

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.

```
Project ▾
  ▾ Assignment08 C:\PythonClass\
    ▾ Mod08Listings
    ▾ venv library root
      _Assignment08.py
      Test.py
      Test2.py
    ▾ External Libraries
    ▾ Scratches and Consoles

29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46

if str(value).isnumeric() == False:
    self.__product_name = value
else:
    raise Exception('Names Cannot Be Numbers')

1 usage
@property
def product_price(self):
    return float(self.__product_price)

1 usage
@product_price.setter
def product_price(self, value):
    if str(value).isnumeric() == True:
        self.__product_price = value
    else:
        raise Exception('Price Must Be A Numeric Value')

objP1 = Product('Lamp', '35')
print(objP1.__str__())

Product

Run Test2 ×

C:\PythonClass\Assignment08\venv\Scripts\python.exe C:\PythonClass\Assignment08\Test2.py
<__main__.Product object at 0x00000194EE4FC990>

Process finished with exit code 0
```

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

```
1 usage
45 def to_string(self):
46
47     object_data_csv = self.product_name + "," + str(self.product_price)
48     return object_data_csv
49
50 objP1 = Product('lamp', '35')
51 print(objP1.__str__())
52 print(objP1.to_string())
```

Run Test2 x

C:_PythonClass\Assignment08\venv\Scripts\python.exe C:_PythonClass\Assignment08\Test2.py

<__main__.Product object at 0x00000221A672CD90>

Lamp,35.0

Process finished with exit code 0

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.

The screenshot displays a Python IDE with a project named 'Assignment08'. The left sidebar shows the project structure, including a 'venv' directory and files like '_Assigment08.py', 'products.txt', 'Test.py', and 'Test2.py'. The main editor window shows the code for '_Assigment08.py', which includes a class definition for a product management system. The code includes comments, a changelog, and a constructor. The 'product_name' property is defined with a getter method. The bottom panel shows the program's execution output, which includes a menu of options and a confirmation message for adding a new item.

```
21
22     product_price: (float) with the product's standard price
23 methods:
24 changelog: (When,Who,What)
25     RRoot,1.1.2030,Created Class
26     CAllen,6.4.2023,Modified code to complete assignment 8
27 """
28 pass # remove after code is added
29 # --Fields--
30 #product_name_str = ''
31 #product_price_flt = ''
32
33 # --Constructor--
34 def __init__(self, product_name, product_price):
35     self.product_name = product_name
36     self.product_price = product_price
37
38 3 usages
39 @property
40 def product_name(self):
41     return str(self.__product_name).title()
42
43 1 usage
Product : product_name()
```

Run _Assigment08

```
4) Exit Program
Which option would you like to perform? [1 to 4] - 2
Enter a Product: test
Enter a Price: 123.99
Test 123.99 has been added to list

Menu of Options
1) Show current data
2) Add a new item
3) Save Data to File
4) Exit Program
```

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 scrapped 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.

The screenshot displays a Python IDE with a file explorer on the left and a code editor on the right. The file explorer shows a project named 'Assignment08' with subfolders 'Mod08Listings' and 'venv library root'. Files include '_Assignment08.py', 'products.txt', 'Test.py', and 'Test2.py'. The code editor shows the following Python code:

```
54         return float(self.__product_price)
55
56     1 usage
57     @product_price.setter
58     def product_price(self, value):
59         try: # setting the product price to a float and error handling if user entered a non-numeric value
60             fltval = float(value) # could probably do without this line but it's a double
61             # safety net for float value error handling
62             if isinstance(fltval, float) == True:
63                 self.__product_price = value
64             else:
65                 raise Exception('Price Must Be A Numeric Value')
66         except:
67             #raise Exception('Price Must Be A Numeric Value')
68             print('Price Must Be A Numeric Value')
69
70     1 usage
71     def to_string(self):
72         """Function to put product and price into CSV format
73         """
74         object_data_csv = self.product_name + "," + str(self.product_price)
75         return object_data_csv
```

The bottom of the image shows a 'Run' window with the following error message:

```
Names Cannot Be Numbers
Traceback (most recent call last):
  File "C:\PythonClass\Assignment08\Assignment08.py", line 238, in <module>
    prod, price = IO.input_product_data()
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\PythonClass\Assignment08\Assignment08.py", line 211, in input_product_data
    prod_new = prod1.product_name
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "C:\PythonClass\Assignment08\Assignment08.py", line 40, in product_name
    return str(self.__product_name).title()
    ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
AttributeError: 'Product' object has no attribute '__product_name'
Please Try Again
Price Must Be A Numeric Value
Process finished with exit code 1
```

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.

```
Command Prompt - python / x + v

Enter a Product: table
Enter a Price: 1000
Table 1000.0 has been added to list

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Save Data to File
    4) Exit Program

Which option would you like to perform? [1 to 4] - 1

*** The current products & prices are: ***
Test | 123.0
Lamp | 35.0
Table | 1000.0
*****

    Menu of Options
    1) Show current data
    2) Add a new item
    3) Save Data to File
    4) Exit Program

Which option would you like to perform? [1 to 4] -
```

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 your program.