Dmitriy Alexandrov

November 27th, 2019

IT FDN 100 A

Assignment08

<https://github.com/umknownone/IntroToProg-Python-Mod08>

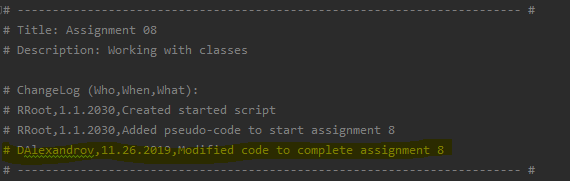
Coding Product Management Script using Classes and Error Handling

# Introduction

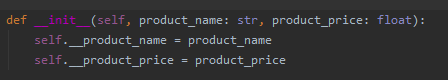
In this paper I am going to go through the steps that I took in adding code to a starter script in order to create a fully functioning Product Management script with a user menu, utilizing classes and error handling.

# Steps Taken in Adding Code to the Starter Script:

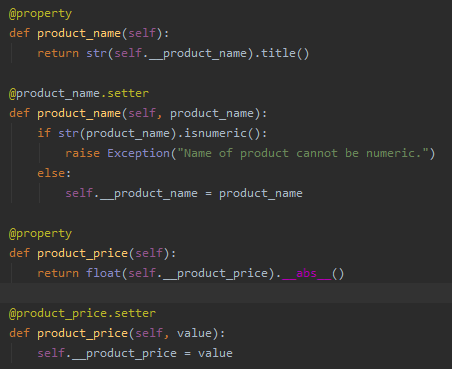
I began with creating a new project in PyCharm called Assignment08 and importing the Assignment08\_Starter.py from the course module. After looking through the script, I first began with updating the header at the top of the script with my information in the Change log section.



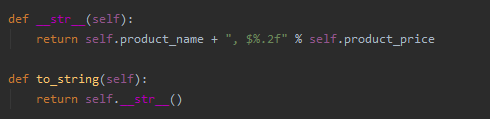
After that, I began with the product class. I saw that the class doc string already had 2 properties in there (product\_name that is a string, and product\_price which is a float value). So I started off with creating the Product class constructor which would take in the self, product\_name variable (with the preferred str), product\_price (with the preferred float) as shown below.

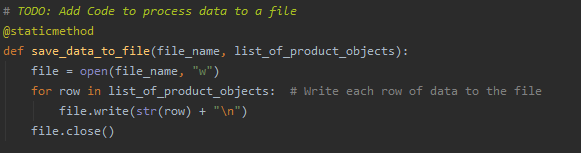


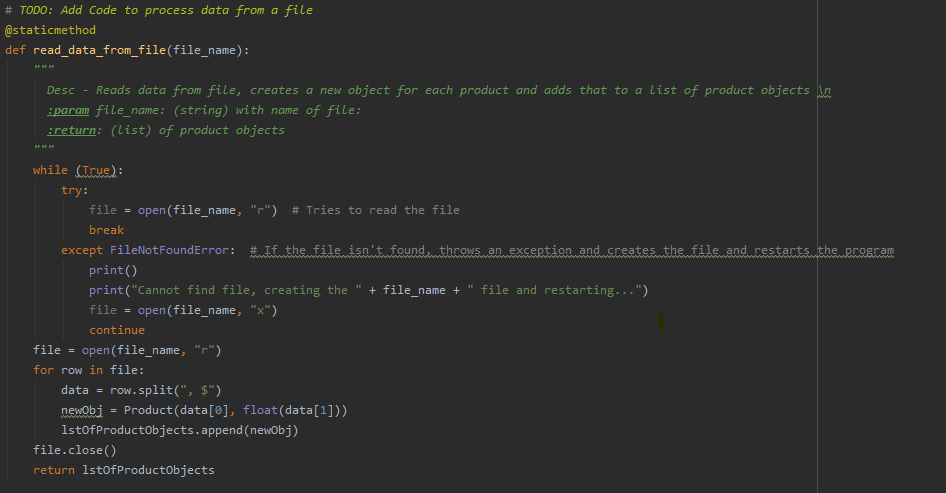
After the creation of the constructor with its two private variables of self.\_\_product\_name and self.\_\_product\_price which take in the input of the values passed to the Product class. I created the getter and setter methods for each property as seen below. Also the setter for the product\_name method included an error handle where if a numeric value was passed for the product name then an exception would be raised to let the user know that a product cannot have a numeric value.

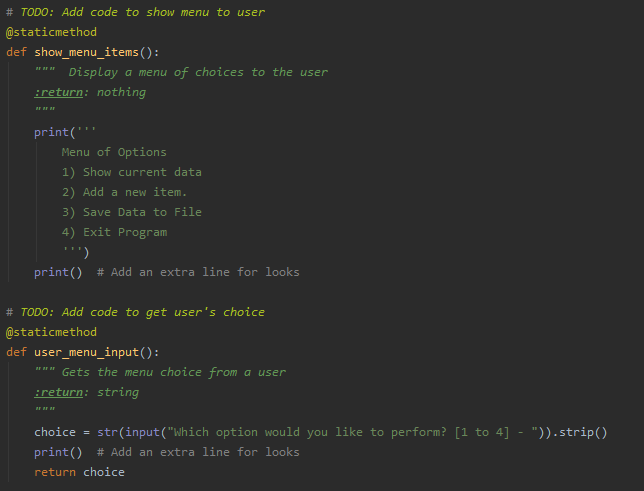


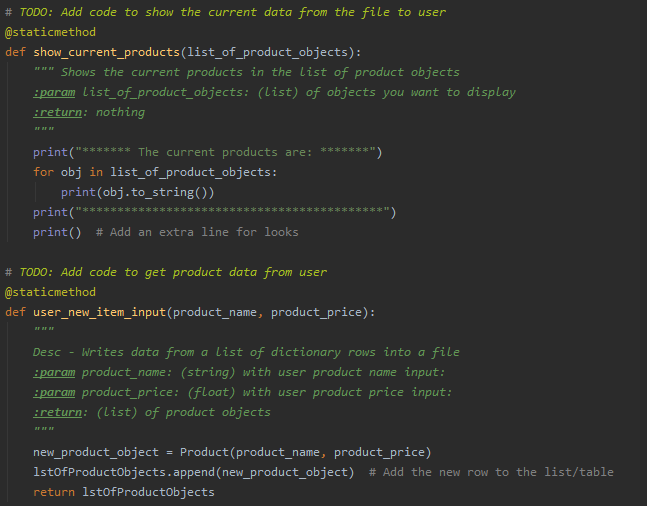
After the getter and setter methods were created for the Product class, I created a custom to\_string and an overwrite of the original \_\_str\_\_ methods specifically for the Product class. Both the new \_\_str\_\_ and to\_string methods return the name of the product follow by a “, $” and the product price for formatting purposes.

After testing and ensuring that all the features of the Product class were functioning as needed, I moved onto creating the FileProcessor class. For this class, I started out by going back to Assignment06 and coping the save to file function as well as the read from file functions into this script as a starting point. I then updated the code for the “save\_data\_to\_file” function to take in a file\_name and a list\_of\_product\_objects per the doc string above. The “save\_data\_to\_file” function looked as following:

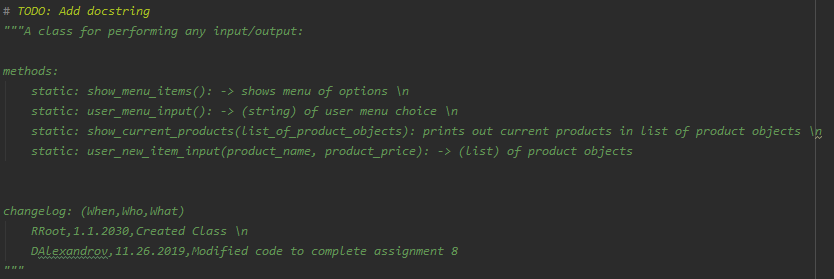
For the “read from file” function, I also modified the code that I had written for Assignment06 and changed it so that instead of creating a dictionary row from the file data, instead the data was split on the “, $” (which is what my formatting for products was above) and a new object of the Product class was created for each item in the file. Then these new product objects are appended to the lstOfProductObjects list.

This completed the tasks in the FileProcessor class, and after testing I moved onto to the IO class. In this class I started out with once again copying the static functions from Assignment06 that were similar to what was needed to be achieved in this assignment. I copied over the menu function that just showed the menu options and changed it to reflect the options of this assignment. I also copied over the user\_menu\_input function that takes the user input and returns what the user chose.

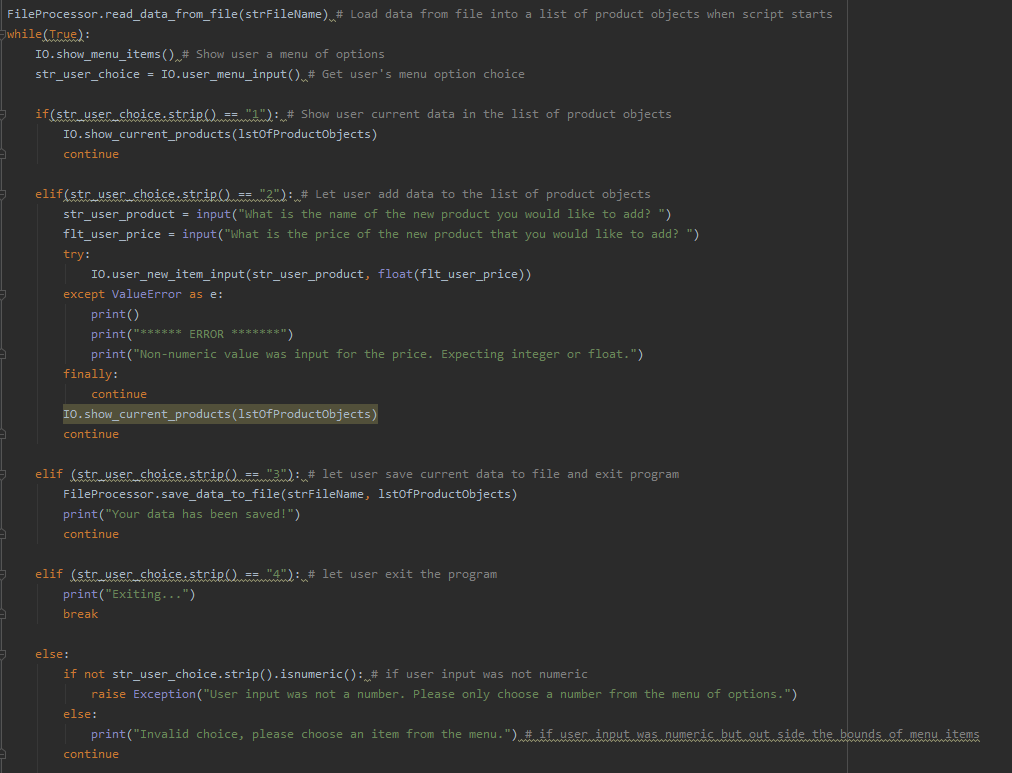
Then I copied over and modified the show current data function, which now takes in the list of product objects (lstOfProductObjects) and for each item in that list prints out using the custom “to\_string” method that I created in the Product class above. Next, I added the method to add a new product named “user\_new\_item\_input” since this method would be processing the user input and passing it over to the Product class to create a new product object which in turn would be appended to the list of objects.



Lastly after testing and ensuring that all the methods that I had just added to the IO class were functioning as needed. I went back to the top of the IO class and added the Doc string at the top so that another person using this code could understand what the class is doing and what methods were in it.



After all the classes, properties and methods were functioning as needed I moved onto the main body of the script. Where I began by calling the FileProcessor class and the “read\_data\_from\_file” method passing it the name of the file (strFileName) to load the file contents into memory. Then I created a while loop as we have done before to keep cycling the user through the menu as they chose different options. The menu and user input were taken in each time cycle of the while loop using IO class and the “show\_menu\_items” and “user\_menu\_input” methods. Depending on which option the user chose, a different class and method would be called to perform those actions. Error handling for the creation of a new item was introduced in case the user did not input a numeric value for the price. And an option to exit the script was added. Also error handling was added if a user did not choose a numeric value from the menu.



# Summary

Overall, the script performed the functions that were asked of it in the assignment. It was really interesting to practice the use of classes and being able to call upon your own created classes to perform actions in your body of code. Below are the outputs of the code running in PyCharm and in the Terminal:

