Tao Ye

June 3, 2020

Foundations of Programming: Python

Assignment 08

https://github.com/bubblecfd/IntroToProg-Python-Mod08

Python Script with Objects and Classes

Introduction

The objective of this assignment is to create Python script with classes and objects for abstraction and encapsulation. Three classes are defined: one class is for storing the data and is used indirectly by creating class objects that hold data with different values; the other two classes are for processing data or I/O and are used directly with static methods. The main application of this script is to maintain a product list through interactions with the user. The user adds product and its price to a list, and save the list to a file. The user can also update the list later at any time. The script will be tested and run in both PyCharm and a Windows command shell as a console application.

Editing and Debugging Code in PyCharm

For this assignment, I created a new folder **C:\\_PythonClass \Assignment08**. Then in PyCharm, I created a new project that uses the **C:\\_PythonClass\Assignment08** folder as its location,and added the starter Python script and renamed to **Assigment08.py**.

**Class Definitions**

For this assignment, the following three classes are defined.

1. Product data class

This is a class for storing data with different values. The constructor **\_\_init\_\_(self)** defines two private attributes **product\_name** (string) and **product\_price** (floating number), and assigns initial values for them. For these two attributes, getter and setter properties are defines to access and set their values to achieve data encapsulation. The class definition is shown below.

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  
 Tao Ye,6.2.2020,Modified code to complete assignment 8  
 """* pass  
  
 *# Constructor -----------------* def \_\_init\_\_(self):  
 self.\_\_product\_name = **""** self.\_\_product\_price = 0.0  
  
 *# Properties ------------------  
 # product\_name* @property *# (getter or accessor)* def product\_name(self):  
 return str(self.\_\_product\_name)  
  
 @product\_name.setter *# (setter or mutator)* def product\_name(self, name):  
 self.\_\_product\_name = name  
  
 *# product\_price* @property *# (getter or accessor)* def product\_price(self):  
 return self.\_\_product\_price  
  
 @product\_price.setter *# (setter or mutator)* def product\_price(self, value):  
 self.\_\_product\_price = value

2. File processing class

This class is for data processing with a file. So it contains only static methods for reading and writing the product list from and to the file. The class methods are used directly. In the method **read\_data\_from\_file()**, I added a **try-except** to capture the file-not-found exception to avoid the abrupt termination of the execution. And I discovered that the empty list (“[]”) still has to be returned even though the file is not found; otherwise the argument **lstOfProductObjects** would become “none” when the **read\_data\_from\_file()** method returns. The code would then terminate with errors later when it tries to append new elements to a “none” list. The class definition is shown below.

class FileProcessor:  
 *"""Processes data to and from a file and a list of product objects:  
  
 methods:  
 save\_data\_to\_file(file\_name, list\_of\_objects):  
 read\_data\_from\_file(file\_name): -> (a list of product objects)  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 Tao Ye,6.3.2020,Modified code to complete assignment 8  
 """* pass  
  
 @staticmethod  
 def read\_data\_from\_file(file\_name):  
 *"""  
 Desc - Reads data from a file into a list of products  
  
 :param file\_name: (string) with name of file  
 :return: (list) of product objects  
 """* list\_of\_objects=[]  
 try:  
 file = open(file\_name, **"r"**)  
 for line in file:  
 objProduct = Product() *# create an object of Product class* data = line.split(**","**) *# data has the list of name and price strings* objProduct.product\_name = data[0]  
 objProduct.product\_price = float(data[1])  
 list\_of\_objects.append(objProduct) *# add the object to the list* file.close()  
 return list\_of\_objects  
 except FileNotFoundError:  
 print(**"File"**, file\_name, **"does not exist yet."**)  
 return list\_of\_objects  
  
 @staticmethod  
 def save\_data\_to\_file(file\_name, list\_of\_objects):  
 *"""  
 Desc - Save a list of products to a file  
  
 :param file\_name: (string) with name of file  
 :param list\_of\_objects: (list) you want to save to the file  
 """* file = open(file\_name, **"w"**)  
 for objProduct in list\_of\_objects:  
 file.write(objProduct.product\_name + **","** + str(objProduct.product\_price) + **"**\n**"**)  
 file.close()

The product list is saved in the file “products.txt” as shown in Figure 1. Each line is a product name with a price value. When reading from this file, the product name and price value are read into a list called **data** with 2 elements. The two elements are assigned to the **product\_name** and **product\_price** attributes of a **Product** class object, respectively. The **Product** class object is then added to a list.

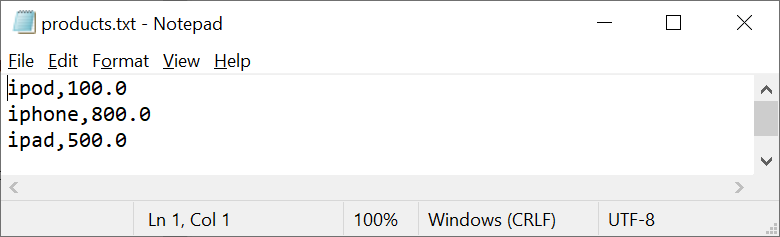


Figure 1. products.txt

3. I/O processing class

The third class defined is a class to handle all interactions with users. This is a class that also contains only static methods for direct use. The definition of this class is listed below.

class IO:  
 *""" A class for perform Input and Output  
  
 methods:  
 OutputMenuItems():  
 InputMenuChoice():  
 ShowCurrentObjectsInList(list\_of\_objects):  
 InputProductAndPrice():  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2030,Created Class  
 Tao Ye,6.3.2020,Modified code to complete assignment 8  
 """* pass  
  
 @staticmethod  
 def OutputMenuItems():  
 *""" Display a menu of choices to the user  
 :return: nothing  
 """* print(**'''  
 Menu of Options  
 1) Show current data  
 2) Add a new product data.  
 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 ShowCurrentObjectsInList(list\_of\_objects):  
 *""" Shows the current list of products  
  
 :param list\_of\_objects: (list) of product objects you want to display  
 :return: nothing  
 """* print(**"\*\*\*\*\*\*\* The current product lists are: \*\*\*\*\*\*\*"**)  
 for objProduct in list\_of\_objects:  
 print(objProduct.product\_name + **" (price: "** + str(objProduct.product\_price) + **")"**)  
 print(**"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"**)  
 print() *# Add an extra line for looks* @staticmethod  
 def InputProductAndPrice():  
 *""" Ask the user to enter product name and price  
  
 :param none:  
 :return: (string) product name and price  
 """* product = input(**"What is the product? - "**).strip()  
  
 while True: *# Ask user for price as a floating number* try:  
 price = float(input(**"What is the price? - "**).strip())  
 break  
 except ValueError:  
 print(**"Price must be a floating number; try again..."**)  
 continue  
 return product, price

The method **OutputMenuItems()** displays a simple user interface menu providing several functions to work with the product list. It shows user current data in the list of product objects, lets user add data to the list of product objects, and lets user save current data to file and exit program. The **InputMenuChoice()** asks the user for the selection in the menu to perform the above-mentioned the operations. The method **ShowCurrentObjectsInList(list\_of\_objects)** displays the content of the list of the product class objects. It prints out the **product\_name** and **product\_price** attributes of **Product** class objects in the list. The method **InputProductAndPrice()** lets the user to input the name and price of a new product to be added to the list. It has the error handling mechanism to accept only floating numbers for the price value. It uses an infinite while loop to continue asking until the user enters a number.

**Main Body**

In the main body of the script, it first calls the **FileProcessor.read\_data\_from\_file()** method to load the data in the file to the list **lstOfProductObjects** before displaying the menu for selection. Depending on the user selection, the methods in class “IO” or “FileProcessor” described above are called accordingly. In the process, the **product\_name** and **product\_price** attributes of **Product** class objects are accessed or set for data processing. The main body script is listed below

*# When the program starts, Load data from products.txt.  
# If the file already exists, load data from the file*lstOfProductObjects = FileProcessor.read\_data\_from\_file(strFileName)  
  
*# Display a menu of choices to the user*while True:  
 IO.OutputMenuItems() *# Shows menu* strChoice = IO.InputMenuChoice() *# Get menu option  
  
 # Process user's menu choice* if (strChoice.strip() == **'1'**): *# Show current list of product objects* IO.ShowCurrentObjectsInList(lstOfProductObjects)  
 continue *# to show the menu* elif (strChoice.strip() == **'2'**): *# Add a new product to the list of the objects  
 # Ask user for new product and price* strProduct, floatPrice = IO.InputProductAndPrice()  
 print() *# Add an extra line for looks  
  
 # Add new object to the List* objProduct = Product() *# A new product object* objProduct.product\_name = strProduct *# setter property for product\_name* objProduct.product\_price = floatPrice *# setter property for product\_price* lstOfProductObjects.append(objProduct) *# add the new object to the list  
  
 # Show current list of product objects* IO.ShowCurrentObjectsInList(lstOfProductObjects)  
 continue *# to show the menu* elif (strChoice.strip() == **'3'**): *# save the data to the file and exit* FileProcessor.save\_data\_to\_file(strFileName, lstOfProductObjects)  
 input(**"Data saved to file! Press the [Enter] key to return to exit."**)  
 break *# exit* else:  
 print(**"Invalid choice; try again..."**)  
 continue

Run the Script

1. Run the script in PyCharm

The first time it is run, the option 2 is selected twice just to create a list with two products (Figure 2):

ipod, 100.0

iphone, 800.0

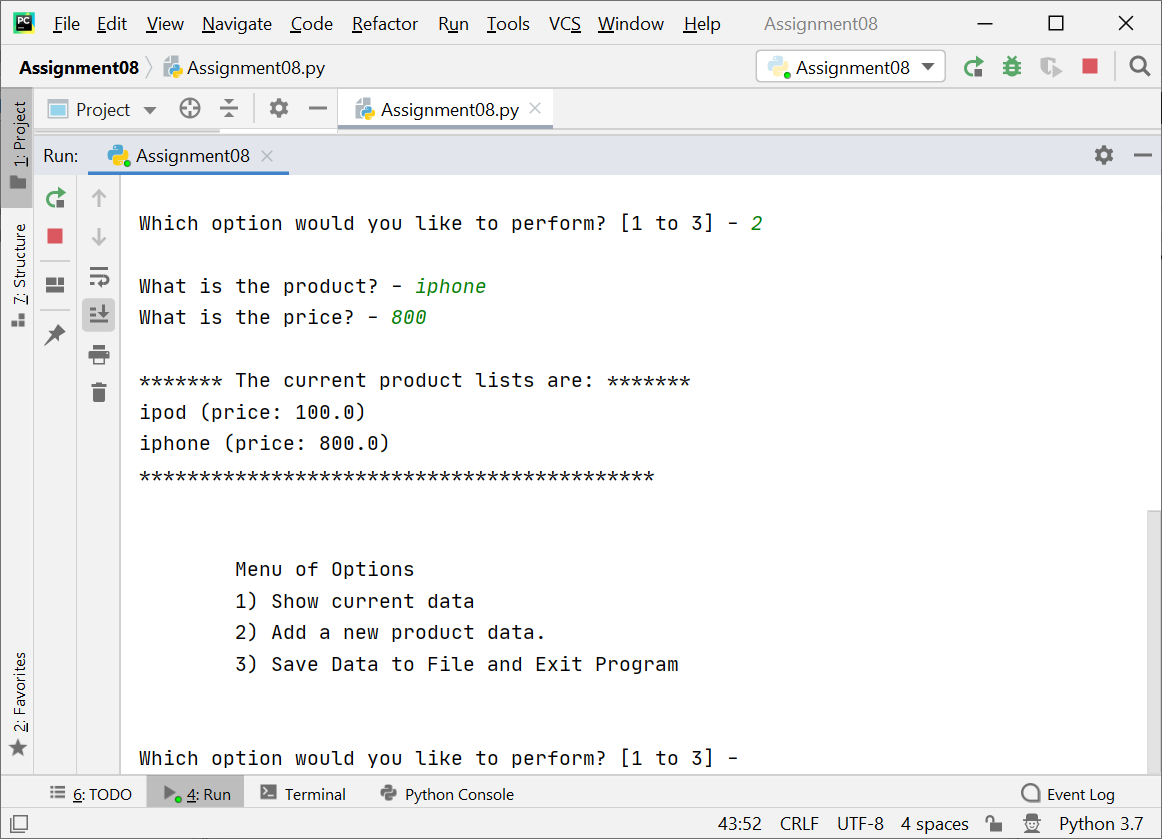


Figure 2. Initial List with Two Products

The list is then saved to a file to create the file and the program exits (Figure 3). The file “products.txt” now appears in the same folder as the script (Figure 4) and its content is shown in Figure 5.

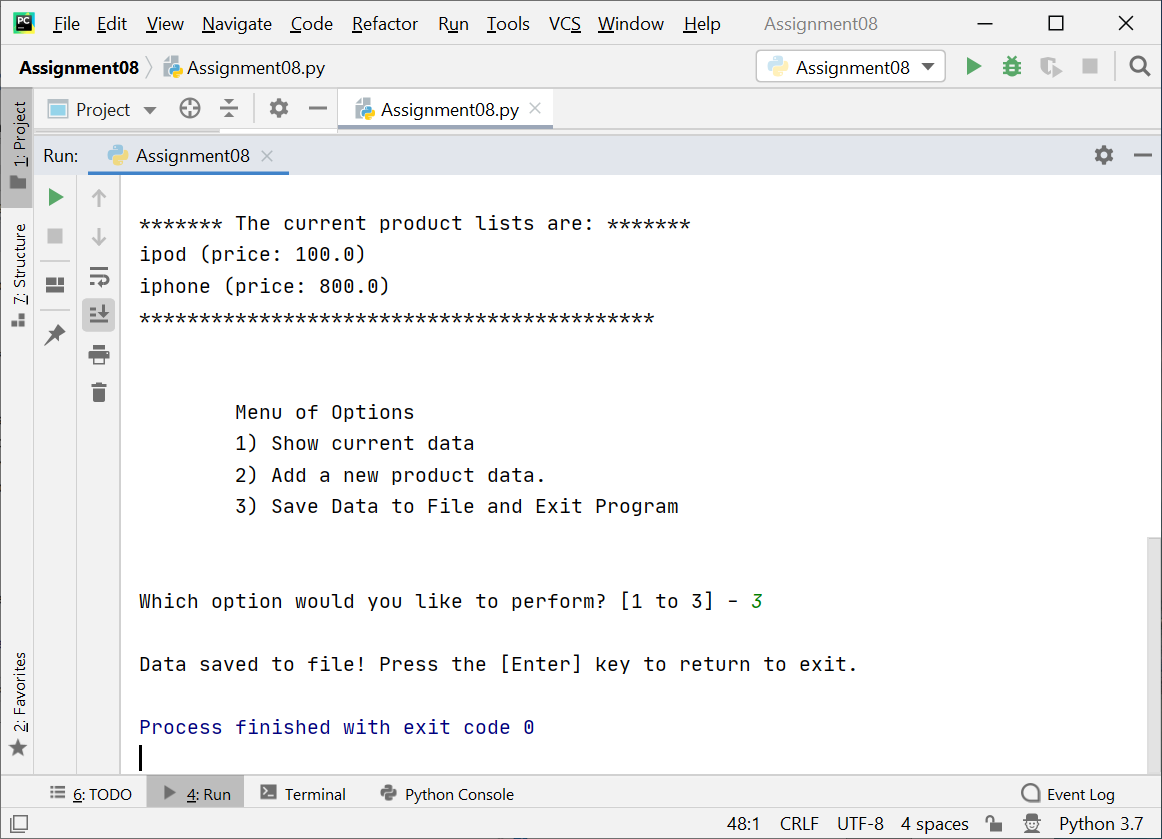


Figure 3. Initial List Saved to a File

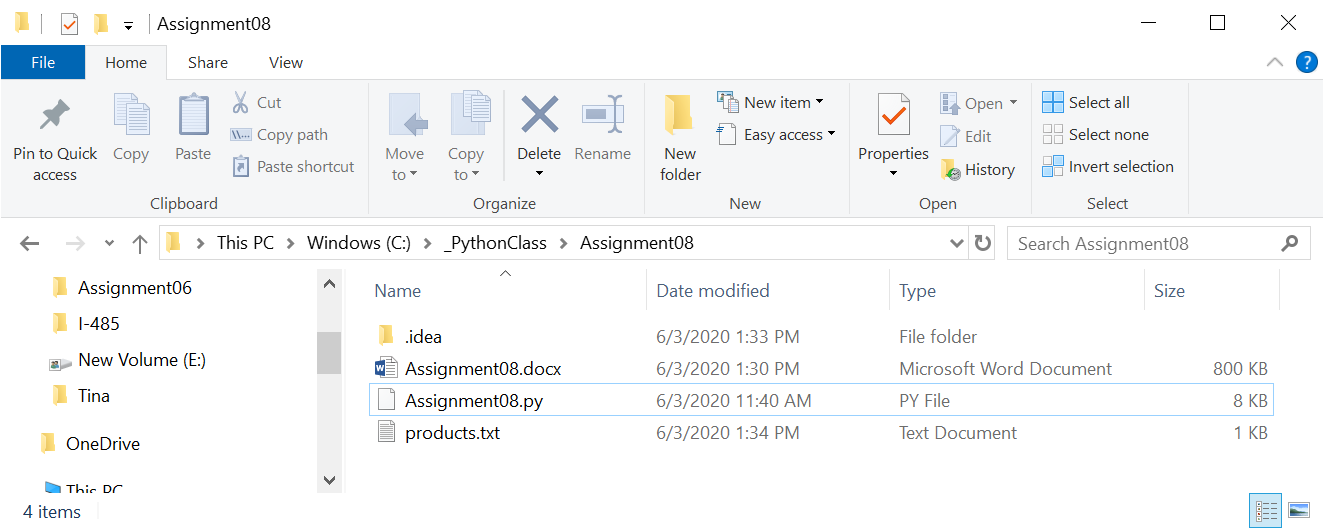


Figure 4. File “products.txt” Created

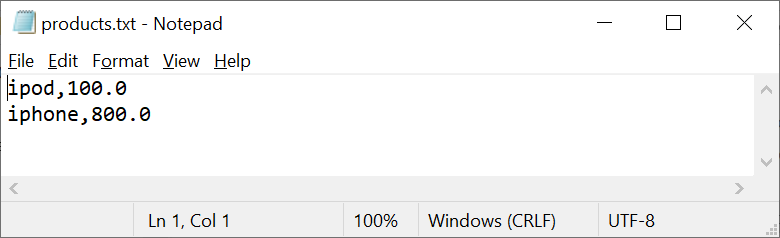


Figure 5. Content of products.txt

At this point, the script is run again and the current list in the file is shown in Figure 6.

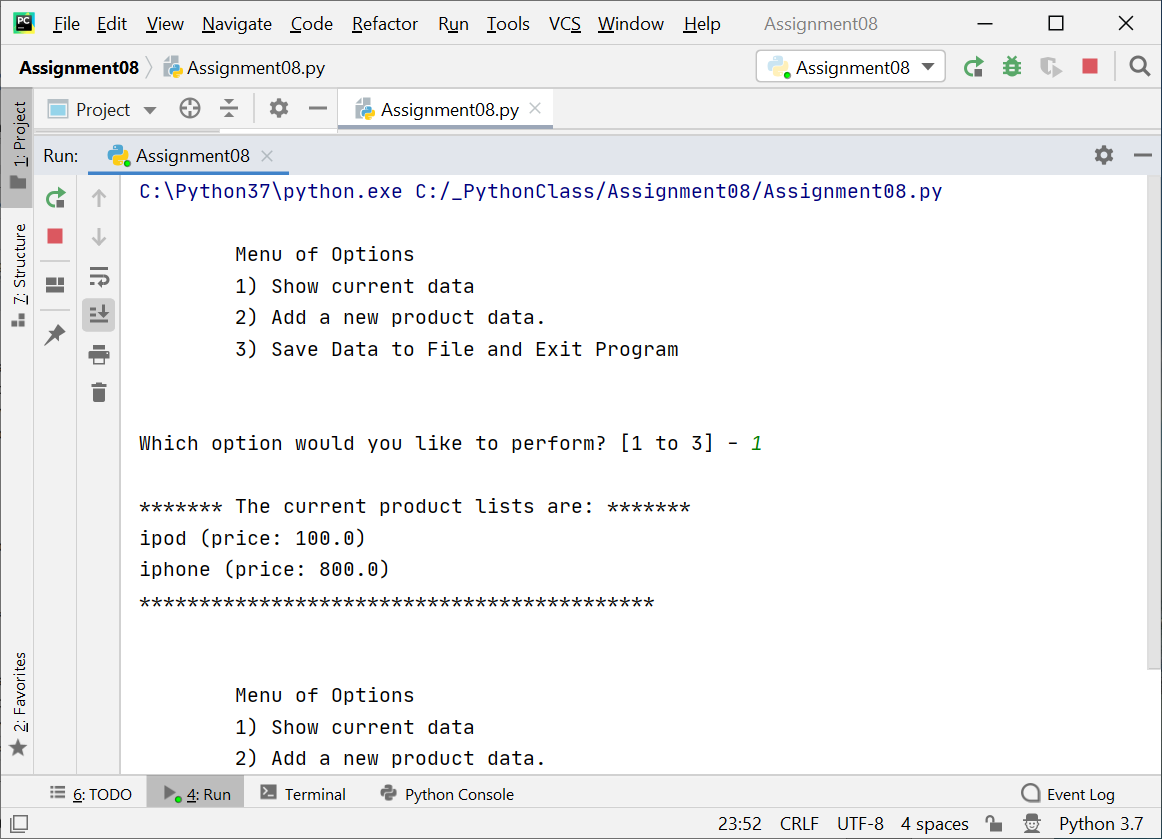


Figure 6. Current List in the File

Now one more product is added and the updated list now has three products

ipod, 100.0

iphone, 800.0

ipad, 500.0

as shown in Figure 7.

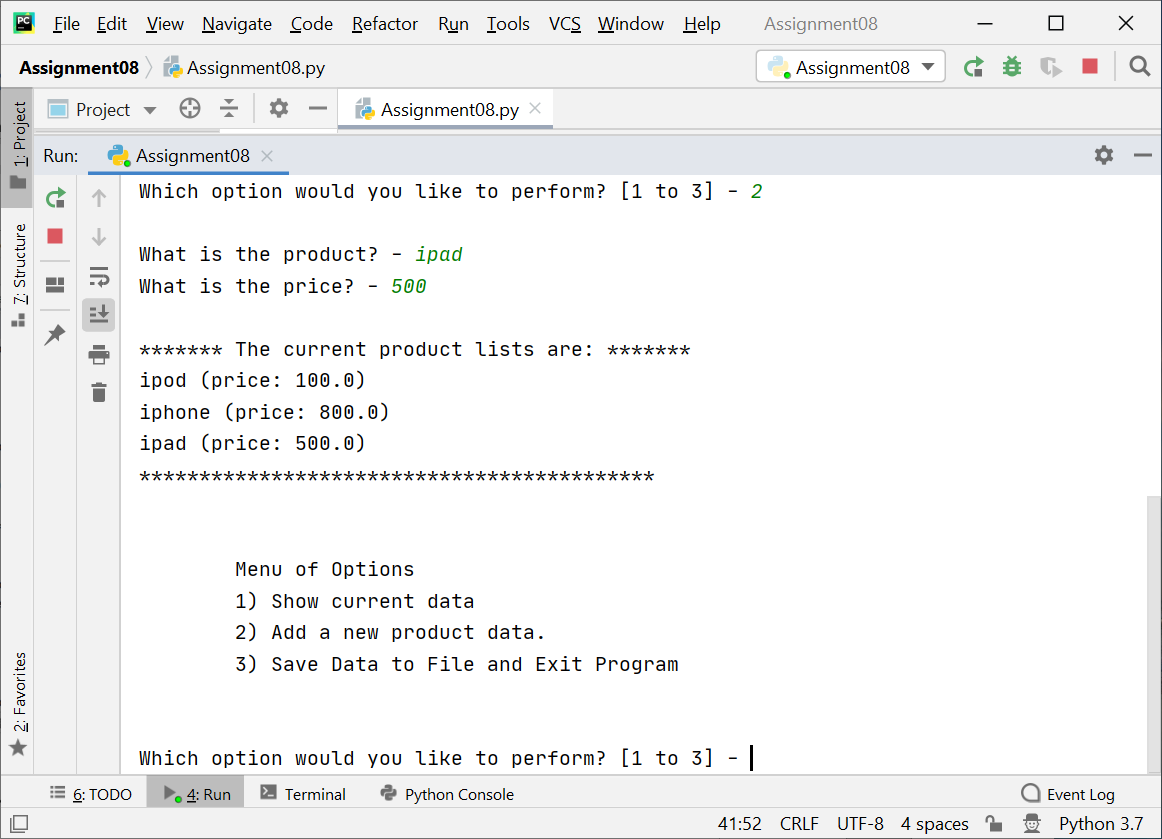


Figure 7. Third Product Added

Then save data and exit. The updated file “products.txt” has three products (Figure 8).

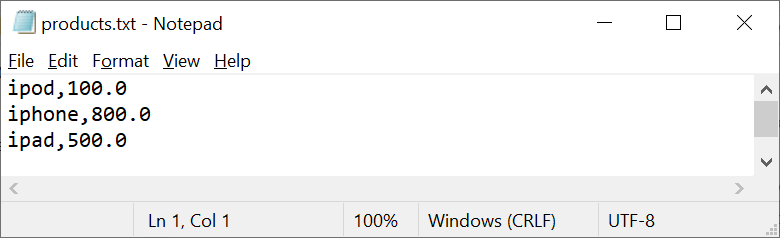


Figure 8. Updated File “products.txt”

2. Run the script in a Windows command shell.

In the Windows command shell, first change the current folder to **C:\\_PythonClass\Assignment08**. The run in Windows command shell continues after the PyCharm run so the file “products.txt” already exists in the folder with three products saved and is correctly listed (Figure 9).

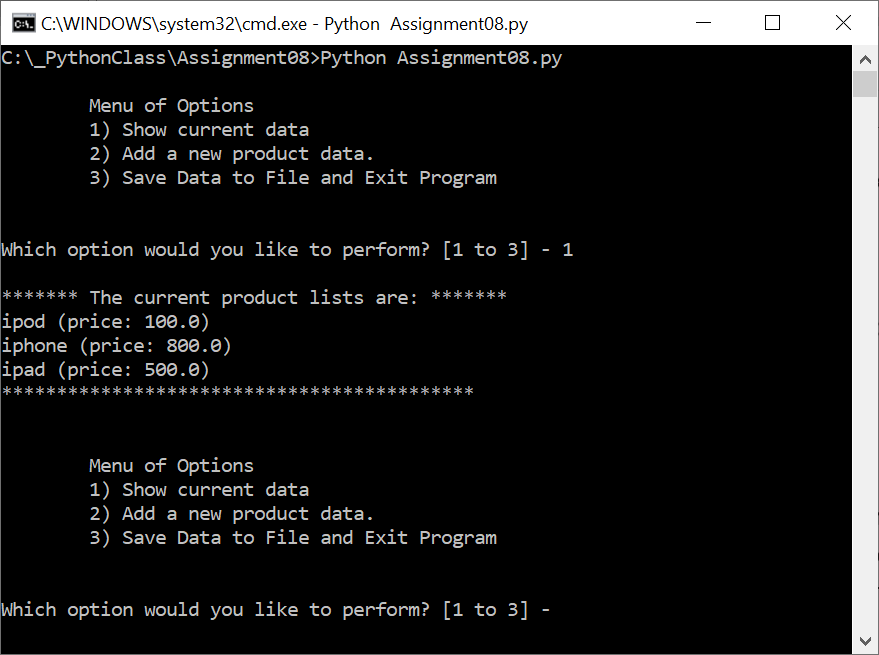


Figure 9. Current List as Read by Script

Add the fourth product, and it becomes

ipod, 100.0

iphone, 800.0

ipad, 500.0

apple watch, 400.0

as shown in Figure 10.

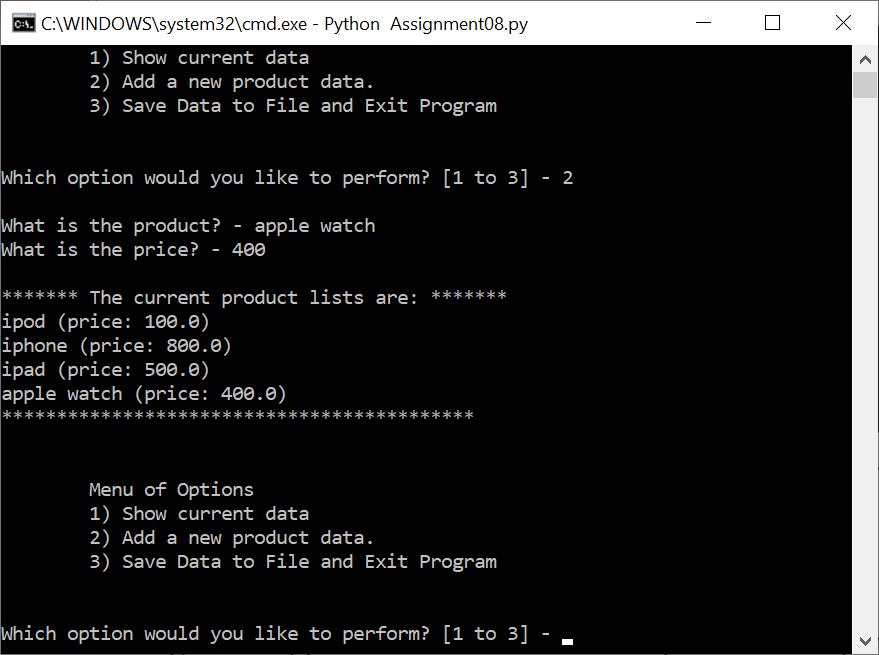


Figure 10. Fourth Product Added

Save the data to file “products.txt” and exit (Figure 11). The updated file “products.txt” is shown in Figure 12.

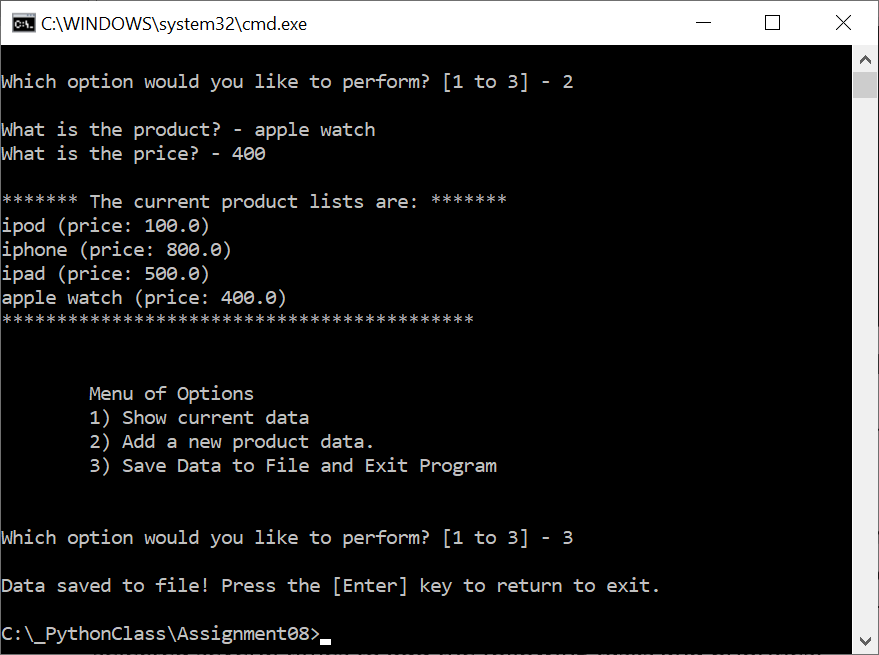


Figure 11. Save to File and Exit

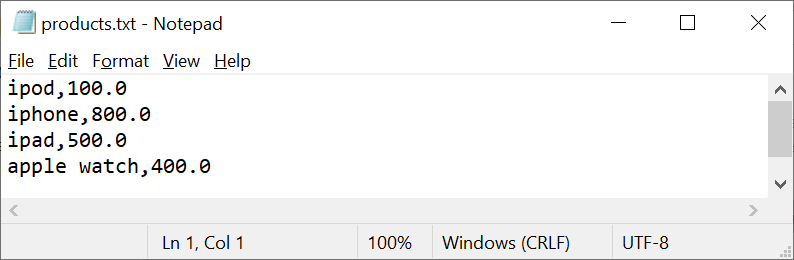


Figure 12. File “products.txt” with Four Products

Conclusion

The assignment is to utilize the object and class in the Python script for abstraction and encapsulation. Three classes are defined for this assignment. One focuses on data storage and the other two focus on data processing. The data storage class “Product” has private attributes and properties to store different values and for accessing and changing their values. The data processing classes mainly have static methods for direct use without objects. In addition, error-handling constructs are also used to capture runtime exceptions. The test runs in both Windows shell and PyCharm showed that the class definition and object instantiation operate correctly.