Jace Riley

Randal Root

IT FDN 100 A

11/26/19

**Classes**

**Introduction**

This assignment was similar to assignment 6 with the addition of a constructor. This did add a different level of complexity to the creation of the code, but also allowed for decent organization.

**Creating Classes/Objects**

Due to the assignments structure, it was very similar to assignment 6, but starting with less information. It was nice to go from the pseudo code to the final product of the code. The first step I took was to write the actual code that would be processing data. The reason that I did this was because it would make it a lot easier to write the code within the classes if I knew what the final product would be. I wasn’t able to write all of the code, so I wrote pseudo code in its place.

Oncee I finished the main body of the code that would be processed I defined a constrictor within the Products class and used that in the FileProcessing class as seen in Figure 1.

class Product:  
 *"""Stores data about a product:  
  
 properties:  
 product\_name: (string) with the products's name  
 product\_price: (float) with the products's standard price  
 methods:  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 JRiley,11.26.2019,Modified code to complete assignment 8  
 """* def \_\_init\_\_(self, ProductName, ProductPrice):  
 self.Name = ProductName  
 self.Price = ProductPrice

**Figure 1.**

I then created two functions within File processing code. As you can see in Figure 2, I utilized the Product class within the ReadDataFromFile function.

class FileProcessor:  
 *"""Processes data to and from a file and a list of product objects:  
  
 methods:  
 save\_data\_to\_file(file\_name, list\_of\_product\_objects):  
  
 read\_data\_from\_file(file\_name): -> (a list of product objects)  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 JRiley,11.26.2019,Modified code to complete assignment 8  
 """* @staticmethod  
 def ReadDataFromFile(strFileName):  
 *""" Reads data utilizing the Product Class* ***:return****: string  
 """* lstOfProductObjects = []  
 file = open(strFileName, "r")  
 for line in file:  
 data = line.split(",")  
 objProduct = Product(data[0].strip(), float(data[1].strip()))  
 row = {"Name":objProduct.Name, "Price":objProduct.Price}  
 lstOfProductObjects.append(row)  
 file.close()  
 return lstOfProductObjects  
  
 @staticmethod  
 def SaveDataToFile(filename, lstOfProductObjects):  
 *""" Saves/appends product name and price to a file* ***:return****: string  
 """* with open(filename, 'a') as file:  
 for row in lstOfProductObjects:  
 file.write(row["Name"] + ", " + str(row["Price"]) + "\n")  
 return "Data saved to file."

Figure 2

I then created four functions within the IO class. These are shown in figure 3.

class IO:  
 *""" Presenting user a menu, user option, reading current data, and adding data...  
  
 Methods:  
 Menu: Showing the menu of options for the user  
 Choice: Receiving the choice option the user input  
 ReadData: Reading the current data in the file to the user  
 AddData: Receiving new data to be added from the user input  
  
 Changelog:  
 JRiley,11.26.2019,Modified code to complete assignment 8  
 """* @staticmethod  
 def OutputMenuItems():  
 *""" Display a menu of choices to the user* ***:return****: string  
 """* print('''  
 Menu of Options  
 1) Show current data  
 2) Add a new item.  
 3) Save data to file and exit program  
 ''')  
 print() # Add an extra line for looks  
  
 @staticmethod  
 def InputMenuChoice():  
 *""" Gets the menu choice from a user* ***:return****: string  
 """* choice = str(input("Which option would you like to perform? [1 to 3] - ")).strip()  
 print() # Add an extra line for looks  
 return choice  
  
 @staticmethod  
 def ShowCurrentData(self):  
 *""" Shows the current items in the file* ***:param*** *lstOfProductObjects: (list) of rows you want to display* ***:return****: nothing  
 """* print("The current items in the file are: \*\*\*\*\*\*\*\*\*\*\*\n")  
 lstOfProductObjects = FileProcessor.ReadDataFromFile(self)  
 for row in lstOfProductObjects:  
 print(row["Name"] + ", $" + str(row["Price"]))  
 print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  
  
 @staticmethod  
 def GetProductData(lstOfProductObjects):  
 *""" Gives the prompt to the user  
 input: Product and Price of product.  
 """* newProduct = input("Add a product: ")  
 newPrice = float(input("Add a price for the product: "))  
 row = {"Name": newProduct, "Price": newPrice}  
 lstOfProductObjects.append(row)  
 return lstOfProductObjects  
 print()

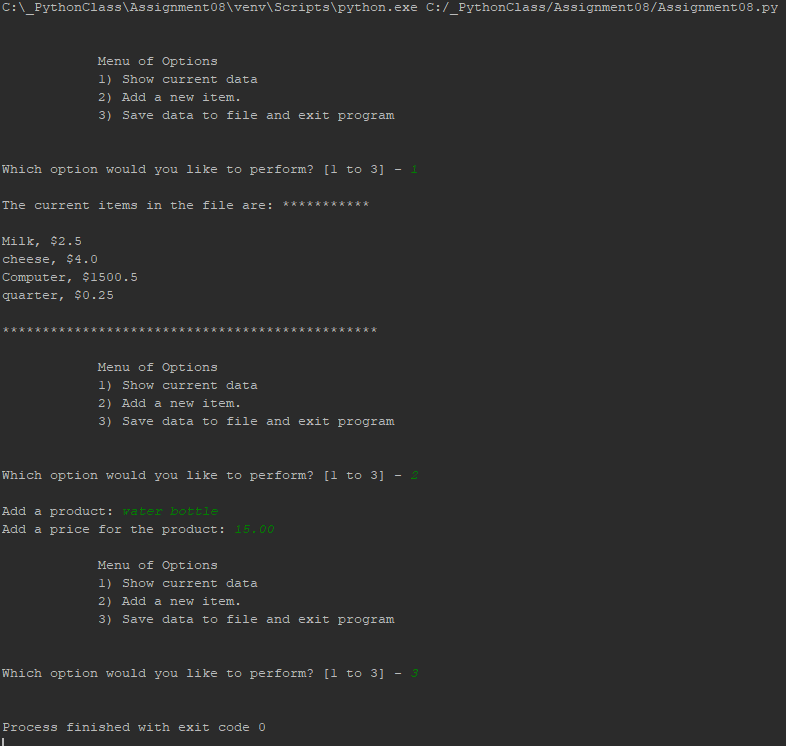
**Figure 3.**

Each of these functions were to serve the purpose as stated in the pseudo code of the assignment starter code provided.

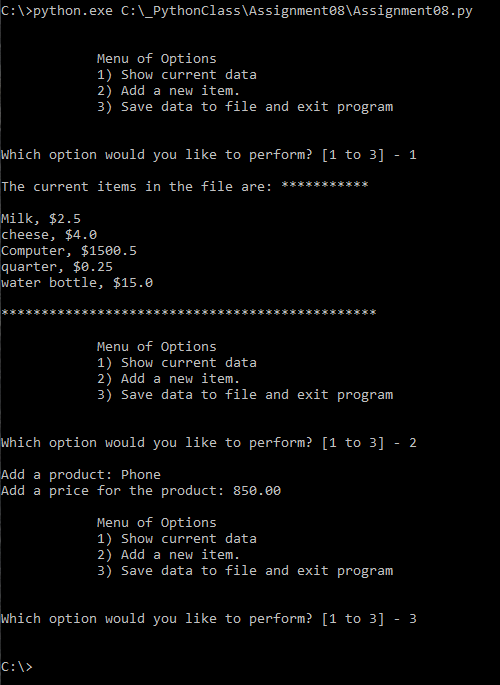
The remainder of the code was the main body of the code, that called these functions and classes. I changed the pseudo code that I wrote to meet the new functions and class names that I created. The code is shown in Figure 4 followed by the output of that code in Figure 5. The same result will be shown while running in OS Command in Figure 6.

while True:  
 # Show the menu of options to the user  
 IO.OutputMenuItems()  
 # Receive user's choice from menu  
 option = IO.InputMenuChoice()  
  
 # Option 1 is to show the current data within the file.  
 if option == '1':  
 IO.ShowCurrentData(strFileName)  
  
 # Option 2 is to have the user add data to the file.  
 elif option == '2':  
 lstOfProductObjects = IO.GetProductData(lstOfProductObjects)  
  
 # Option 3 is to allow the user to either save the data to a text file or exit without saving.  
 elif option == '3':  
 FileProcessor.SaveDataToFile(strFileName, lstOfProductObjects)  
 break  
 else:  
 try:  
 raise TypeError("Invalid Option")  
 except TypeError as te:  
 print(te, "\nChoose 1 to 3 only.")

**Figure 4**



**Figure 5**



**Figure 6**

**Summary**

This exercise utilized creating classes/objects and functions that were to be used in a code to add a product and a price. It was nice to be able to learn this utilizing similar code to other assignments. The reason this is nice is because it allows us to focus on the purpose and meaning behind the code, rather than rack our brains on how to apply it to code.