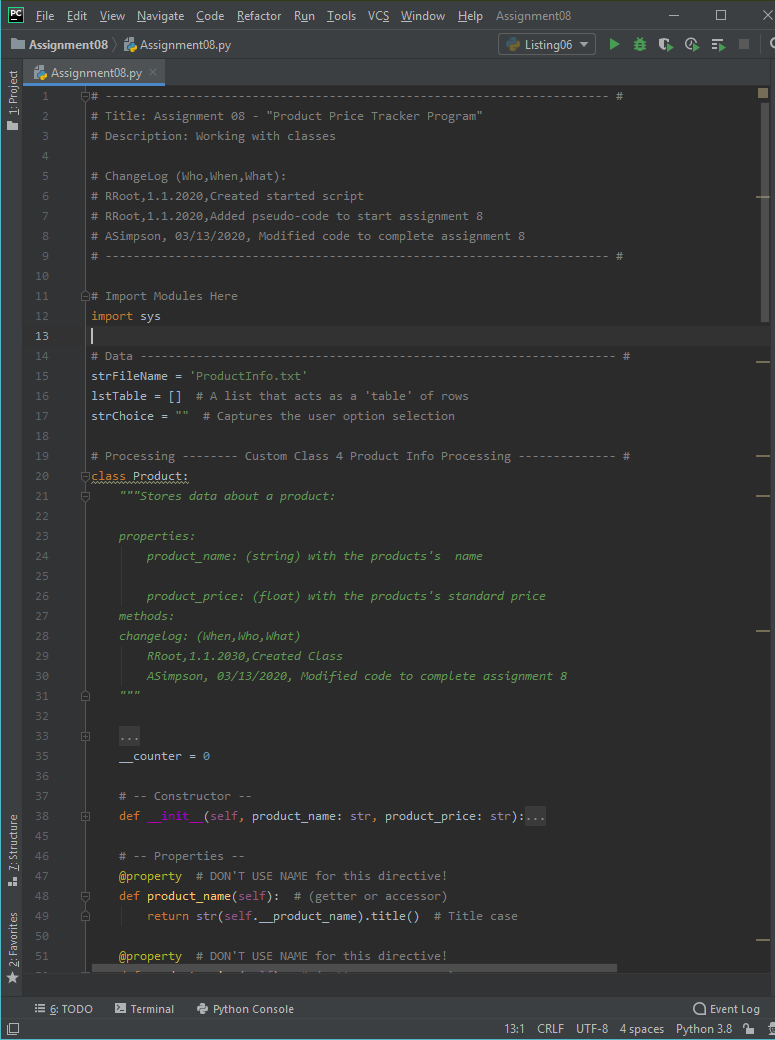
To install GitHub Desktop, you navigate to its download page <https://desktop.github.com/> select your operating system, and download the installation file. Then start the installation, which is quick and straightforward!

In the "Clone a repository" dialog box, verify the GitHub.com repository and the local folder where the files are copied to, before clicking the "Clone" button. This button starts the download process.

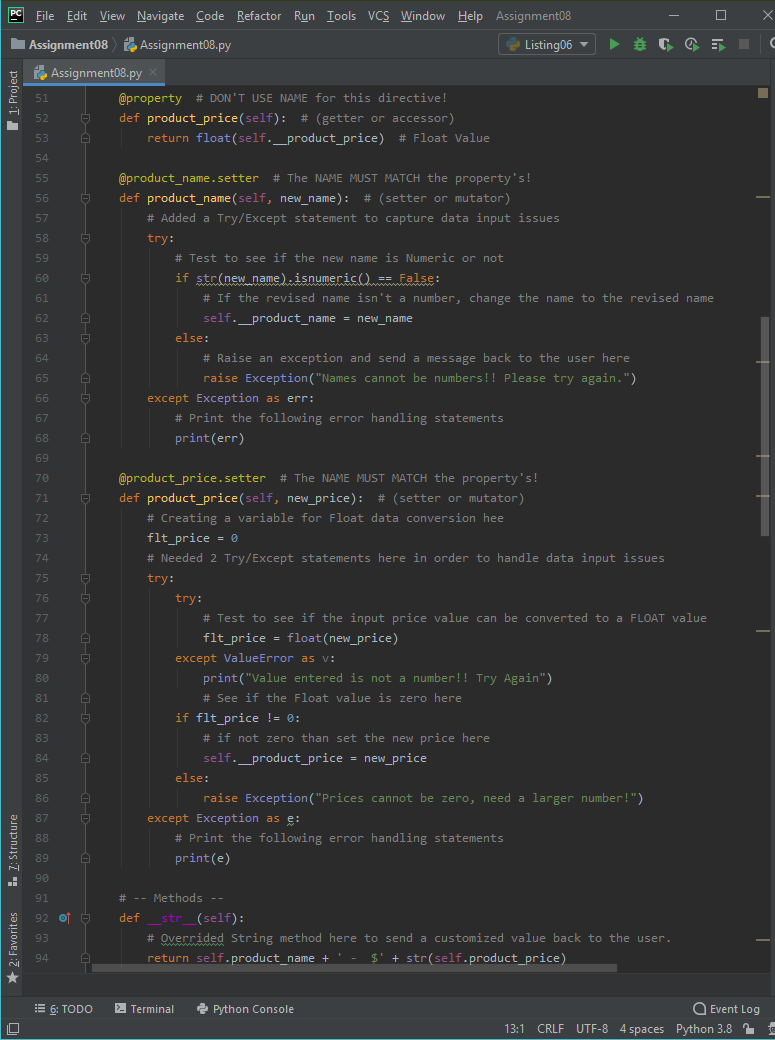
Once the files download into the folder, you can open the folder and see the copied files. GitHub Desktop offers a convenient button to open the folder, but of course, you can always use Windows Explorer or Finder to locate the files.

**Assignment08 – The “Product Price Tracker Program” Script:**

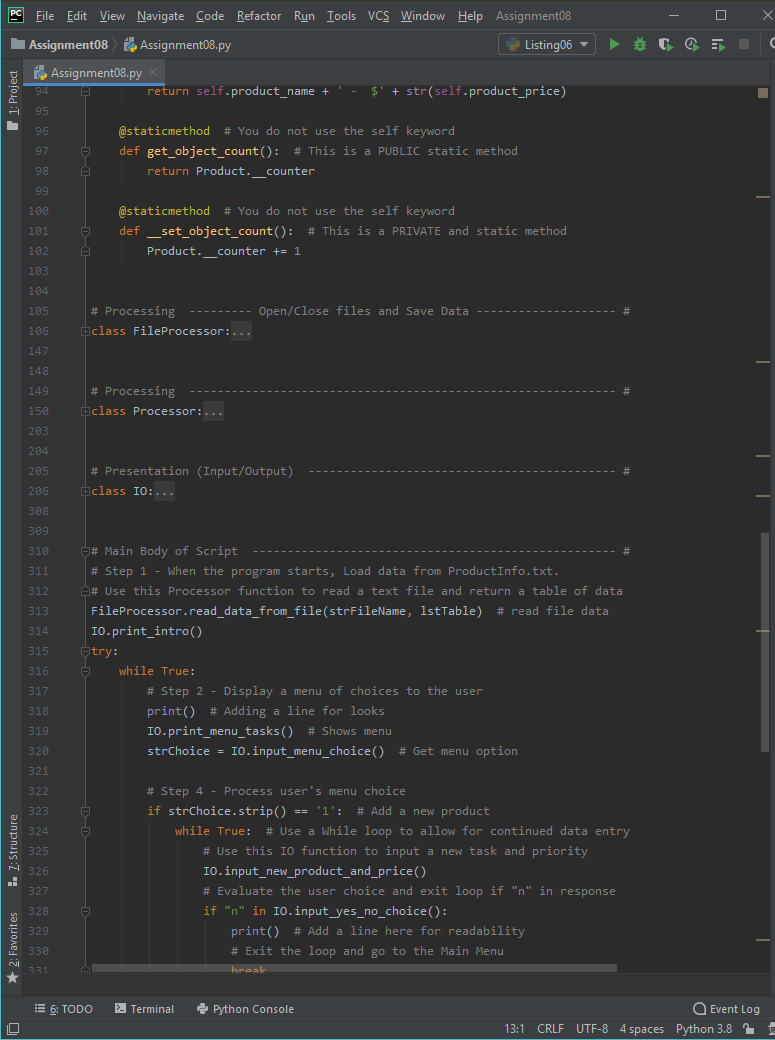
Figures 9, 10, 11 and 12 show the code from the “Product Price Tracker Program” Python Script. Figures 13 thru 15 show this script running from within the PyCharm IDE. And then Figures 16 thru 19 show the script running from the Command Console.



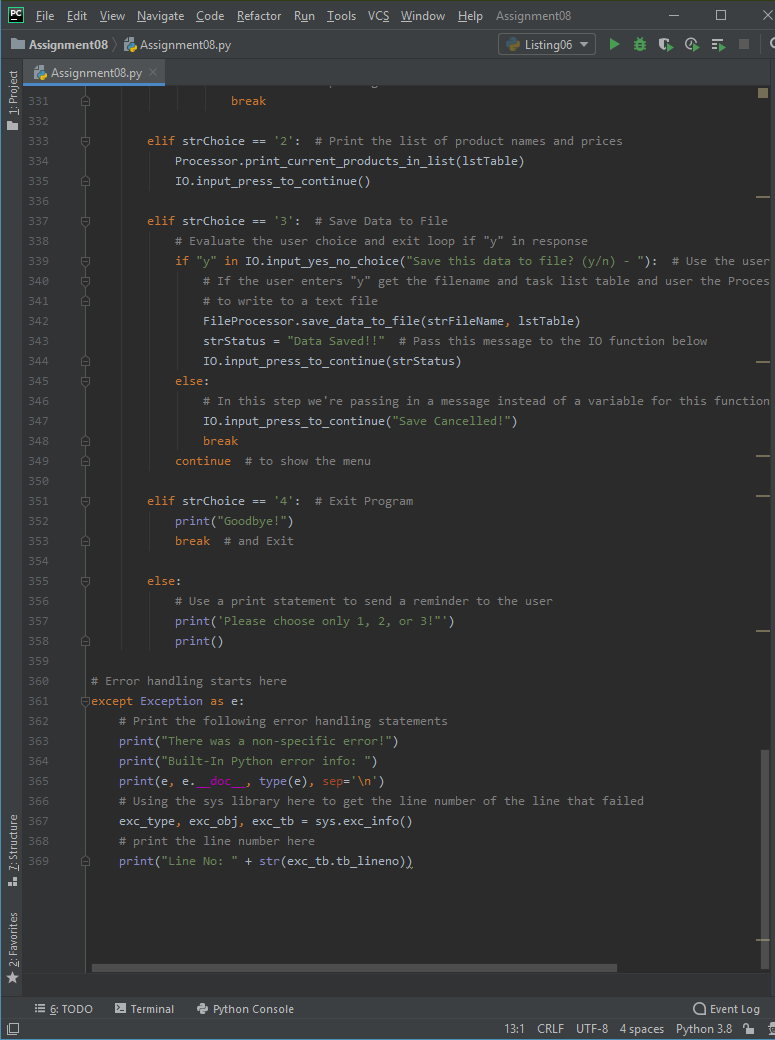
***Figure******9*** *– “Product Price Tracker Program” Python Script (Assignment 8) using PyCharm IDE*



***Figure******10*** *– “Product Price Tracker Program” Python Script (Assignment 8) using PyCharm IDE*

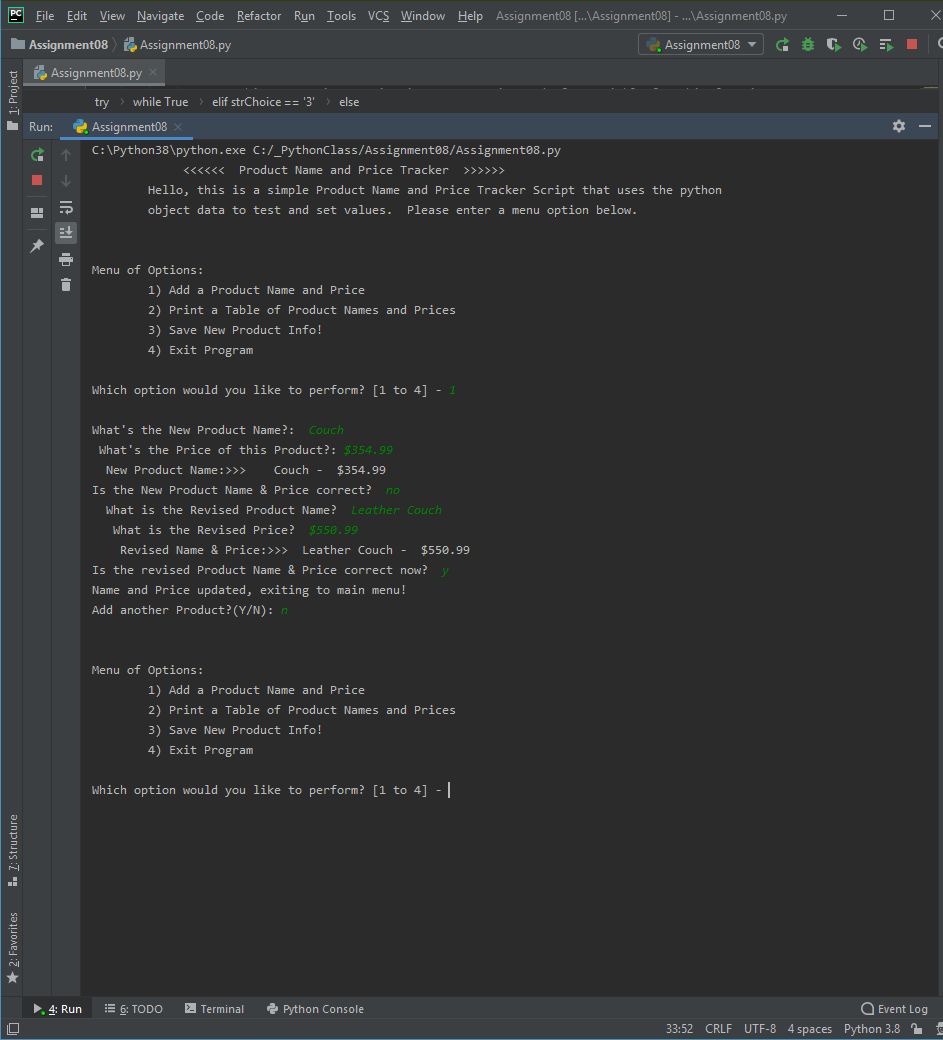


***Figure******11*** *– “Product Price Tracker Program” Python Script (Assignment 8) using PyCharm IDE*

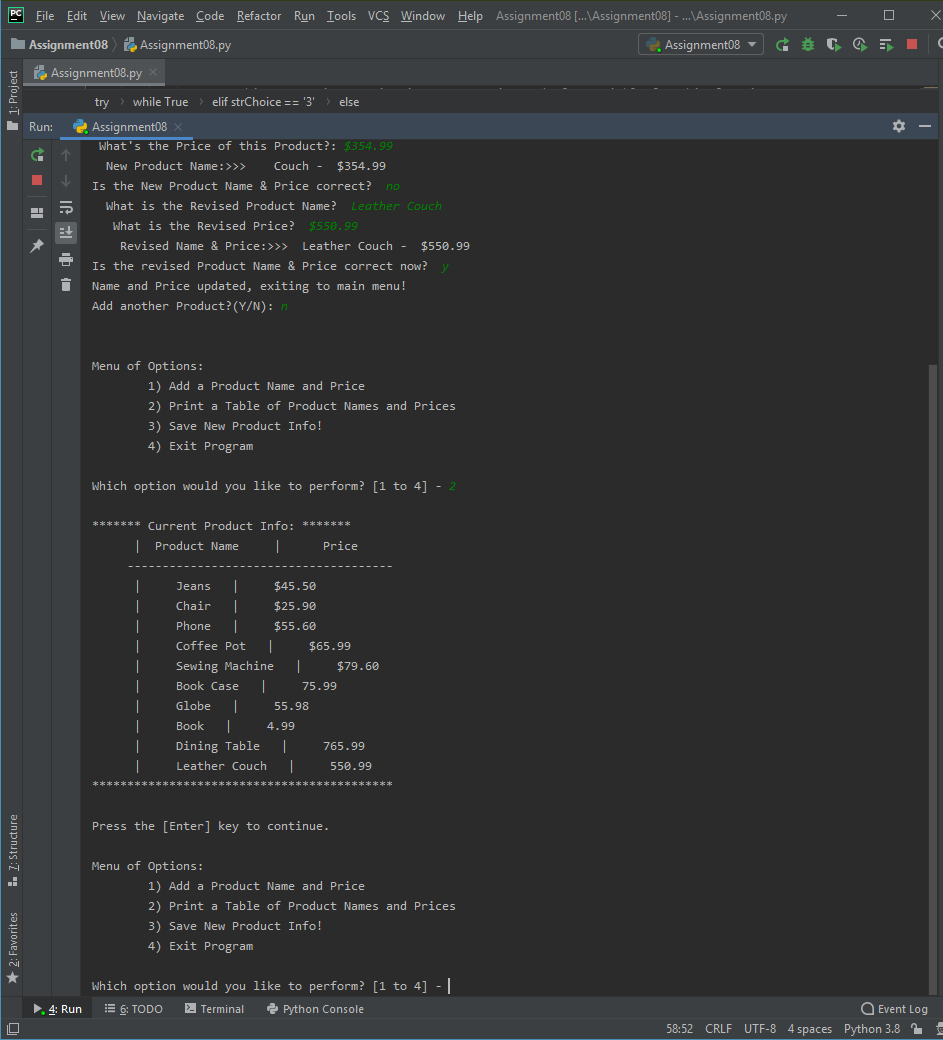


***Figure******12*** *– “Product Price Tracker Program” Python Script (Assignment 8) using PyCharm IDE*

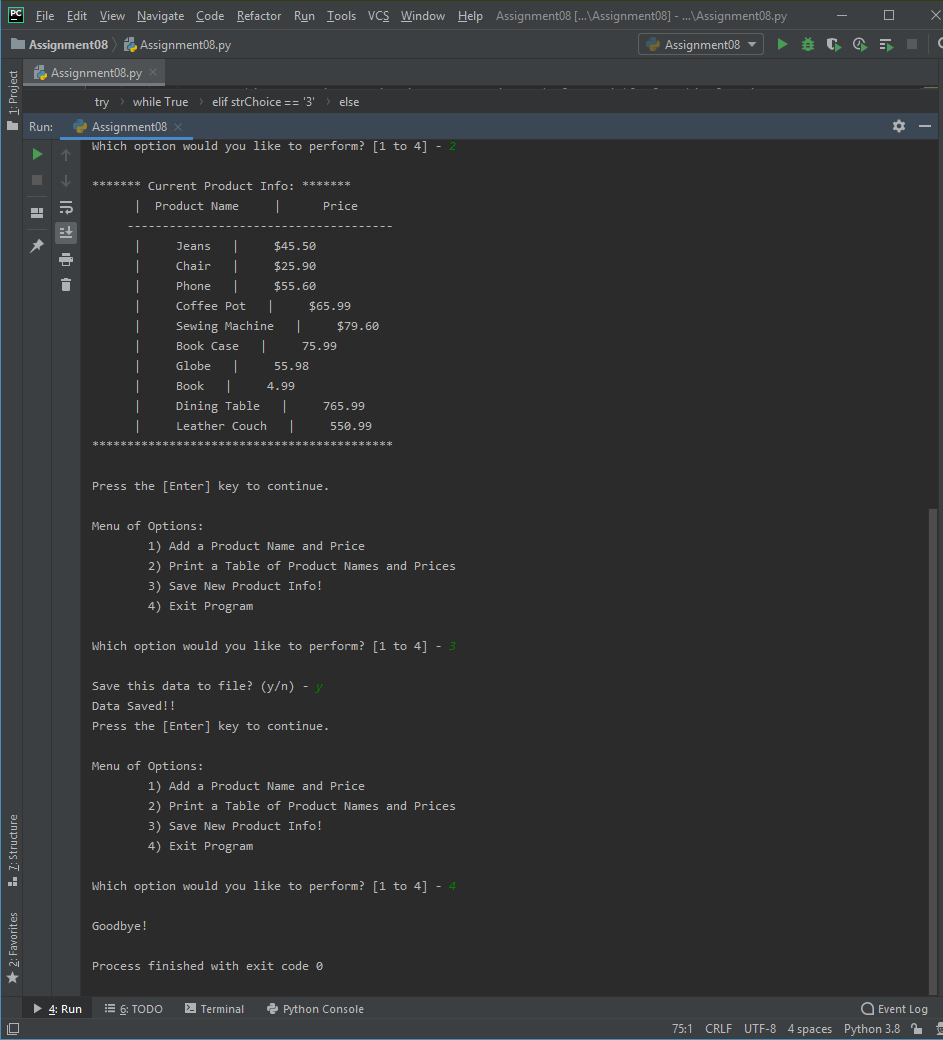
The following screenshots, Figures 13, 14 and 15 shows the Assignment 8 “Product Price Tracker Program” script running from within the PyCharm IDE. Please note that I added an introduction to the script as well as a message informing the user that the script has closed.



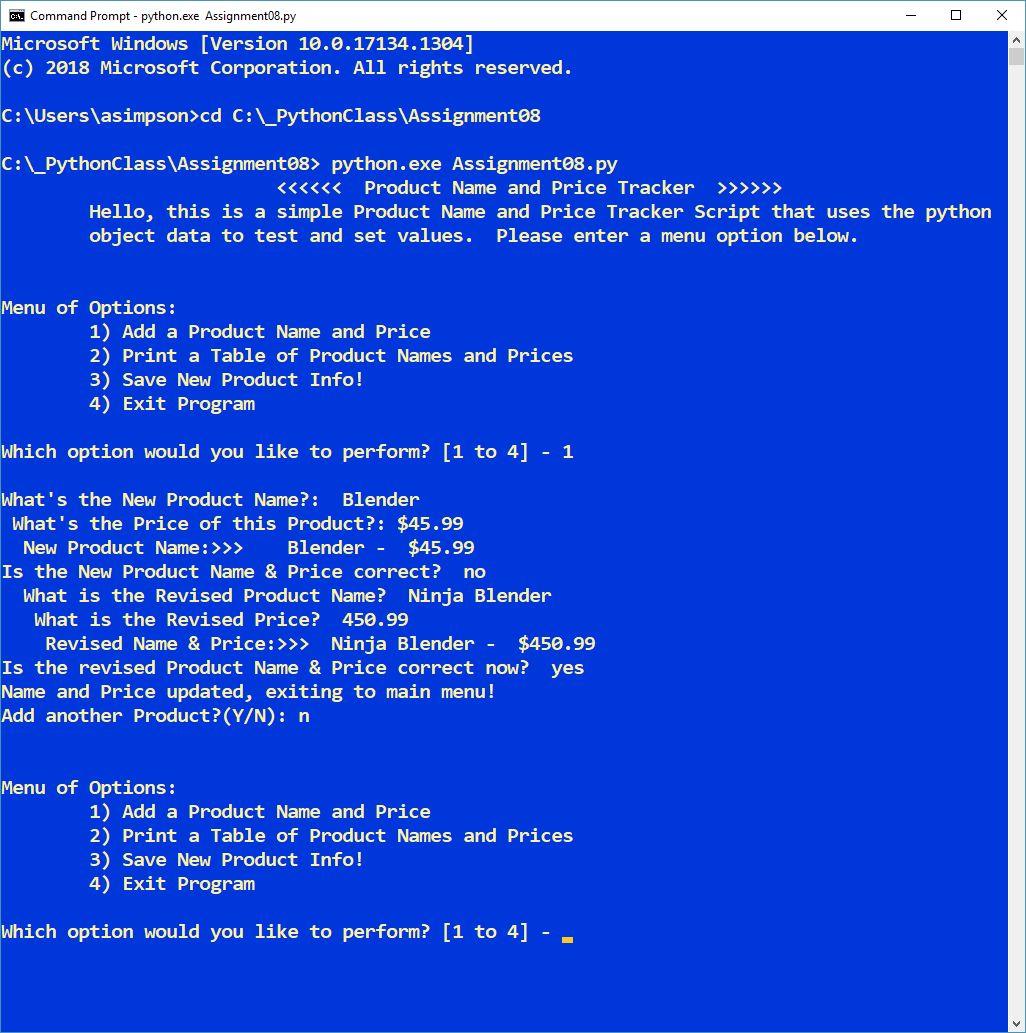
***Figure******13*** *– Running Option #1 from the “Product Name and Price Tracker” Python Script (Assignment 8) using PyCharm IDE*



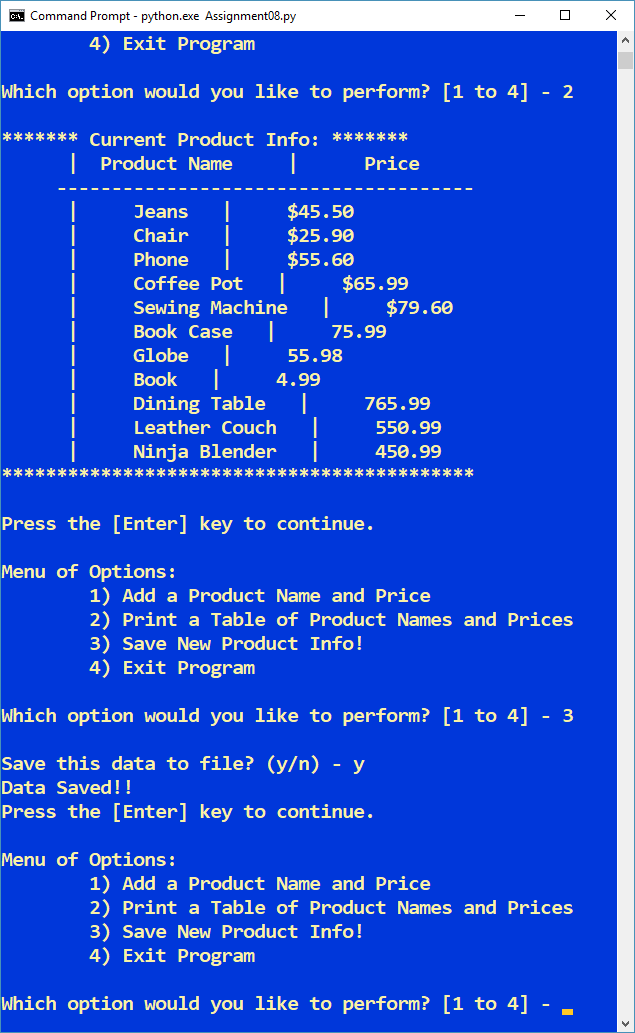
***Figure******14*** *– Running Option #2 from the “Product and Price tracker Program” Python Script (Assignment 8) using PyCharm IDE*



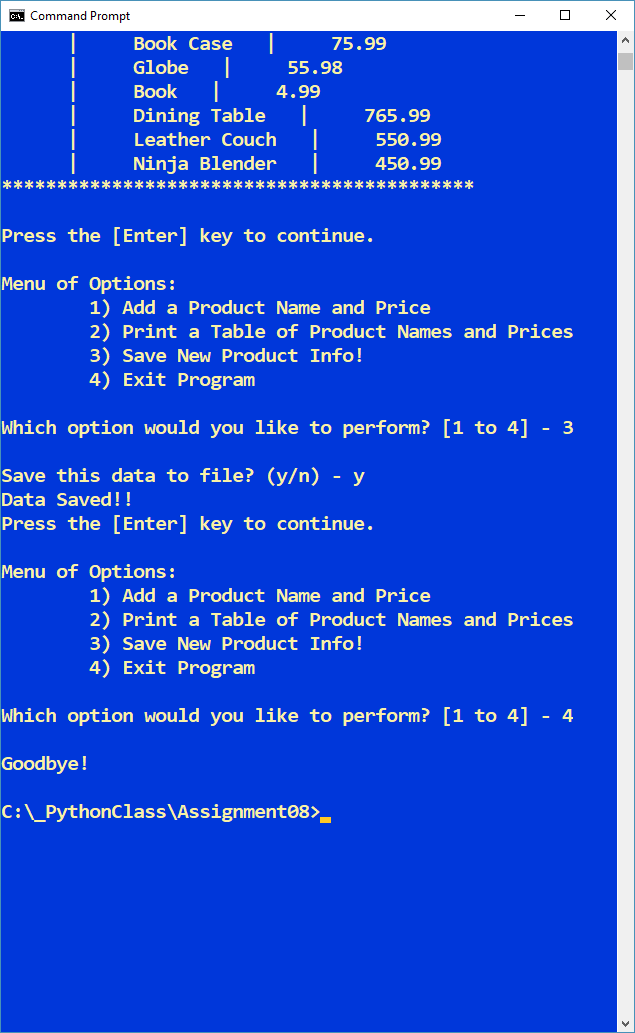
***Figure******15*** *– Running Option #3 and #4 from the “Product and Price tracker Program” Python Script (Assignment 8) using PyCharm IDE*



***Figure******16*** *– Running Option #1 from the “Product and Price tracker Program” Python Script (Assignment 8) using using the Command Console*



***Figure******17*** *– Running Option #2 from the “Product and Price tracker Program” Python Script (Assignment 8) using the Command Console*



***Figure******18*** *– Running Option #3 and #4 from the “Product and Price tracker Program” Python Script (Assignment 8) using the Command Console*

**Summary**

In summary, as part of module 8 and assignment 8, I learned all about classes and object oriented programming.

I learned how software objects can combine functions and data (methods and attributes) to mimic real world objects. How to write classes, the blueprints of objects. I learned about a special method called a constructor that is automatically invoked when a new object is instantiated. I also learned how to create and initialize object attributes through a constructor. I also learned about ow to create class-wide elements such as attributes through constructors. I also learned about GitHub Desktop and how that works with the GitHub cloud environment.

In conclusion, I put many of the concepts mentioned above into the “Product and Price Tracker Program” python script. I’m ready to take on chapter 9 where we’ll learn more about Object-Oriented programming (OOP).