How to Use Custom Classes to Create and Edit a File

# Understanding the Task

Assignment 08 requires the program to collect product and price information using custom classes and static methods. The program gives the user the option to add data, display data, and save/exit the program. The data is saved to a text file. A starter python file is provided. The assignment requires this code to be run in PyCharm and the OS console window.

# Creating the Script

For this program, I created the code in PyCharm using the Assignment08 starter file provided and used the pseudo-code to guide me through creating the code for the various tasks the program is supposed to perform. First, I created a “Product” class to collect and store the product and price information. The properties of this class include a the product\_name and product\_price. These are initialized through the first function using the \_\_init\_\_ function and including “self” in the function arguments. The product name and price are accessed and set through a “getter” and “setter”. The script includes error handling to check if the product name does not include numbers and the product price does not include letters and numbers and an error is raised if necessary. A string method is included to return the product name and price separated by a comma. The Product Class is shown in Figure 1.

*Figure 1. Product Class*

class Product:def \_\_init\_\_(self, product\_name = **""**, product\_price= 0.0):self.\_\_product\_name = product\_name  
 self.\_\_product\_price = product\_price@property  
 def product\_name(self): *# getter* return str(self.\_\_product\_name)  
 @product\_name.setter  
 def product\_name(self, name):  
 if str(value).isnumeric() == False:  
 self.\_\_first\_name = name  
 else:  
 raise Exception(**"Names cannot be numbers"**)@property  
 def product\_price(self): *# getter or acessor* return str(self.\_\_product\_price)  
 @product\_price.setter  
 def product\_price(self, price):  
 if str(value).isnumeric() == True:  
 self.\_\_first\_name = price  
 else:  
 raise Exception(**"The value should be a number"**)def \_\_str\_\_(self):  
 return self.product\_name + **', '** + self.product\_price

A menu “Menu of Options” is displayed in the main body of the script within a while loop and is displayed after each choice the user makes. The while loop continues to display these options until the user chooses user exits the program. The functions to complete these tasks are contained within the “FileProcessor” class (performing tasks with the data) and the “IO” (taking inputs and providing outputs) classes. The menu of options appears as follows:

Menu of Options

1) Show Current Data

2) Add data

3) Save Data to File

## Initial Processing

The code to initially import the Product List into a “list of rows” or “lstOfProductObjects” to work with the data was already pre-written into the starter module file. This is accomplished by calling the “read\_data\_from\_file()” function within the “FileProcessor” class.

## Option 1: Showing Current Data

The current data is displayed if the user selects option 1. The current “list of product objects” is loaded outside the while loop by calling on the following function within the File Processor class:

lstOfProductObjects = FileProcessor.read\_data\_from\_file(strFileName, lstOfProductObjects)

The read\_data\_from\_file() function reads the rows of lines in the file and reads each object within the row split by a comma. The Product() class is used to assign these values to the “product\_name” and “product\_ price”. The entire list of rows is returned.

*Figure 2. Product Class*

def read\_data\_from\_file(file\_name, list\_of\_rows):try:  
 file\_name = open(file\_name, **"x"**)  
 file\_name.close()  
 except FileExistsError:  
 pass  
  
 file = open(file\_name, **"r"**)  
 for line in file:  
 product\_name, product\_price = line.split(**","**)  
 row = Product(product\_name, product\_price)  
 list\_of\_rows.append(row)  
 file.close()  
 return list\_of\_rows

The rows with the list are displayed with a simple “for” loop as shown in Figure 3.

*Figure 3. Product Class*

*# Show user current data in the list of product objects*if choice.strip() == **'1'**:  
 for row in lstOfProductObjects:  
 print(row)  
 continue

## Option 2: Adding New Data

New data is added to the list table by collecting new user inputs for the task and priority from the input\_new\_task\_and\_priority() function in the IO class. The values are assigned to the “product” and “price” in the Product() class through the add\_data\_to\_list() function in the Processor class. The new dictionary row is appended to the list table. The script for Option 2 is shown in Figure 4 and the IO and Processing functions are shown in Figures 5 and 6, respectively.

*Figure 6. Function to add new data to the list of rows*

*Figure 5. New product and price collection function*

@staticmethod  
def add\_data\_to\_list(name, price, list\_of\_rows):  
 row = Product(name, price)  
 list\_of\_rows.append(row)  
 return list\_of\_rows

def input\_new\_product\_and\_price():  
 product = input(**"Enter the product: "**)  
 if product.isalpha() == True:  
 pass  
 else:  
 raise ValueError  
 price = input(**"Enter the price: "**)  
 if price.isnumeric() == True:  
 pass  
 else:  
 raise ValueError  
 return product, price

*Figure 4. Option 2 while loop script*

*# Let user add data to the list of product objects*elif choice.strip() == **'2'**:  
 product, price = IO.input\_new\_product\_and\_price()  
 lstOfProductObjects = FileProcessor.add\_data\_to\_list(product, price, lstOfProductObjects)  
 continue

## Option 3: Saving Data and Exiting

The list table data can be saved to the “products.txt” file by selecting Option 3. The list table is passed to the write\_data\_to\_file() function in the Processor class and the data is written to the text file. This function writes the “product\_name” and “product\_price” to the individual rows of the file, separated by a comma. The program breaks the while loop and exits the program after the data is saved. The script for Option 3 is shown in Figure 7 and and the write\_data\_to\_file() functions are shown in Figures 8.

*Figure 8. Writing data to the file*

*# TODO: Add Code to process data to a file*def write\_data\_to\_file(file\_name, list\_of\_rows):  
 file = open(file\_name, **"w"**)  
 for row in list\_of\_rows:  
 file.write(row.product\_name + **","** + row.product\_price + **"**\n**"**)  
 file.close()  
 return list\_of\_rows

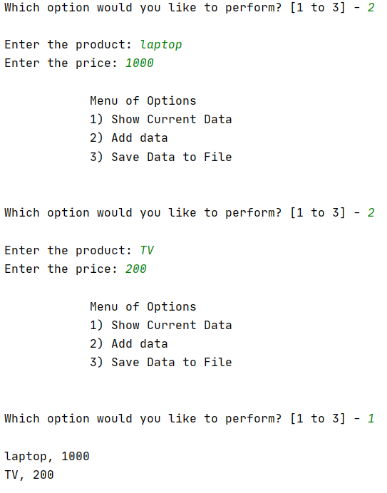
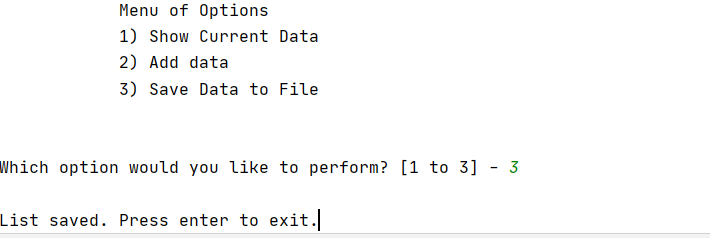
*Figure 7. Option 3 Script*

elif choice.strip() == **'3'**:  
 lstOfProductObjects = FileProcessor.write\_data\_to\_file(strFileName, lstOfProductObjects)  
 input(**"List saved. Press enter to exit."**)  
 break *# exit*else:  
 print(**"Invalid Option"**)

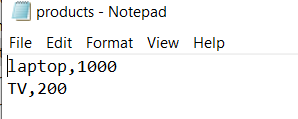
# Running the Code

To run the code, I first tested the code in PyCharm. I ran the program, followed the prompts and verified that my inputs were written to the products.txt file. The program worked in PyCharm as shown in Figure 9 (images are labeled 1-2 to show the order of steps taken) and wrote the final list to the products.txt file as shown in Figure 10.

2



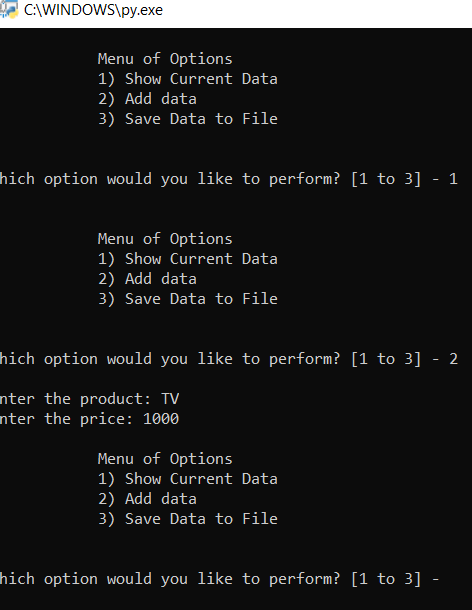
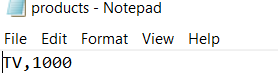
1



*Figure 10. products.txt output from running Assignment0.Py in PyCharm*

*Figure 9. Assignment 08 running PyCharm*

After testing the code in PyCharm, I then ran the same program in the command console. The program can be seen running in Figure 11 and the output can be seen in Figure 12. I performed only one option just to demonstrate that the program can be run in the command console.



*Figure 12. Assignment08 running in the console window*

*Figure 11. Assignment08 running in the console window*

# Summary

To complete this assignment, I used the starter file for Assignment 08 provided and edited the script in PyCharm. I followed the prompts laid out in the pseudo-code and comments to accomplish the assignment tasks. A while loop in the main body of the script was used to give the user the options to display, add, and save data. The products and their prices were assigned through the properties of a custom class. The tasks were accomplished through various functions contained within the IO and Processor classes. This assignment demonstrated how to create custom classes with more advanced properties and to properly separate the various methods of a program.