Name: Cameron Allen
Date: June 10, 2023
Course: IT FDN 110 A
GitHub: https://github.com/ct-allen/IntroToProg-Python-Mod07.git

Assignment 08 – The Final

Intro

For the final we were to take the provided starter "bare bones" of a program and build upon it to create a product and price data list tool. We would end up using everything we have learned thus far in the course. Creating lists, saving to files, reading data from files, presenting information to a user, taking inputs from a user, organize our code with classes and functions.

Data

We started with new techniques learned in week 8 of using a data class with constructor, properties, and setter methods. As discussed in the week 08 lecture when you print your object from the data class you will receive the address (Figure 1). In Figure 2 we correct this by making a *to_string* function to return the data we want to see instead of the data address. After Figure 2 was snipped we also created a *__str__* function which calls the *to_string* function and converts the default setting of the *__str__* function.

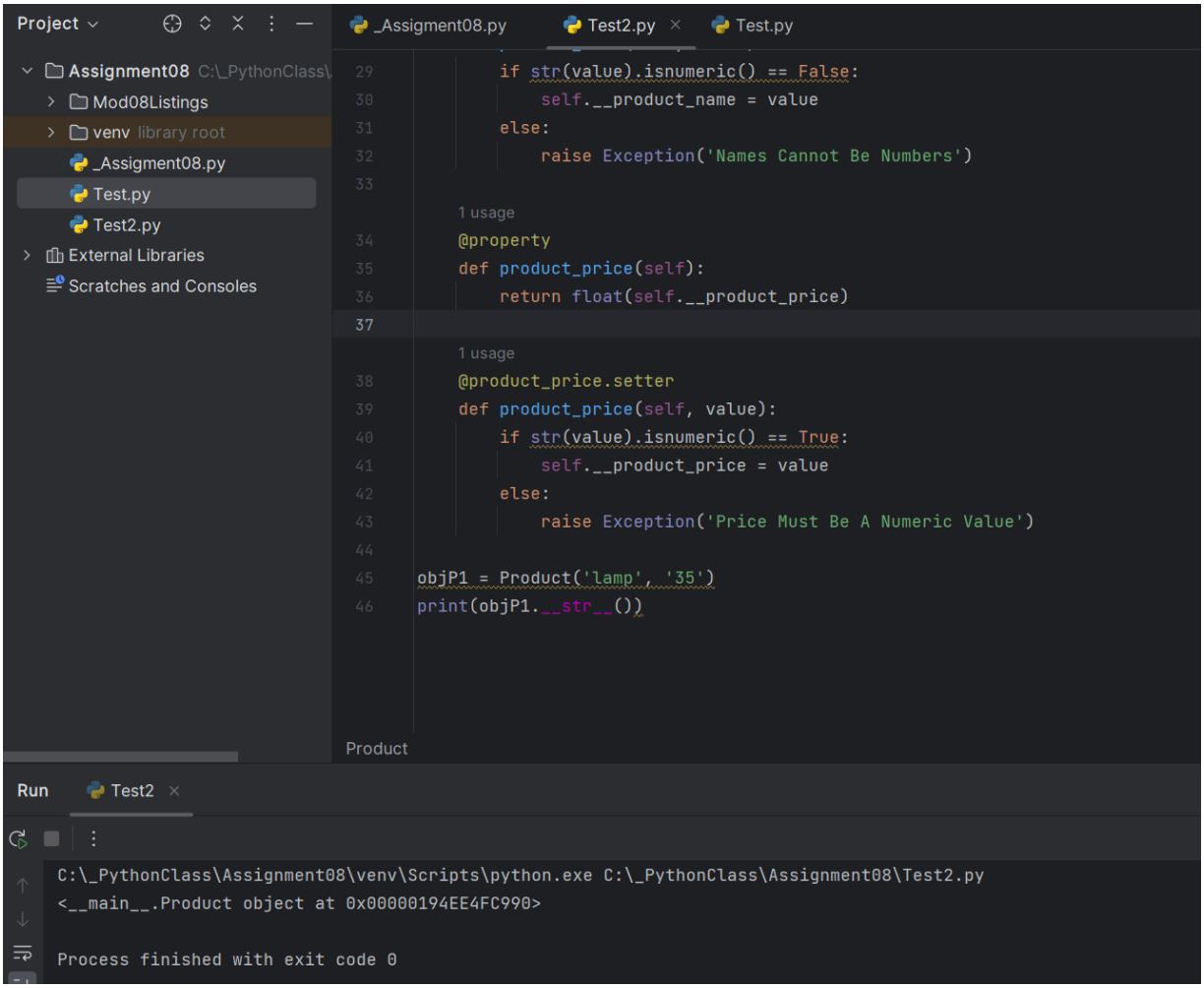

Figure 1: Trying to print the product and price but receiving the memory address of the object as explained in the module 08 lecture.

Figure 2: Showing the difference in returning information from the *'to_string'* function and the default *__str__* function in our Product class.

From the starter script it was indicated that the price should be a float value. This introduced a challenge when it came to the error handling for it, since *isnumeric()* doesn't work for a float value. Searching through the internet I found what I thought would be the solution with a *isdigit()* function shown in Figure 3. But as shown in Figure 4 this still gave me an error when the user uses a decimal in their price value.

```
        2 usages
34      @property
35      def product_price(self):
36          return float(self.__product_price)
37
        1 usage
38      @product_price.setter
39      def product_price(self, value):
40          if value.isdigit() == True:
41              self.__product_price = value
42          else:
43              raise Exception('Price Must Be A Numeric Value')
44
        2 usages
45      def to_string(self):
46
47          object_data_csv = self.product_name + "," + str(self.product_price)
48          return object_data_csv
49
50      def __str__(self):
51          return self.to_string()
52
53  product = input("Enter product: ")
54  price = (input('Enter Price: '))
55  objP1 = Product(product,price)
56  print(objP1.__str__())
57  print(objP1.to_string())
```

Figure 3: Trying to get float to work with my error handling using *isdigit()* on line 40. Unable to find a similar function as *isnumeric()* for a float check.

```
C:\_PythonClass\Assignment08\venv\Scripts\python.exe C:\_PythonClass\Assignment08\Test2.py
Enter product: lamp
Enter Price: 10.99
Traceback (most recent call last):
  File "C:\_PythonClass\Assignment08\Test2.py", line 55, in <module>
    objP1 = Product(product,price)
            ^^^^^^^^^^^^^^^^^^^^^^
  File "C:\_PythonClass\Assignment08\Test2.py", line 21, in __init__
    self.product_price = product_price
    ^^^^^^^^^^^^^^^^^^
  File "C:\_PythonClass\Assignment08\Test2.py", line 43, in product_price
    raise Exception('Price Must Be A Numeric Value')
Exception: Price Must Be A Numeric Value

Process finished with exit code 1
```

Figure 4: Error received when using *isdigit()* function in Figure 3 code.

Since that didn't work I did more searching and discovered a very handy function *isinstance()*. This function takes an object and a parameter to check your object against. In Figure 5 you'll see I take the variable *'value'* and check that it as a float value.

I also have a try/except block for setting the *'value'* given to the *product_price* function to a float value. This provides two levels of error handling. I could probably get rid of one of the layers of error handling, but it works and adds some redundancy.

```
1 usage
@product_price.setter
def product_price(self, value):
    try:  #setting the product price to a float and error handling if user entered a non-numeric value
        fltval = float(value)
        if isinstance(fltval, float) == True:
            self.__product_price = value
        else:
            raise Exception('Price Must Be A Numeric Value')
    except:
        raise Exception ('Price Must Be A Numeric Value')
```

Figure 5: A try/except combined with *isinstance()* used to provide error handling if *'value'* was not a float value.

Main Script

In my main script body, I couldn't figure out why my *IO.input_product_data()* function wasn't returning the variables it is supposed to. It took me a while and some looking back at our past modules to realize that you need to assign the variables that will be returning when you call the function, shown in Figure 6 on line 232.

```
230     # Let user add data to the list of product objects
231     elif choice_str == '2':
232         prod, price = IO.input_product_data()
233         print(prod, price, 'have been added to list')
234 
235         FileProcessor.add_data_to_list(prod=prod, price=price, lstOfProductObjects=lstOfProd
236 
```

Figure 6: For a while I couldn't figure out why my values were not returning but then found that I need to assign the variables for what is returning in front of the function call shown here on line 232.

Run it!

Once all the code was added in the required fields, we run the program in Pycharm and in the Command Line. We also made sure to enter values that would "break" the program to check our error handling features.

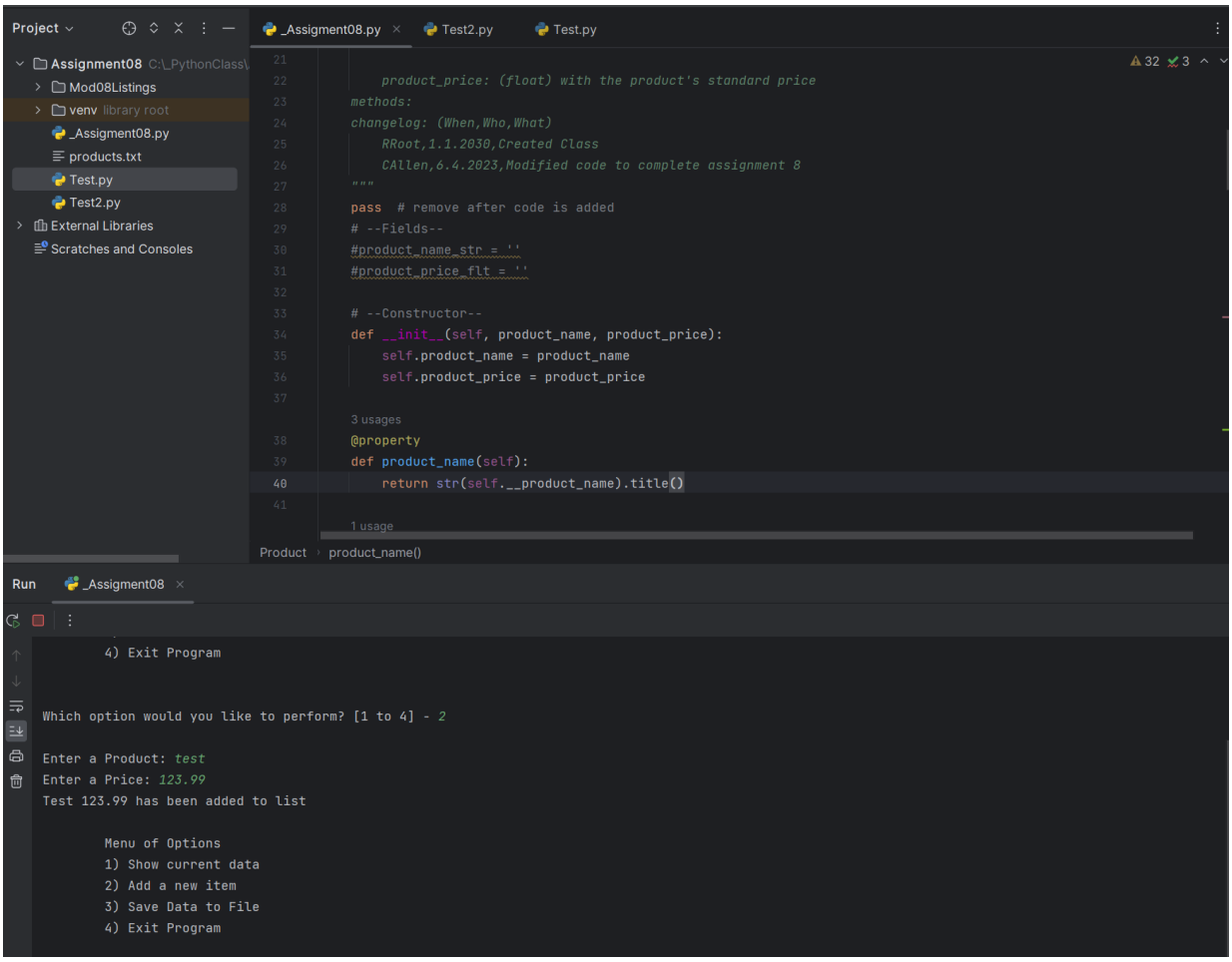

Figure 7: Adding and item to our list and the program confirming the item and price have been added.

One thing I noticed was although I had exception error handling, I didn't like that the program didn't just return you to the main menu. In Figure 8, I experimented with just a print prompt, instead of a raise exception, in the hopes it would prompt the user with the message and send you back to the main body. Unfortunately, that didn't work either, so I scraped that and kept the raise exception code.

In the future it would be nice to add some code for sending the user back to main menu instead of stopping the program with the error message.

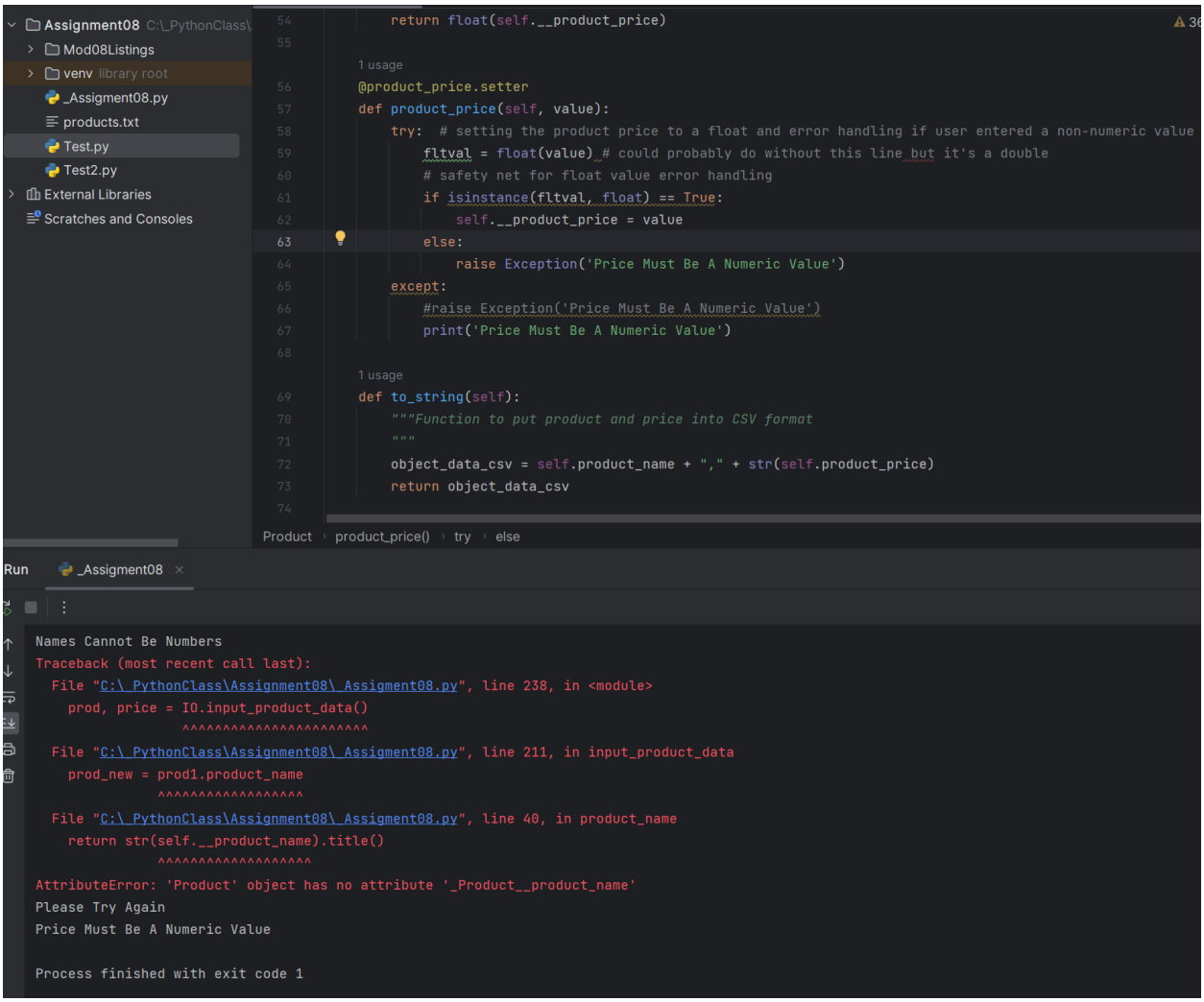

Figure 8: An attempt to use a print prompt in our try/except block to send the user back to the main menu after *product_price_setter* failed.

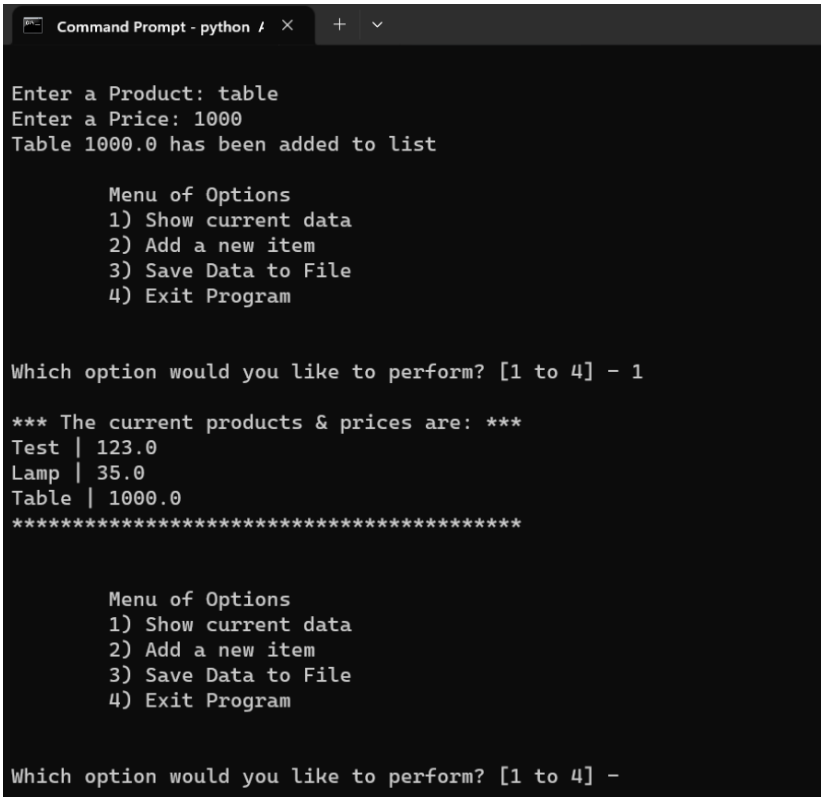

Figure 9: Running our program in the command line.

Summary

Since this assignment built upon all that we have done in this entire course it was fairly straightforward of what to add to the starter script. Even better was that this script very closely resembled our Assignment06. There were only a couple added complexities, one being utilizing the new Data Class with its getter & setter. The other being able to adjust our script for the use of a float value.

For me I find it easier to start with at least something rather than a blank page. From there you can pull from past modules and experience to put together what you need to get the outcome that is required for you program.