Erica Peterson

August 29th, 2022

Foundations of Programming: Python

Assignment 08

https://github.com/ericapet/ItFdn100-Mod08

Custom Classes

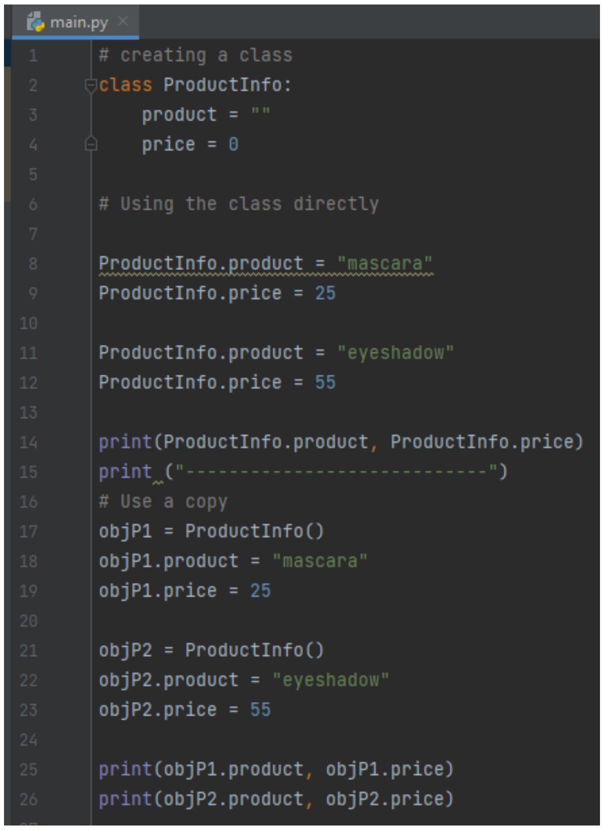
Introduction

This week in the foundations of programming: Python course, we learned about custom classes and how to use them to organize code. Then, I used these skills to create a script with 3 custom classes that stores information about products and their prices in a list.

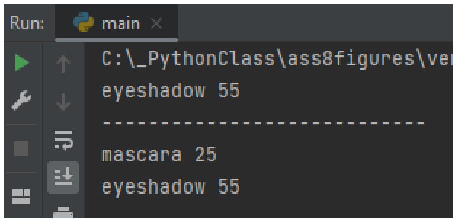
Classes

Classes are a way of grouping data and functions. Classes are basically another level up on the organizational hierarchy, functions organize statements and classes organize functions. Classes can also reuse functions. This organization makes the program easier to create and easier for others to understand. Functions and classes only run when they are called, unlike statements.

Classes can also be copied, to save each individual piece of data within it. Typically, a programmer would use a class directly if its focus is processing data and would use a copy of the class if its for storing unique data points. When you create a copy you can save multiple sets of data, rather than overwriting the first data with the second. An example of a class and copying that class can be seen in Figure 1, on the following page.



```
1   # creating a class
2   class ProductInfo:
3       product = ""
4       price = 0
5
6   # Using the class directly
7
8   ProductInfo.product = "mascara"
9   ProductInfo.price = 25
10
11  ProductInfo.product = "eyeshadow"
12  ProductInfo.price = 55
13
14  print(ProductInfo.product, ProductInfo.price)
15  print ("---------------------------")
16  # Use a copy
17  objP1 = ProductInfo()
18  objP1.product = "mascara"
19  objP1.price = 25
20
21  objP2 = ProductInfo()
22  objP2.product = "eyeshadow"
23  objP2.price = 55
24
25  print(objP1.product, objP1.price)
26  print(objP2.product, objP2.price)
```



```
C:\_PythonClass\ass8figures\ve
eyeshadow 55
----------------------------
mascara 25
eyeshadow 55
```

Figure 1. How to create a class is seen in the left image in lines 1-4. Lines 6-15 show how to directly use a class. Lines 16-26 how to create a copy of that class. The right image shows the output of running this program. Notice that the print statement from line 14 only shows the second piece of data entered, this is because it is not a copy and therefore the class can only hold 1 data point. Notice that the print statement from lines 25 and 26 show both pieces of data because it is a copy.

Class Patterns

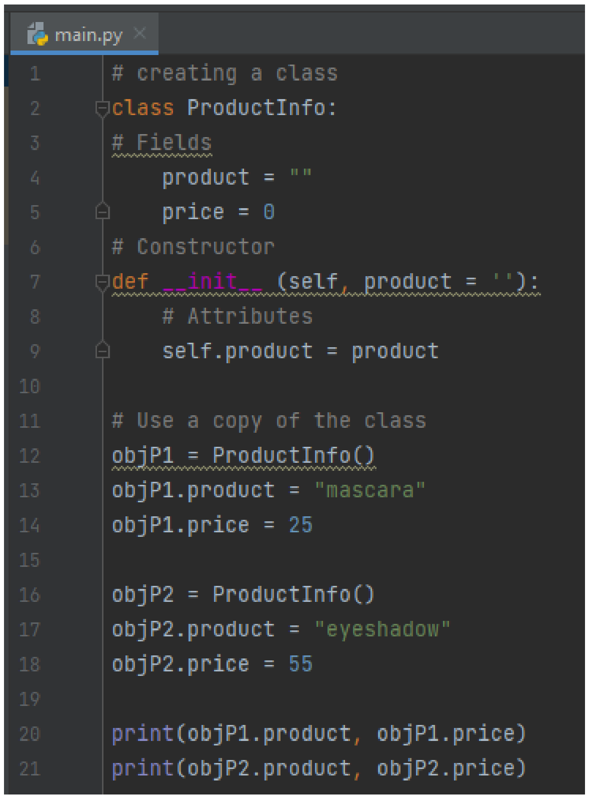
Classes usually have fields, constructors, properties, and methods.

Fields and Attributes

Fields are variables that are inside of a class. Attributes are virtual fields, and these are typically used in python instead of fields. In Figure 1, above, product and price are examples of fields within the class ProductInfo. An example of a field can also be seen in Figure 2.

Constructors

Constructors are functions (methods) within a class that automatically run when you crate an object from the class. Constructors often set the initial values of field data. Python’s constructors use the name “__init__”, with double underscores. Constructors allow you to crate and initialize objects of a class, making the objects ready to use.



main.py

```python
1   # creating a class
2   class ProductInfo:
3   # Fields
4       product = ""
5       price = 0
6   # Constructor
7   def __init__(self, product = ''):
8       # Attributes
9       self.product = product
10
11  # Use a copy of the class
12  objP1 = ProductInfo()
13  objP1.product = "mascara"
14  objP1.price = 25
15
16  objP2 = ProductInfo()
17  objP2.product = "eyeshadow"
18  objP2.price = 55
19
20  print(objP1.product, objP1.price)
21  print(objP2.product, objP2.price)
```

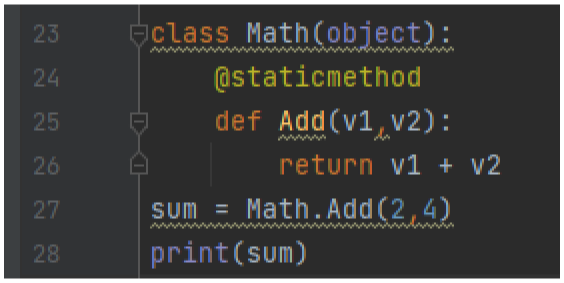
Figure 2. How to use fields, attributes, and constructors. The fields, product and price, are in lines 3-5. The constructor is in line 6-9.

Properties and Methods

Properties are functions used to manage attribute data. Typically, you create 2 properties per each attribute, one for getting data and one for setting data. These are called getters and setters. Getters are used in your presentation code whereas setters are usually used in processing and error handling. Methods are other functions inside of a class that allow you to organize processing statements into named groups.

The Self Keyword and Static Methods

The keyword "self" in the constructor method is used to refer to data and functions found in an object instance. The code of a class gets loaded into memory when your program starts running and this can only occur once. But you can have multiple instances of a class, each representing a copy and the copy is identified by using the pronoun "self". If you want to have methods called directly from the class, without an object, you must add the @staticmethod before the function, then you can call the method by using the name of the class and the name of the method together. Classes can have both self-methods and static methods but typically they have one or the other. When a class focuses on processing data, it uses static methods. When the class focuses on storing data use self-methods. An example using @staticmethod can be seen in Figure 3, below. An example using the self keyword can be found in line 7 of Figure 2.



```
23  class Math(object):
24      @staticmethod
25      def Add(v1,v2):
26          return v1 + v2
27  sum = Math.Add(2,4)
28  print(sum)
```

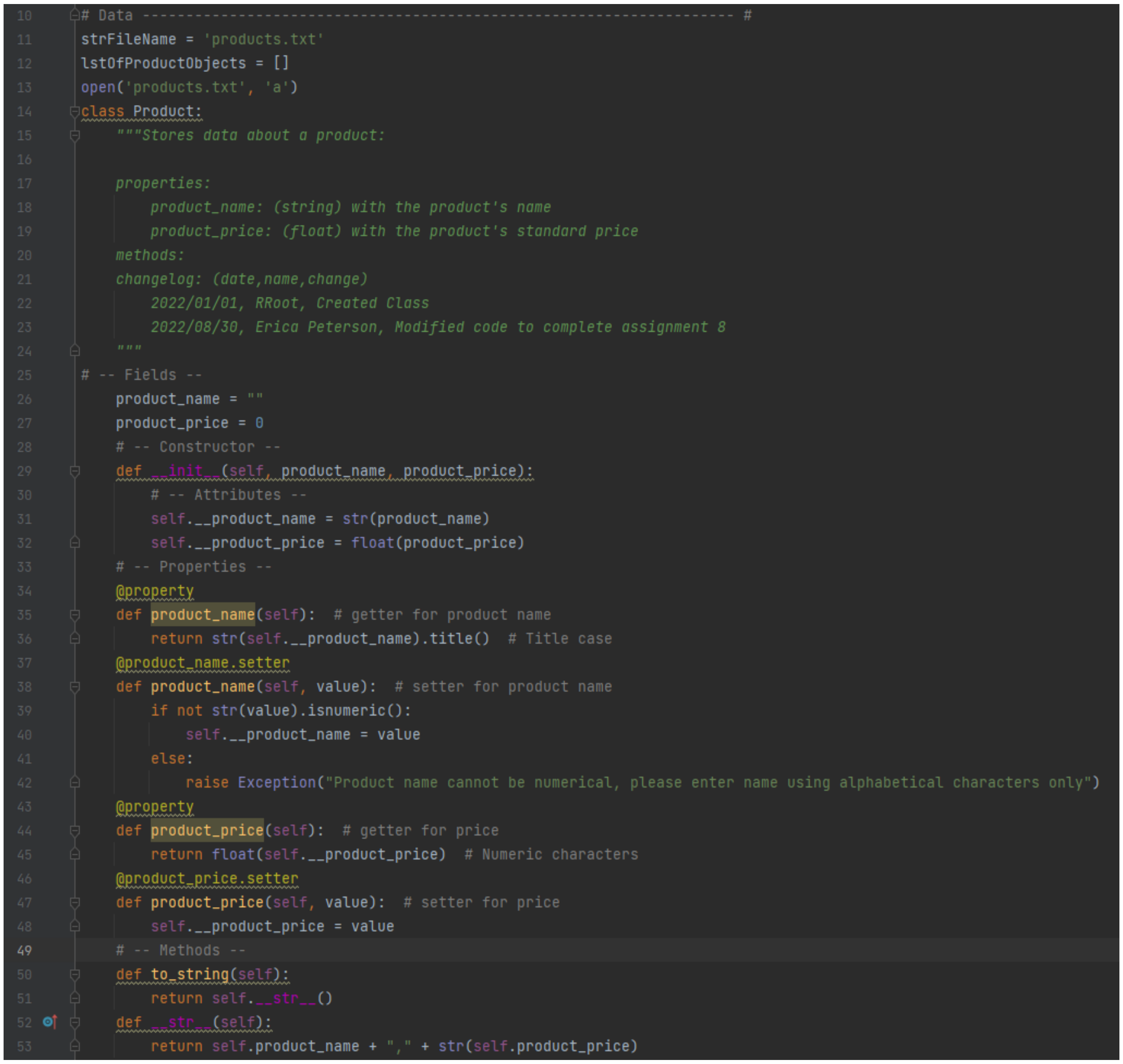
Figure 3. How to use @staticmethod to process data. Notice in line 27 we call the method by class.method(value) which in this case is Math.Add(2,4).

An Example of Custom Classes in a Script

Now that we have a bit more understanding of custom classes and their patterns, we can create a script. This script will track products and their prices in a list in a .txt file. The user will be shown a menu of choices to either add data, show the data, save the data, and exit.

Data

Before we get to the user interface, we are going to create some classes. The first class we create will be a self method class, to store data about the product. This class will be called Product and has fields, a constructor, 4 properties with 2 getters and 2 setters, and 2 methods. Figure 4, on the following page shows the completed data code block.

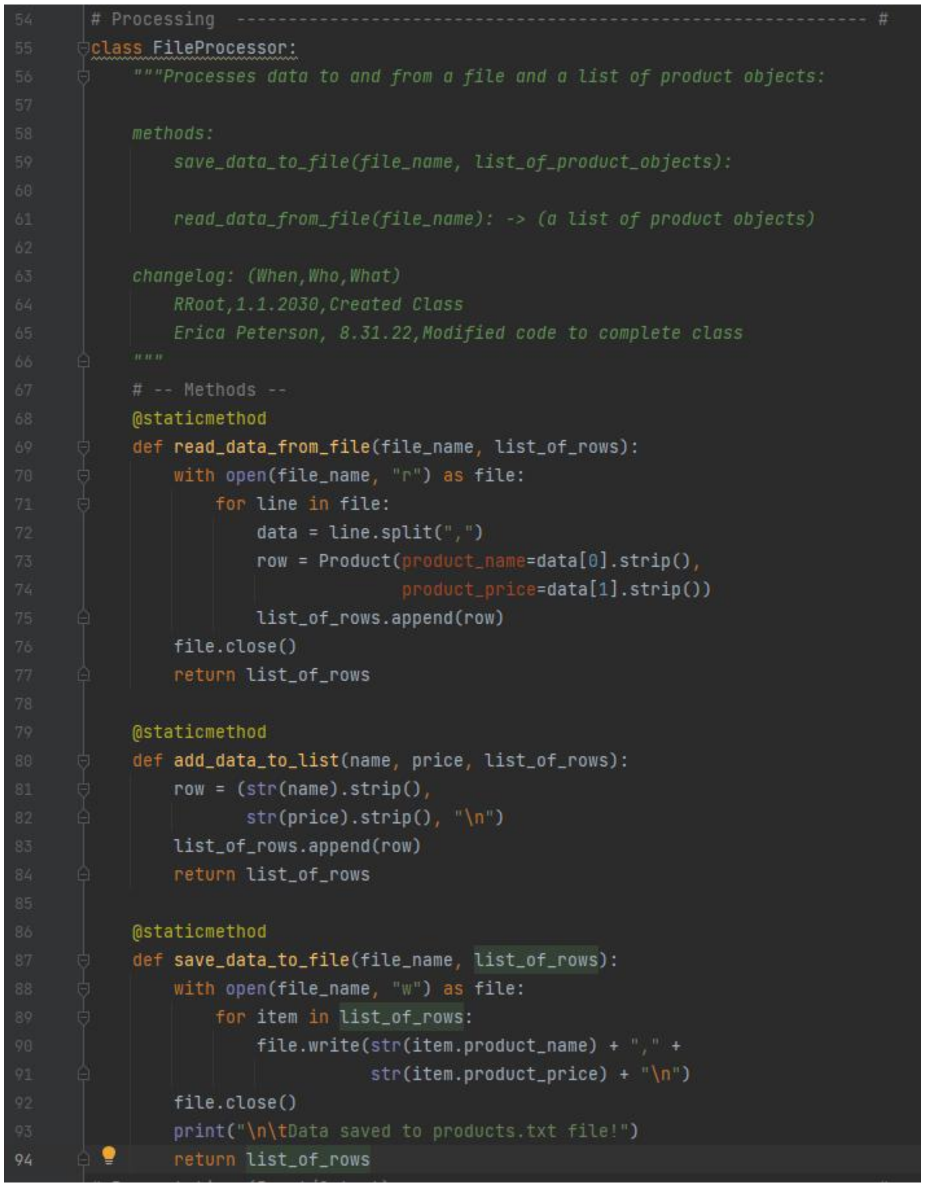


```
10  # Data -------------------------------------------------------------------- #
11  strFileName = 'products.txt'
12  lstOfProductObjects = []
13  open('products.txt', 'a')
14  class Product:
15      """Stores data about a product:
16  
17      properties:
18          product_name: (string) with the product's name
19          product_price: (float) with the product's standard price
20      methods:
21      changelog: (date,name,change)
22          2022/01/01, RRoot, Created Class
23          2022/08/30, Erica Peterson, Modified code to complete assignment 8
24      """
25  # -- Fields --
26      product_name = ""
27      product_price = 0
28      # -- Constructor --
29      def __init__(self, product_name, product_price):
30          # -- Attributes --
31          self.__product_name = str(product_name)
32          self.__product_price = float(product_price)
33      # -- Properties --
34      @property
35      def product_name(self):  # getter for product name
36          return str(self.__product_name).title()  # Title case
37      @product_name.setter
38      def product_name(self, value):  # setter for product name
39          if not str(value).isnumeric():
40              self.__product_name = value
41          else:
42              raise Exception("Product name cannot be numerical, please enter name using alphabetical characters only")
43      @property
44      def product_price(self):  # getter for price
45          return float(self.__product_price)  # Numeric characters
46      @product_price.setter
47      def product_price(self, value):  # setter for price
48          self.__product_price = value
49      # -- Methods --
50      def to_string(self):
51          return self.__str__()
52      def __str__(self):
53          return self.product_name + "," + str(self.product_price)
```

Figure 4. The code block for storing product data. The class, Product, starts at line 14. Lines 15-24 are a docstring that explains what this code block will do. Lines 25-27 show the fields (variables) that will be used in this class. Lines 28-32, show a constructor which sets the initial values for the field data (line 29, __init__).Lines 33-48 hold 4 properties, the first two are for the product name, one is a getter and the other is a setter. Notice in the product_name.setter, there is an example of error handling if the user tries to enter a numerical product name then a message will be displayed letting them know they cannot do this. The second two properties are a getter and setter for the product price. Finally, in lines 49-53 are the methods, using the __str__ method to return the classes data as a string.

Processing

The next code block focuses on processing the data and for this reason we will use @staticmethod for our class, FileProcessor. This code block contains all the methods for processing the data, reading data from the file, adding data to the list, and saving data to the file. Figure 5 contains the processing code block, below.

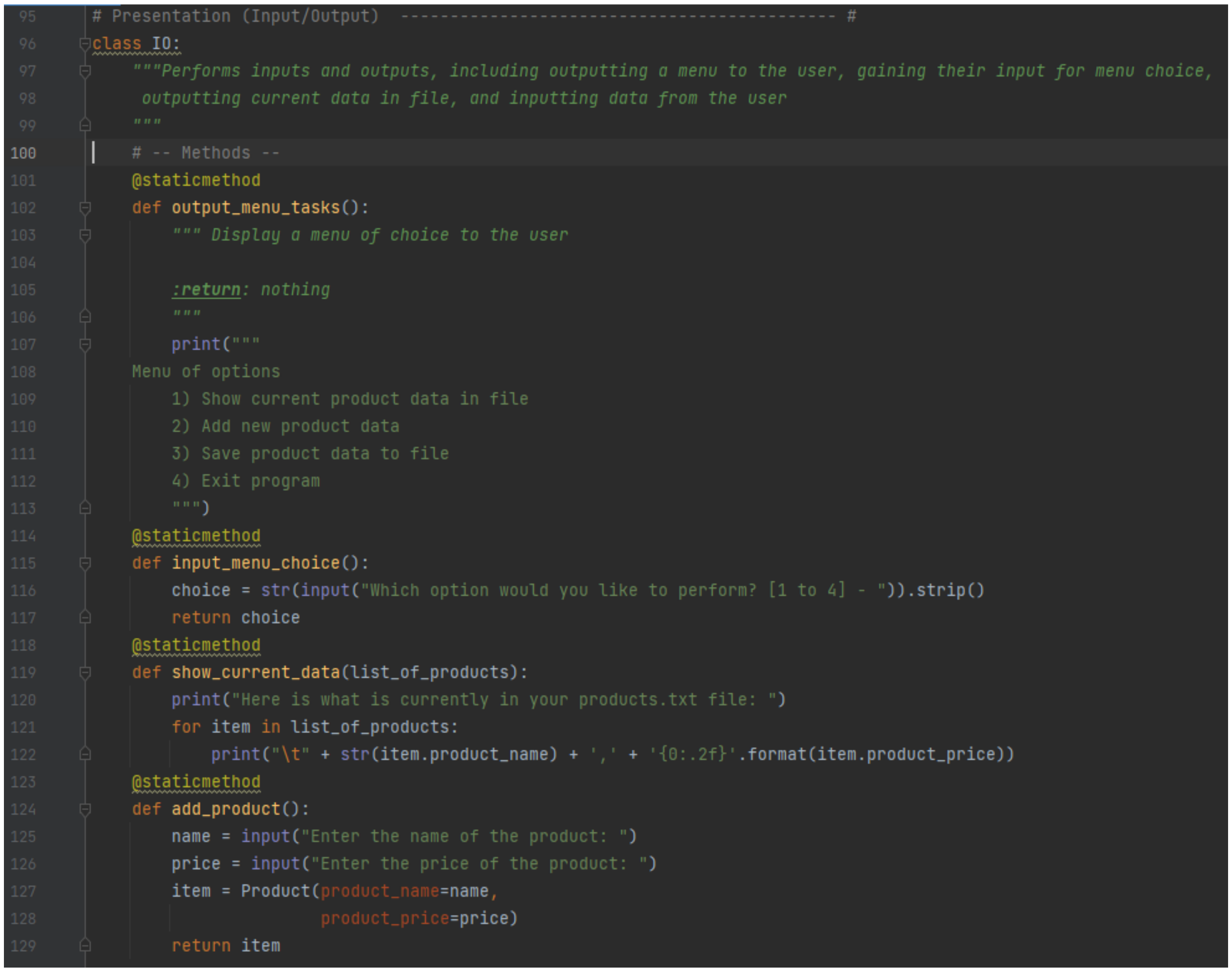


```
54  # Processing  ------------------------------------------------------------ #
55  class FileProcessor:
56      """Processes data to and from a file and a list of product objects:
57
58      methods:
59          save_data_to_file(file_name, list_of_product_objects):
60
61          read_data_from_file(file_name): -> (a list of product objects)
62
63      changelog: (When,Who,What)
64          RRoot,1.1.2030,Created Class
65          Erica Peterson, 8.31.22,Modified code to complete class
66      """
67      # -- Methods --
68      @staticmethod
69      def read_data_from_file(file_name, list_of_rows):
70          with open(file_name, "r") as file:
71              for line in file:
72                  data = line.split(",")
73                  row = Product(product_name=data[0].strip(),
74                                product_price=data[1].strip())
75                  list_of_rows.append(row)
76          file.close()
77          return list_of_rows
78
79      @staticmethod
80      def add_data_to_list(name, price, list_of_rows):
81          row = (str(name).strip(),
82                 str(price).strip(), "\n")
83          list_of_rows.append(row)
84          return list_of_rows
85
86      @staticmethod
87      def save_data_to_file(file_name, list_of_rows):
88          with open(file_name, "w") as file:
89              for item in list_of_rows:
90                  file.write(str(item.product_name) + "," +
91                             str(item.product_price) + "\n")
92          file.close()
93          print("\n\tData saved to products.txt file!")
94          return list_of_rows
```

Figure 5. The code block for processing the data. Lines 56-66 are a docstring, explaining what the block is going to do. The first @staticmethod can be found in lines 68-77 this method reads data from the file, first it must open the file (line 70), then it reads the data adding a comma between the product and price. The next method adds data to the list (lines 79-84). The final method saves data to the file (lines 86-94) and prints out a message to the use stating that the data has been saved.

Presentation

The next code block focuses on presenting the menu to the user and gaining their inputs. This class will be called IO, for input and output. Because data is also being processed here, we will be using the @staticmethod again. Then, there will be a method for menu choice, showing current data, and adding a new product. The presentation code block can be found below in Figure 6.

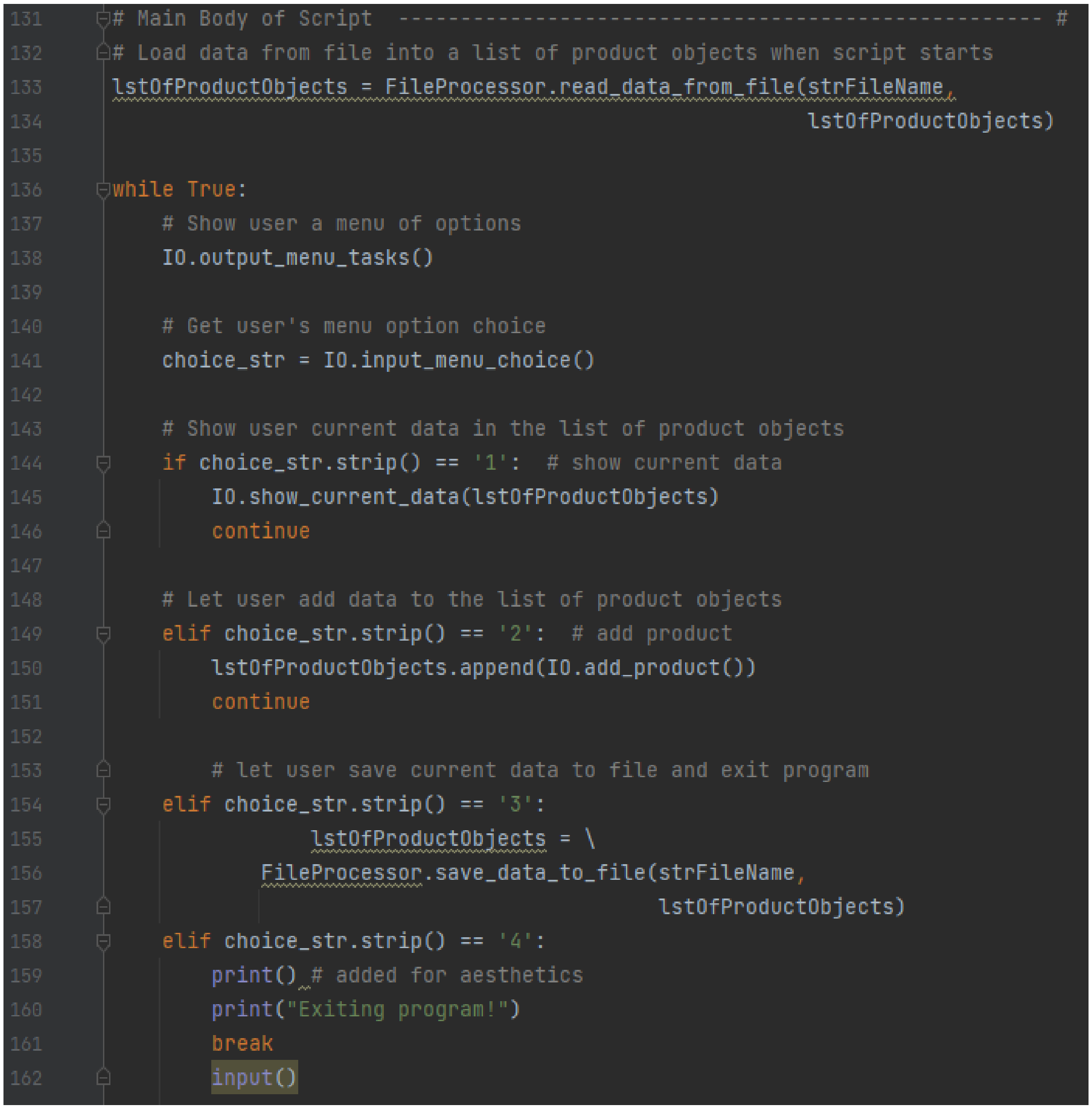


```
95   # Presentation (Input/Output)  ------------------------------------------- #
96   class IO:
97       """Performs inputs and outputs, including outputting a menu to the user, gaining their input for menu choice,
98        outputting current data in file, and inputting data from the user
99       """
100      # -- Methods --
101      @staticmethod
102      def output_menu_tasks():
103          """ Display a menu of choice to the user
104
105          :return: nothing
106          """
107          print("""
108      Menu of options
109          1) Show current product data in file
110          2) Add new product data
111          3) Save product data to file
112          4) Exit program
113          """)
114      @staticmethod
115      def input_menu_choice():
116          choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
117          return choice
118      @staticmethod
119      def show_current_data(list_of_products):
120          print("Here is what is currently in your products.txt file: ")
121          for item in list_of_products:
122              print("\t" + str(item.product_name) + ',' + '{0:.2f}'.format(item.product_price))
123      @staticmethod
124      def add_product():
125          name = input("Enter the name of the product: ")
126          price = input("Enter the price of the product: ")
127          item = Product(product_name=name,
128                         product_price=price)
129          return item
```

Figure 6. The code block for presentation, gaining inputs and outputting data. Line 96 starts the class, IO. Lines 97-99 are a docstring explaining the point of the code block. Lines 101-113 are the method for displaying a menu to the user, using a print statement. Lines 114-117, ask the user which menu option they would like and saves that as the "choice" variable. Lines 118-122 is the method that will be called to show what is currently in the products.txt file. Lines 123-129 get user input on the name and price of the product they want to add to the list.

Main Body

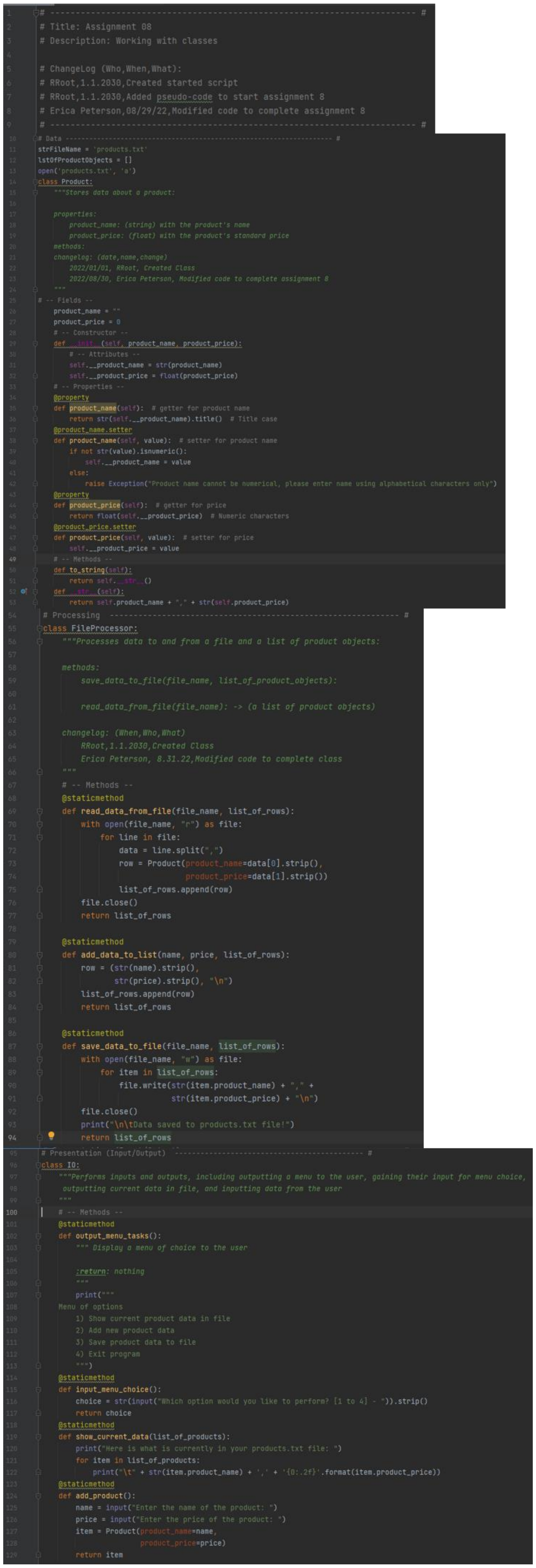
Finally, we have the main body of the script. This is where we will call all our classes and methods that we created in the previous code blocks. First, we will set up a while loop to display the menu to the user. Then, we will use if-elif statements, similar to Assignments 5 and 6, to go through the menu choices based on *if* the user input the corresponding number from the menu. I also chose to add in a fourth option, to exit the program, using break to break the loop and exit. The code for the main body can be seen on the following page, in Figure 7.



```
131 # Main Body of Script  ---------------------------------------------------- #
132 # Load data from file into a list of product objects when script starts
133 lstOfProductObjects = FileProcessor.read_data_from_file(strFileName,
134                                                         lstOfProductObjects)
135
136 while True:
137     # Show user a menu of options
138     IO.output_menu_tasks()
139
140     # Get user's menu option choice
141     choice_str = IO.input_menu_choice()
142
143     # Show user current data in the list of product objects
144     if choice_str.strip() == '1':  # show current data
145         IO.show_current_data(lstOfProductObjects)
146         continue
147
148     # Let user add data to the list of product objects
149     elif choice_str.strip() == '2':  # add product
150         lstOfProductObjects.append(IO.add_product())
151         continue
152
153         # let user save current data to file and exit program
154     elif choice_str.strip() == '3':
155                 lstOfProductObjects = \
156             FileProcessor.save_data_to_file(strFileName,
157                                             lstOfProductObjects)
158     elif choice_str.strip() == '4':
159         print() # added for aesthetics
160         print("Exiting program!")
161         break
162         input()
```

Figure 7. The main body of the script. First, in line 133, we load data from the file into a list of product objects. Then, line 136 sets up a while loop for displaying the menu. Line 138, calls the class IO and the method output_menu_tasks() to display the menu. Line 141, gets the users menu choice by calling the IO class and the input_ menu_choice() method. Next, we set up an if-elif statement for each of the 4 menu options from lines 144-162. The first option, show current data in list, can be seen in lines 144-146. The second option, adding data to the list, can be seen in line 148-151. The third option, saving current data, can be seen in lines 153-156. These 3 options are performed by calling the class, and then the method. Finally, the fourth option, to exit, can be seen in lines 158-162, this option does not use a class or a method, only the break function and some print statements for the user interface.

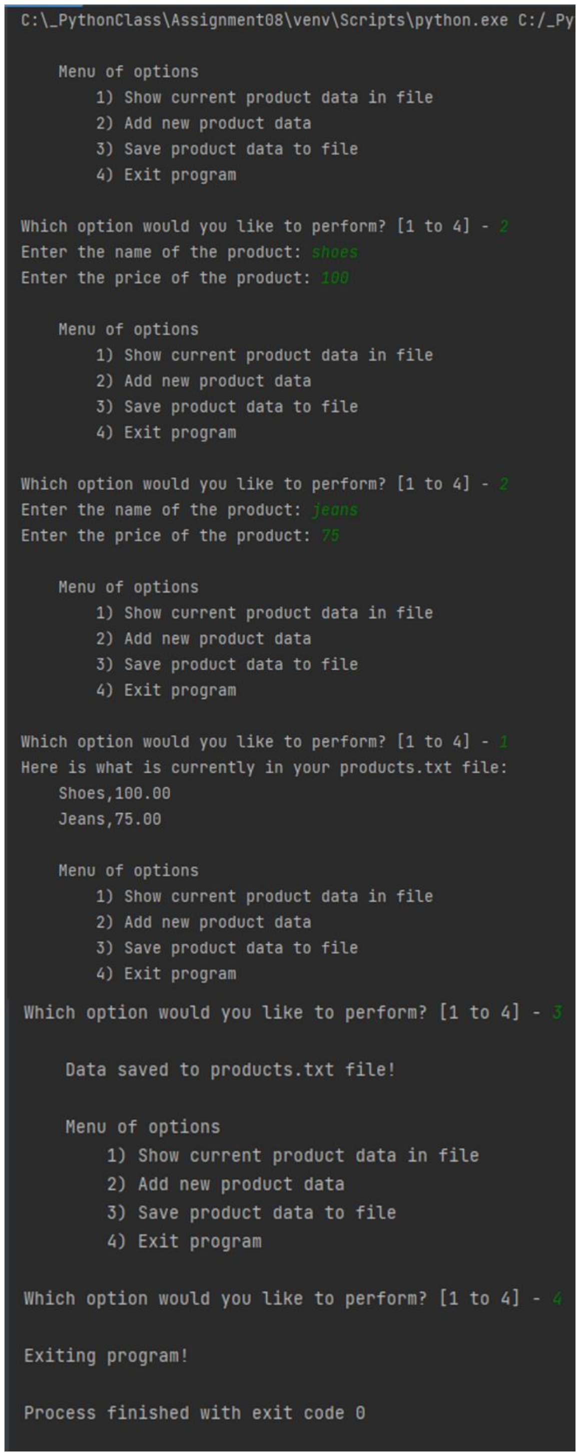
That’s all folks! Now we can look at the entire script all together, as well as an example of it running in PyCharm and in Command Prompt. The entire script can be seen on the following page, in Figure 8. The script running in PyCharm can be seen on the next page, in Figure 9, and then on the next page we can see it running in Command Prompt in Figure 10.



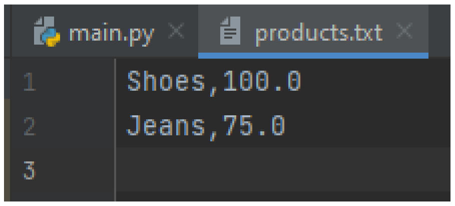
```python
# ------------------------------------------------------------------------ #
# Title: Assignment 08
# Description: Working with classes

# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# RRoot,1.1.2030,Added pseudo-code to start assignment 8
# Erica Peterson,08/29/22,Modified code to complete assignment 8
# ------------------------------------------------------------------------ #
# Data -------------------------------------------------------------------- #
strFileName = 'products.txt'
lstOfProductObjects = []
open('products.txt', 'a')
class Product:
    """Stores data about a product:

    properties:
        product_name: (string) with the product's name
        product_price: (float) with the product's standard price
    methods:
    changelog: (date,name,change)
        2022/01/01, RRoot, Created Class
        2022/08/30, Erica Peterson, Modified code to complete assignment 8
    """
# -- Fields --
    product_name = ""
    product_price = 0
    # -- Constructor --
    def __init__(self, product_name, product_price):
        # -- Attributes --
        self.__product_name = str(product_name)
        self.__product_price = float(product_price)
    # -- Properties --
    @property
    def product_name(self):  # getter for product name
        return str(self.__product_name).title()  # Title case
    @product_name.setter
    def product_name(self, value):  # setter for product name
        if not str(value).isnumeric():
            self.__product_name = value
        else:
            raise Exception("Product name cannot be numerical, please enter name using alphabetical characters only")
    @property
    def product_price(self):  # getter for price
        return float(self.__product_price)  # Numeric characters
    @product_price.setter
    def product_price(self, value):  # setter for price
        self.__product_price = value
    # -- Methods --
    def to_string(self):
        return self.__str__()
    def __str__(self):
        return self.product_name + "," + str(self.product_price)
 # Processing  ------------------------------------------------------------ #
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
        Erica Peterson, 8.31.22,Modified code to complete class
    """
    # -- Methods --
    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        with open(file_name, "r") as file:
            for line in file:
                data = line.split(",")
                row = Product(product_name=data[0].strip(),
                              product_price=data[1].strip())
                list_of_rows.append(row)
        file.close()
        return list_of_rows

    @staticmethod
    def add_data_to_list(name, price, list_of_rows):
        row = (str(name).strip(),
               str(price).strip(), "\n")
        list_of_rows.append(row)
        return list_of_rows

    @staticmethod
    def save_data_to_file(file_name, list_of_rows):
        with open(file_name, "w") as file:
            for item in list_of_rows:
                file.write(str(item.product_name) + "," +
                           str(item.product_price) + "\n")
        file.close()
        print("\n\tData saved to products.txt file!")
        return list_of_rows
# Presentation (Input/Output)  -------------------------------------------- #
class IO:
    """Performs inputs and outputs, including outputting a menu to the user, gaining their input for menu choice,
     outputting current data in file, and inputting data from the user
    """
    # -- Methods --
    @staticmethod
    def output_menu_tasks():
        """ Display a menu of choice to the user

        :return: nothing
        """
        print("""
    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program
        """)
    @staticmethod
    def input_menu_choice():
        choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
        return choice
    @staticmethod
    def show_current_data(list_of_products):
        print("Here is what is currently in your products.txt file: ")
        for item in list_of_products:
            print("\t" + str(item.product_name) + ',' + '{0:.2f}'.format(item.product_price))
    @staticmethod
    def add_product():
        name = input("Enter the name of the product: ")
        price = input("Enter the price of the product: ")
        item = Product(product_name=name,
                       product_price=price)
        return item
```

Figure 8. The entire script all together, in PyCharm.

Below, in Figure 9, is the program running in PyCharm.



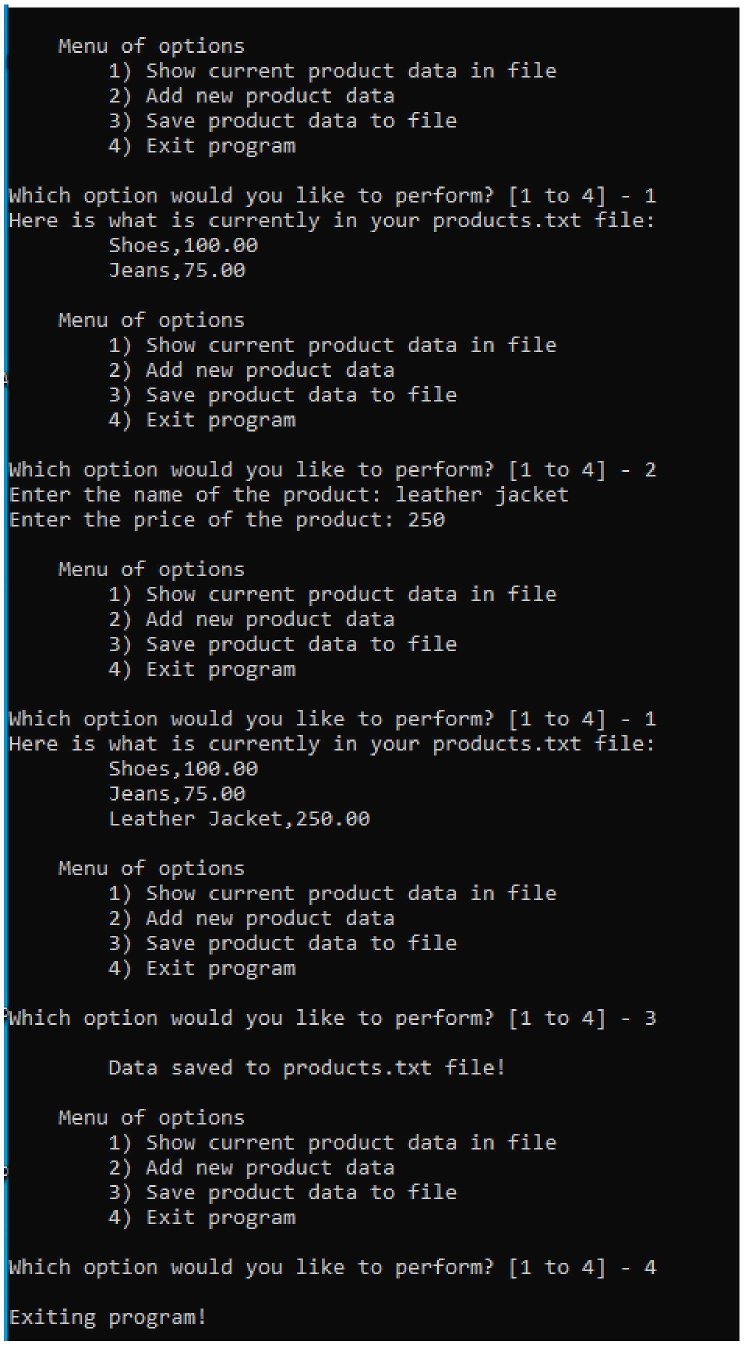
```
C:\_PythonClass\Assignment08\venv\Scripts\python.exe C:/_Py

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 2
Enter the name of the product: shoes
Enter the price of the product: 100

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 2
Enter the name of the product: jeans
Enter the price of the product: 75

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 1
Here is what is currently in your products.txt file:
    Shoes,100.00
    Jeans,75.00

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program
Which option would you like to perform? [1 to 4] - 3

    Data saved to products.txt file!

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

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

Exiting program!

Process finished with exit code 0
```



```
Shoes,100.0
Jeans,75.0

```

Figure 9. The program running in PyCharm is on the left, whereas the products.txt file is on the right.

Next, let's check that the program also runs in Command Prompt, as expected, in Figure 10. Here, we can also have the program load up the file from the last time we ran it, in PyCharm in Figure 9, and see that the data is in the file.



```
    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 1
Here is what is currently in your products.txt file:
        Shoes,100.00
        Jeans,75.00

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 2
Enter the name of the product: leather jacket
Enter the price of the product: 250

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 1
Here is what is currently in your products.txt file:
        Shoes,100.00
        Jeans,75.00
        Leather Jacket,250.00

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

Which option would you like to perform? [1 to 4] - 3

        Data saved to products.txt file!

    Menu of options
        1) Show current product data in file
        2) Add new product data
        3) Save product data to file
        4) Exit program

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

Exiting program!
```

Figure 10. An example of the program running in Command Prompt. Notice that when selecting option 1, the first time, it shows the data from when we previously ran this in PyCharm so the user could continually update this products list.

Summary

This week was particularly challenging for me, as it felt like a lot of jargon trying to learn and understand classes. Hopefully, this assignment and document will be helpful in the future. I have also uploaded the document and code to my GitHub for others to see and comment on.