Eric Forster

Dec. 6th, 2020

IT FDN 110 A Au 20

Assignment 08

github.com/e4ster/IntroToProg-Python-Mod08

**Objects and Classes**

**Introduction**

This week I made a script that allows a user to make a list of products and their respective price. In order to do this, I made a class definition which stored the name and price attributes. This provided a blueprint for me to make an object and store the data each time the user wanted to add data to the list. The script is broken into data, processing, IO, and the main body. For the sake of space, I’m not going to review the header of the script.

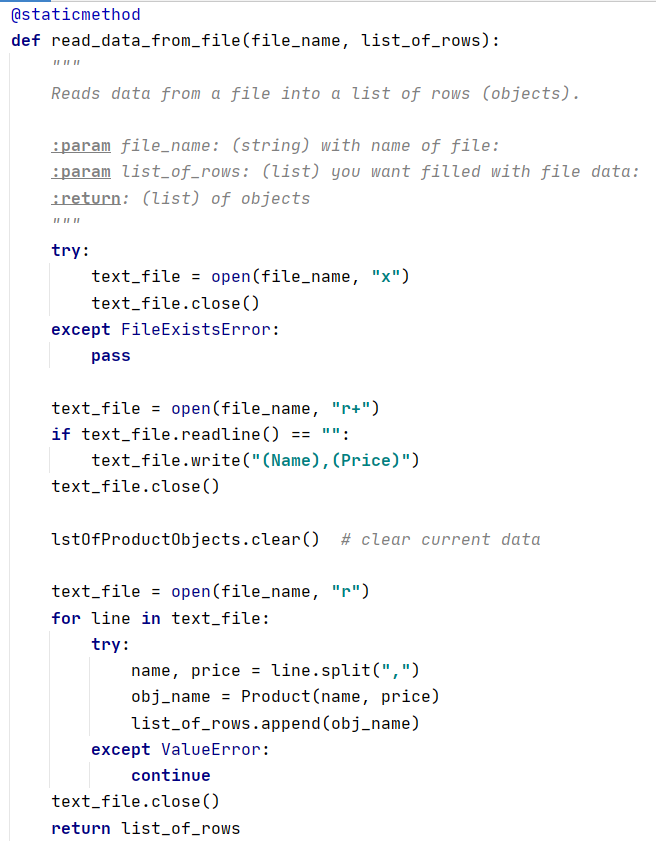
**Step 1: Load Data from File**

To start the main body, I ran the following line:



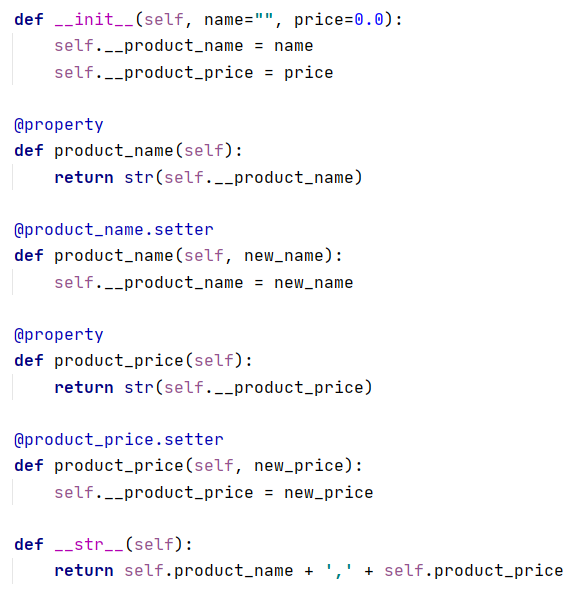
***Figure 1.1: Executable line to read data from file.***

In this line, I use strFileName and lstOfProductObjects as arguments. These were my only two global variables in the script. These are passed to parameters in a static method contained in the Processor class. This means I can call the method without it belonging to an object first.



***Figure 1.2: Method that reads text file at start.***

At the beginning when the script is ran, I use a try/except block to try to open the file. If the file doesn’t exist, it will create a file and then close it with the x mode. Next I open the file again to see if there is anything in the first line with the r+ mode. If the line is empty, it adds a Name and Price header for the columns of data. Then I clear the global list of objects to ensure the list is fresh, and then open the file again in r mode. Using a for loop, I read through each line and split the value with the comma, and then created a new object called obj\_name. By assigning it the value of Product(name, price), it is now an object of type Product, and inherits that contents and attributes. I passed the name and price variables to the \_\_init\_\_ constructor method, which assigns the values to those attributes as the object is created. I will display the Product class in a moment. Once the object had data stored in it from the first line, I appended it to our global list, and then it moved down to the next line in the text file. The ValueError exception is included because at first I was having issues with carriage returns causing empty lines in my text file, but isn’t needed anymore. Let’s take a look at the Product class.

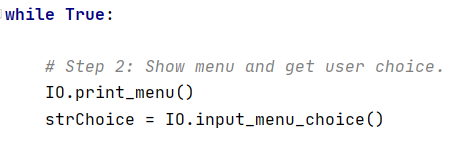


***Figure 1.3: Product class definition.***

For the sake of space I’m not going to cover the class title and header. The \_\_init\_\_ method up top is the constructor which is ran automatically and assigns values to the product\_name and product\_price attributes. Each object received their own values. At the bottom I defined the \_\_str\_\_ method which defines what is printed when the object is called to print. In the middle, I have the getters and setters. In this script I didn’t end up using the .setters, but I used the getters. This meant I called .product\_name to read the value of the private attribute .\_\_privatename. The properties provide a layer of security between the user and the private methods, and can filter/customize how the attributes are accessed. Once all the objects are appended after reading the text file, the list is returned.

**Step 2: Show Menu, Get User Choice**

Next I executed the following chunk of code:

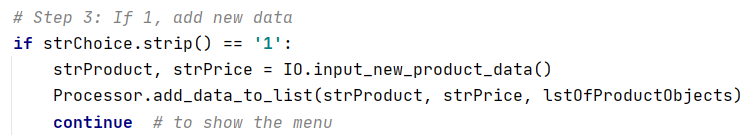


***Figure 2.1: Calling the menu and input methods.***

This starts the Main loop. I called a print\_menu static method from the IO class. This simply displays some menu text for the user to see. They had the option to choose 1, 2, or 3. Next I called the input\_menu\_choice method from the IO class, and then assigned the returned value to strChoice. I won’t display this static method because it is simply one input() method. It returns character they typed as a string, which I later evaluated to see if it is a 1, 2, or 3.

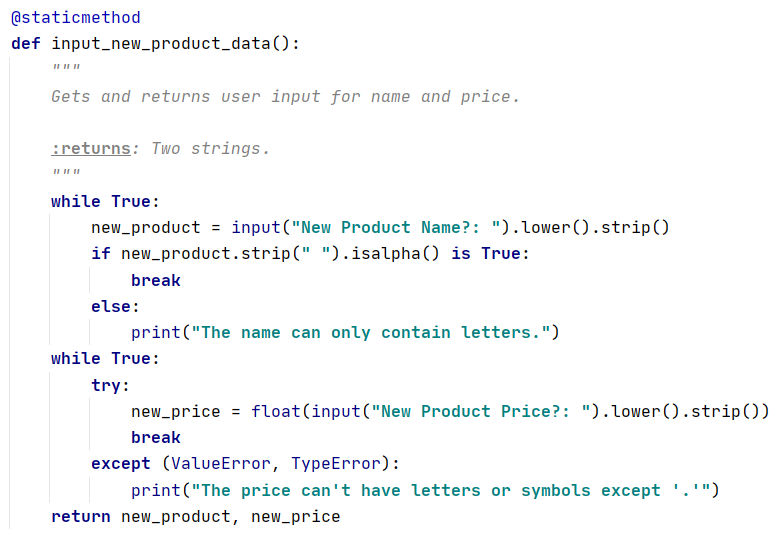
**Step 3: Add New Data**

Using an if, elif, else block, I determined what the user wanted to do. If 1 was chosen, they wanted to add data to the list. Here is the executable code:



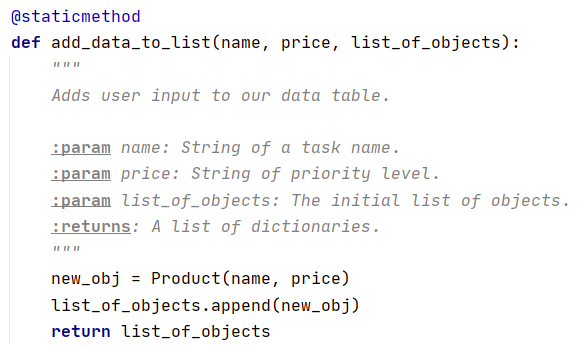
***Figure 3.1: Calling methods for user input, and adding to list.***

First I called the input\_new\_product\_data method.



***Figure 3.2: Getting user input data.***

Using two while True loops, I get user data that is acceptable for the list. The name loop checks to see if each character is in the alphabet, and the price loop checks to see if each character is a number. This ensures our data we are storing makes sense for later use. The method returns the two values. These two values are then passed to the next method, add\_data\_to\_list.

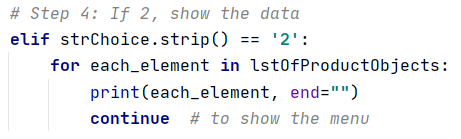


***Figure 3.3: Adding data to our list.***

In this method, it created a new object of type Product, and passed is the name and price parameters for its attributes. Then it appended that new object to our list of objects, and returned our new completed list.

**Step 4: Show the Data**

When the user chose option 2, I called this executable:

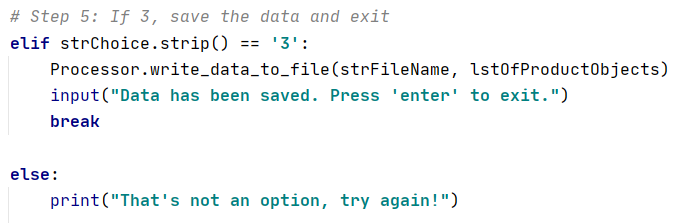


***Figure 4.1: Printing the data.***

For this chunk, I just used a simple for loop to iterate through each element in the list of objects. Since each object had its own \_\_str\_\_ method put together, it printed all of the data the user needed to see. This was a case where I thought it was easier to not create a separate method.

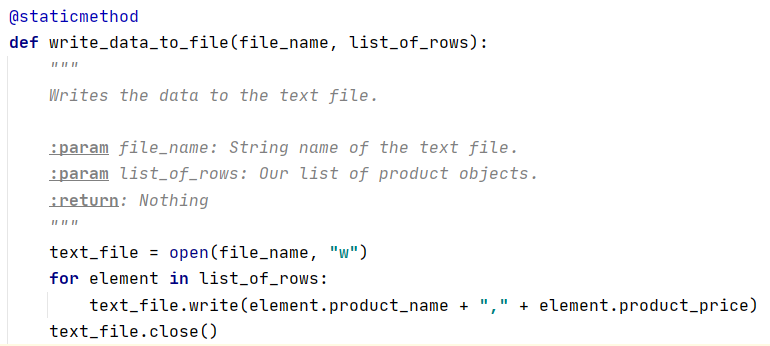
**Step 5: Save Data and Exit**

The last bit of executable code is shown below:



***Figure 5.1: Executable to save and exit.***

I first called the write\_data\_to\_file method. Let’s take a look:

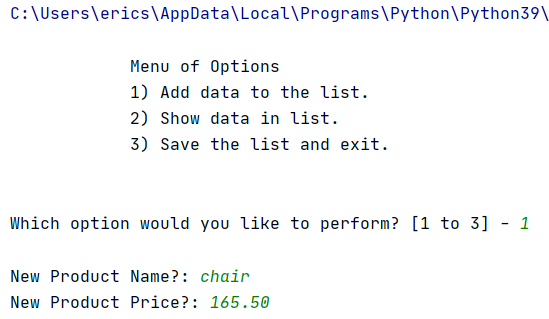


***Figure 5.2: Method that writes to the file.***

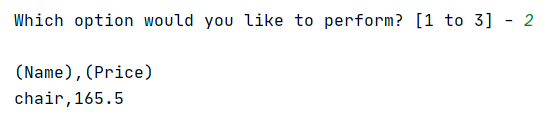
I opened the text file with the w mode, and then iterated through each element in my list of objects. For each element, I wrote a string which consisted of a getter for the product name, and a getter for the product price, separated by a comma. Once this is complete, the program let’s the user know the save is complete. The final else statement is a catch-all in case the user entered a choice other than 1, 2, or 3.

**Step 6: Running the Script**

I will first show the program running in PyCharm.

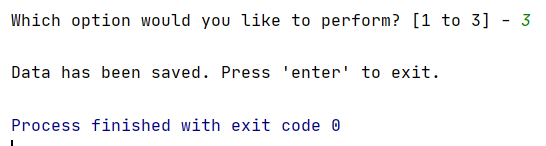


***Figure 6.1: Adding data to list in PyCharm.***



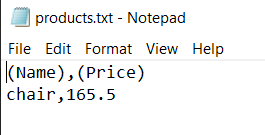
***Figure 6.2: Choosing option two.***

As you can see, I added one object to my list, but when I printed the current list, the column headers were already added when the file was created.

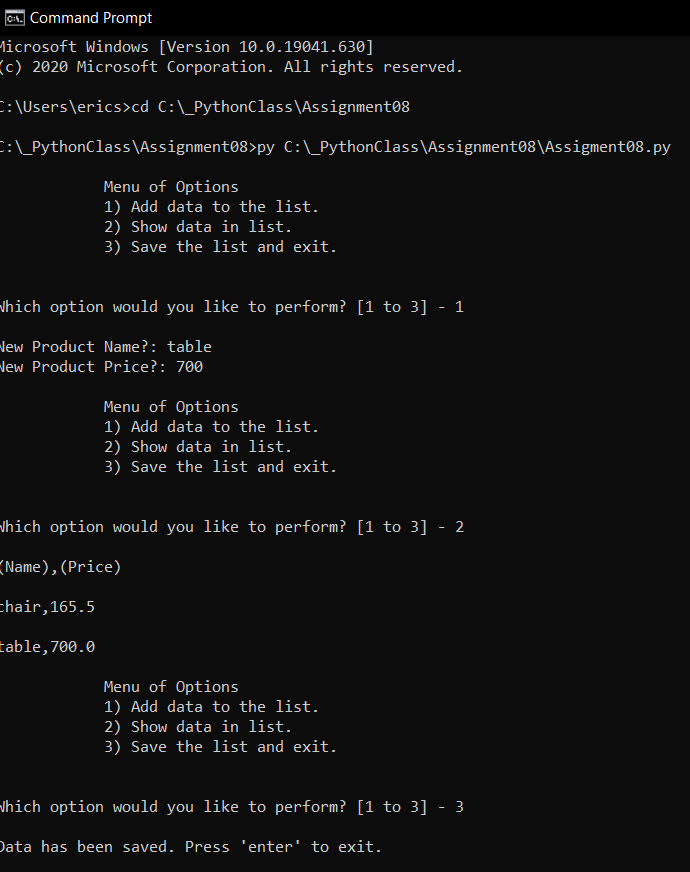


***Figure 6.3: Option three.***

Option 3 saves the data and allows you to exit. See the text file created:



***Figure 6.4: Data saved to the text file.***



***Figure 6.5: Script ran in the command prompt.***

**Summary**

At first glance, attributes, properties, fields, and variable all seem to blend together. After going through this exercise, I now see the subtle difference between attributes and properties, and how properties can help keep your attributes safe. The keyword ‘self’ I found to be confusing as well, but I think it is just one of those syntax details I will get used to. Understanding these syntax details opens up a very powerful world of objects and inheritance. As we move toward next weeks lesson, I hope to explore all of the double underscore built in methods and find their uses.