Chris Perry

November 27, 2019

Foundations of Programming: Python Assignment 08

Github Link - https://github.com/cpwasthere/Module08

Class Object Product List Python Script

Introduction

In this assignment, I will go over how to use object classes to simplify code in a python script that will present the user with 4 choices and ask them to input a product name and it's price. Also from the main menu the user can view the current data they have input, choose to save their data to a local file (in the working directory) and/or quit the program. I will be creating the script using Python3, PyCharm CE and macOS.

Coming up with the Code

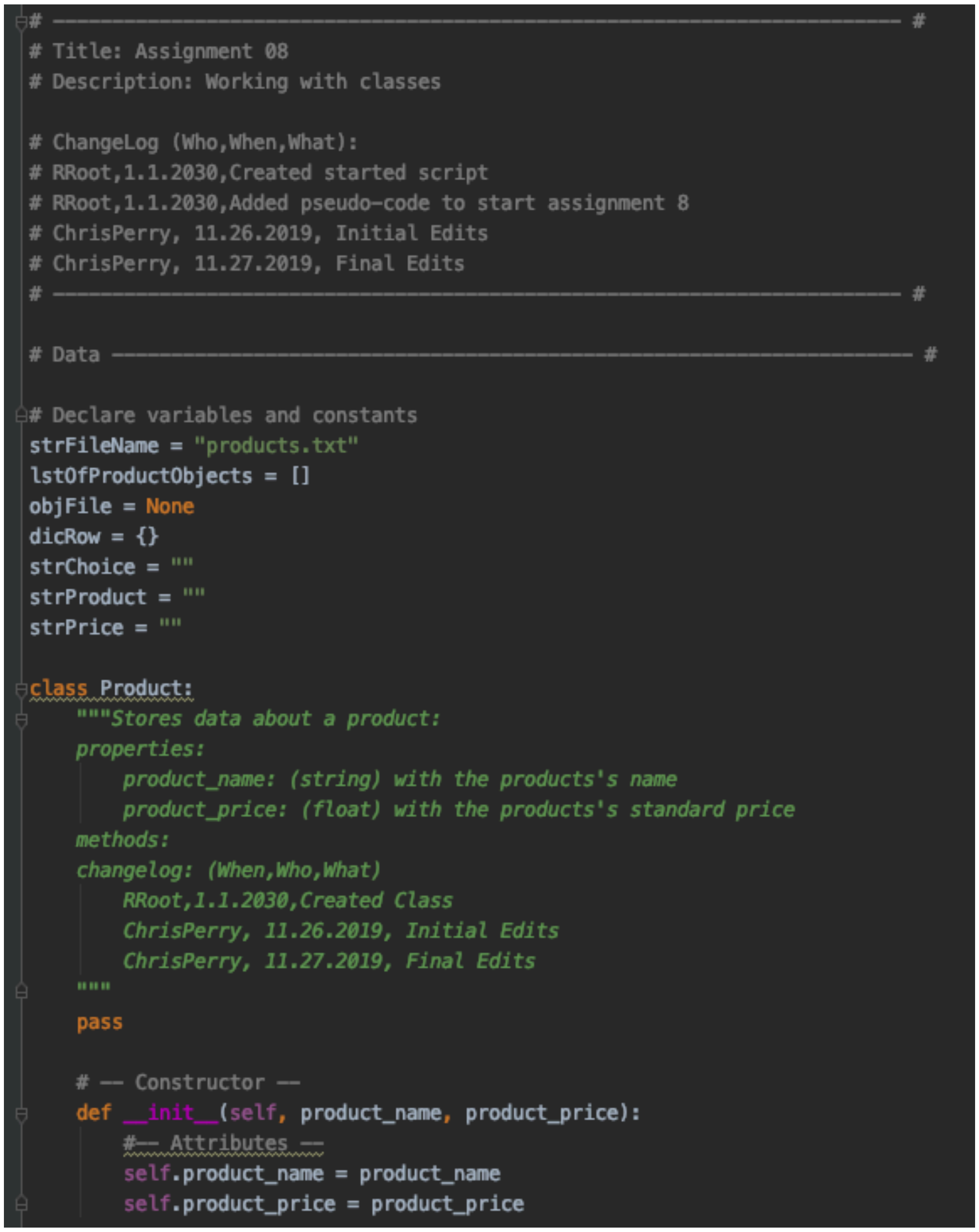
To write my code, I used PyCharm CE - a popular python programming editor/application.

As apart of the assignment, I was given a template python script that provided some starter code and general direction on where to go with the assignment.

This assignment is very similar to assignment 6 (working with functions) and a lot of that code can and was reused for this assignment. The main thing that I needed to add to this assignment was object classes and provide the ability to assign them values based off user input.

How the Code Works

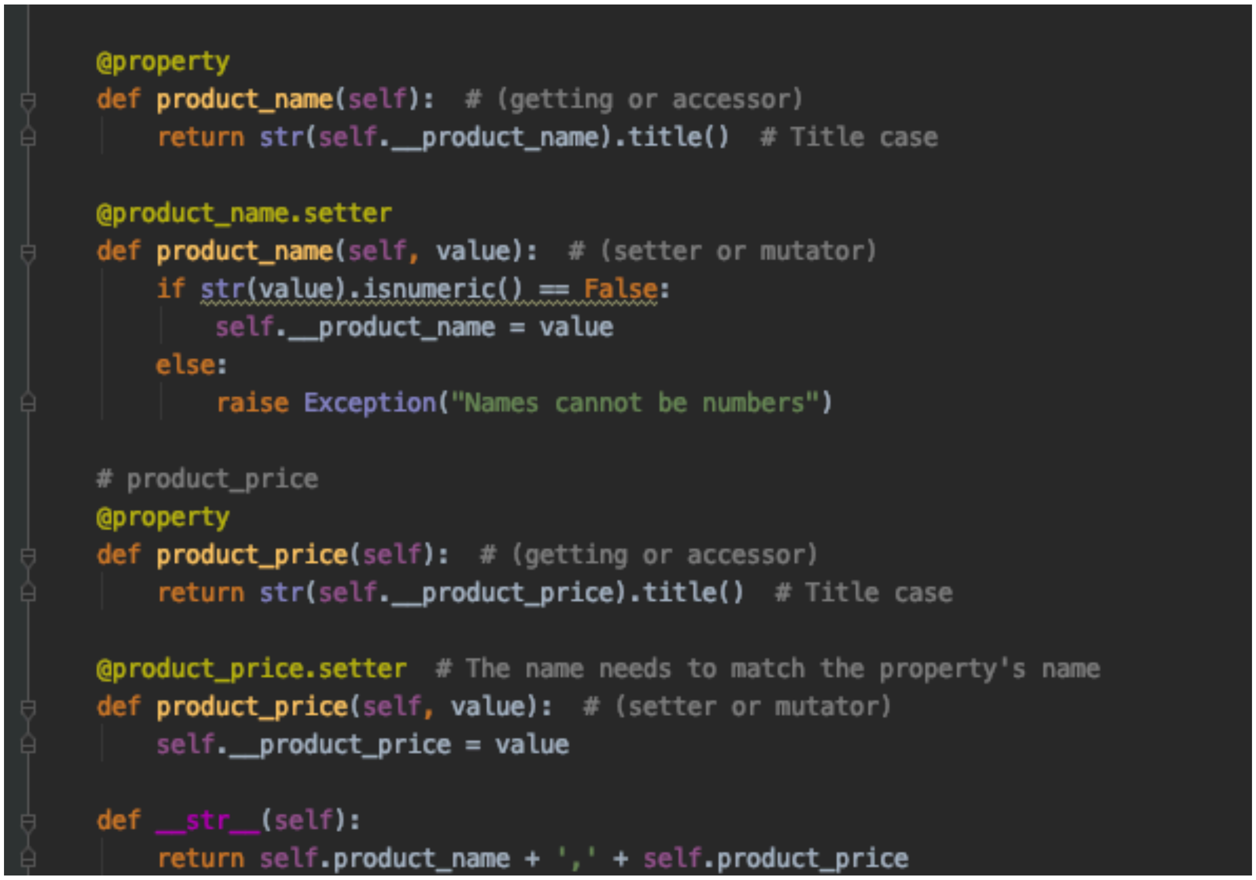
To start the script, I make sure the script has a proper heading and comments and the script defines all appropriate variables so they are available throughout the script. As stated earlier, a medium amount of the code was pre-filled and so all the variables were pre-defined. The local file “products.txt” was also created before running the script.



```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
# ChrisPerry, 11.26.2019, Initial Edits
# ChrisPerry, 11.27.2019, Final Edits
# ------------------------------------------------------------------------ #

# Data ------------------------------------------------------------------- #

# Declare variables and constants
strFileName = "products.txt"
lstOfProductObjects = []
objFile = None
dicRow = {}
strChoice = ""
strProduct = ""
strPrice = ""

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
        ChrisPerry, 11.26.2019, Initial Edits
        ChrisPerry, 11.27.2019, Final Edits
    """
    pass

    # -- Constructor --
    def __init__(self, product_name, product_price):
        #-- Attributes --
        self.product_name = product_name
        self.product_price = product_price
```

Also shown above in our first screenshot is the beginning of our first class, our product class. I also provided a docstrings section to let anyone that picks up this script have the ability to read the comments and know what's going on.

We first set up our constructor and then define it's attributes.



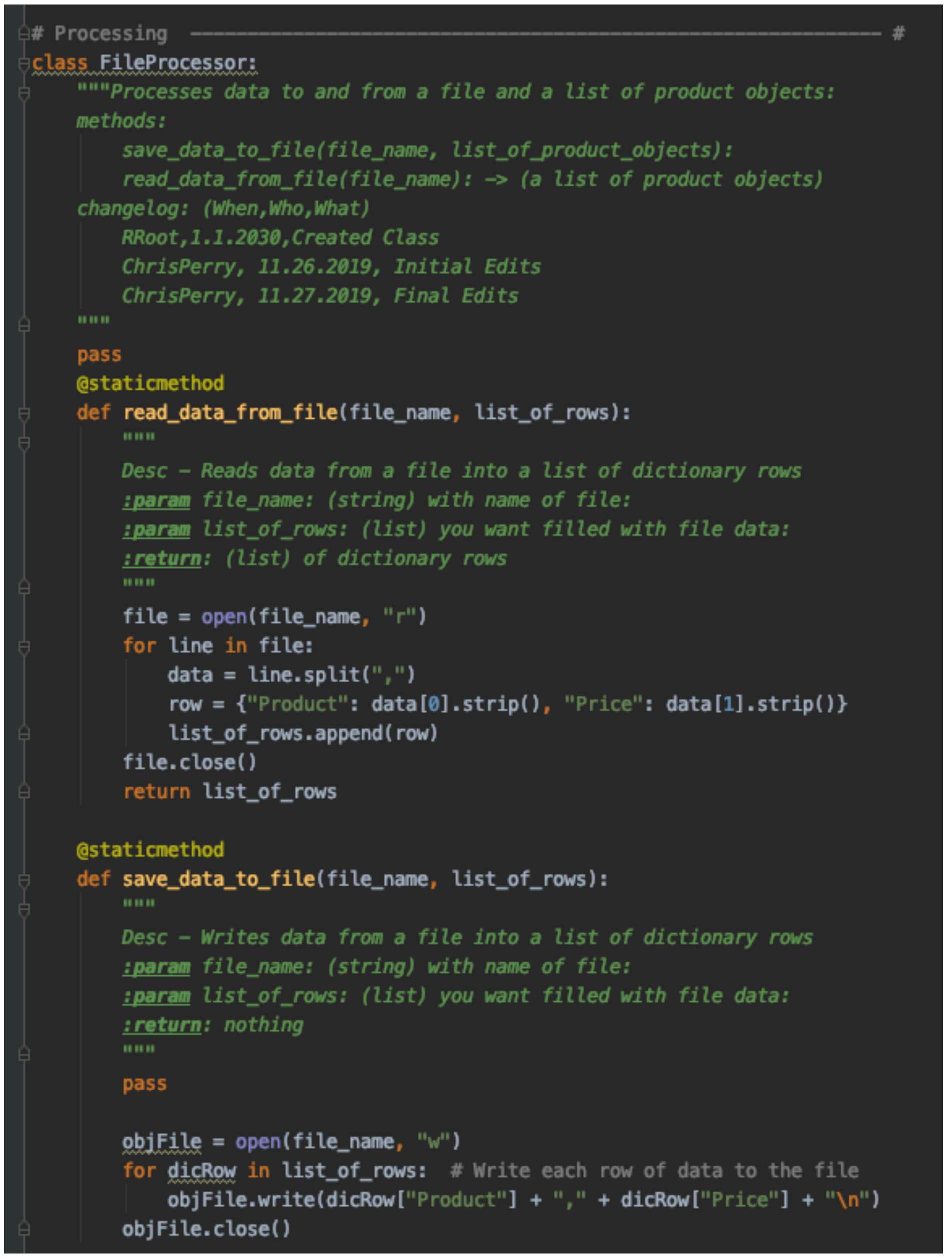
```
    @property
    def product_name(self):  # (getting or accessor)
        return str(self.__product_name).title()  # Title case

    @product_name.setter
    def product_name(self, value):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__product_name = value
        else:
            raise Exception("Names cannot be numbers")

    # product_price
    @property
    def product_price(self):  # (getting or accessor)
        return str(self.__product_price).title()  # Title case

    @product_price.setter  # The name needs to match the property's name
    def product_price(self, value):  # (setter or mutator)
        self.__product_price = value

    def __str__(self):
        return self.product_name + ',' + self.product_price
```

Once the attributes are assigned, we then have to set what are known as getters and setters up so our script can properly read our classes in our code. Above, we have 2 getters and 2 setters. They are called product_name and product_price. We first "get" our property names and then we "set" their values.

At the end of this screen, we make the results a bit neater and make their output comma separate values.

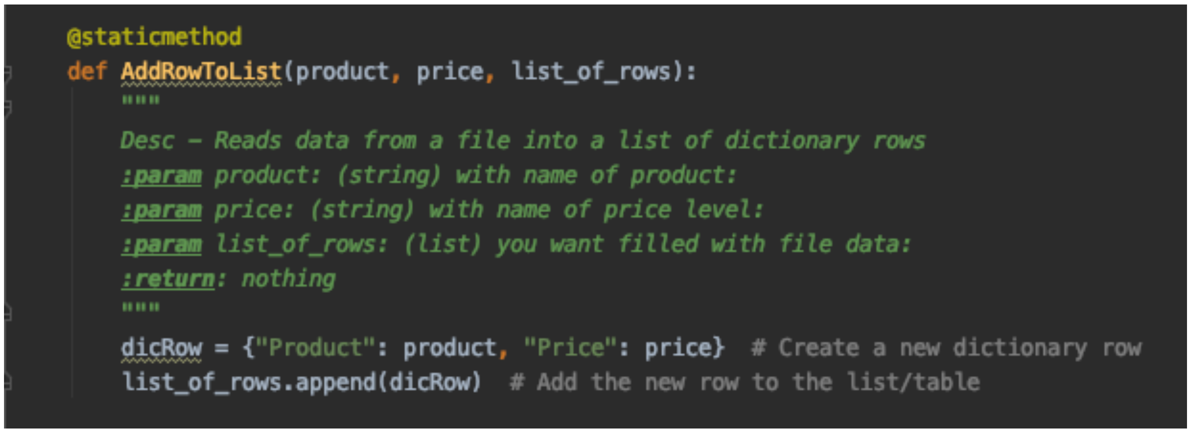
Next we move on to our FileProcessor class. As mentioned before, most of this was written already in assignment 6 and instead of reinventing the wheel, was used again here.

Our code in this Class has 3 functions all of which uses arguments. The first function (read_data_from_file) is what we use to read the data from our current list. Our second function (save_data_to_file), as our docstrings suggest, writes data from a file into a list of dictionary rows.



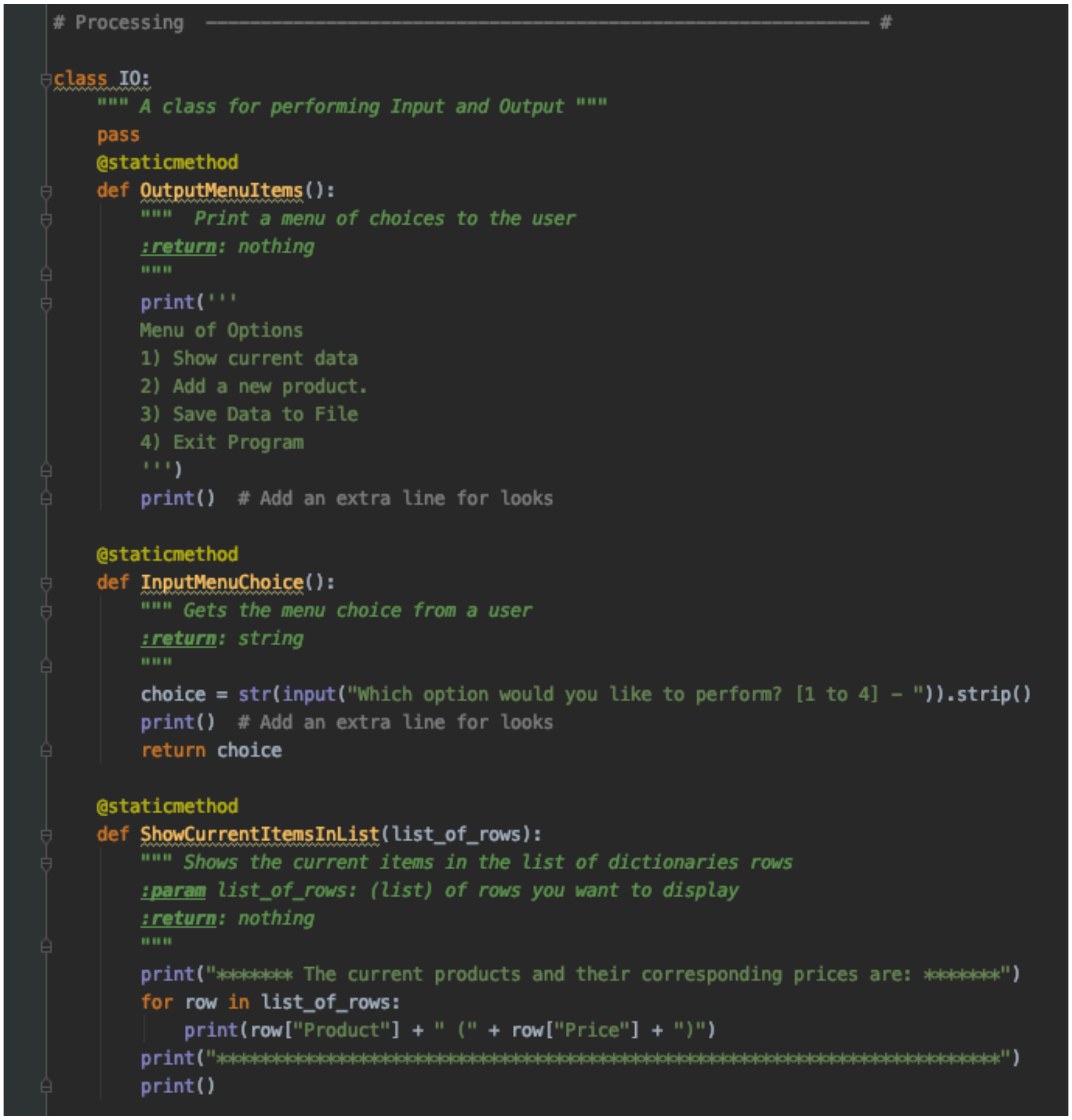
```
# Processing  ---------------------------------------------------------------- #
class FileProcessor:
    """Processes data to and from a file and a list of product objects:
    methods:
        save_data_to_file(file_name, list_of_product_objects):
        read_data_from_file(file_name): -> (a list of product objects)
    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        ChrisPerry, 11.26.2019, Initial Edits
        ChrisPerry, 11.27.2019, Final Edits
    """
    pass
    @staticmethod
    def read_data_from_file(file_name, list_of_rows):
        """
        Desc - Reads data from a file into a list of dictionary rows
        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        file = open(file_name, "r")
        for line in file:
            data = line.split(",")
            row = {"Product": data[0].strip(), "Price": data[1].strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows

    @staticmethod
    def save_data_to_file(file_name, list_of_rows):
        """
        Desc - Writes data from a file into a list of dictionary rows
        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: nothing
        """
        pass

        objFile = open(file_name, "w")
        for dicRow in list_of_rows:  # Write each row of data to the file
            objFile.write(dicRow["Product"] + "," + dicRow["Price"] + "\n")
        objFile.close()
```

Our third and final function in this class is called (AddRowToList) and is what we use add new entries based off user input into memory while our script runs.



```
@staticmethod
def AddRowToList(product, price, list_of_rows):
    """
    Desc - Reads data from a file into a list of dictionary rows
    :param product: (string) with name of product:
    :param price: (string) with name of price level:
    :param list_of_rows: (list) you want filled with file data:
    :return: nothing
    """
    dicRow = {"Product": product, "Price": price}  # Create a new dictionary row
    list_of_rows.append(dicRow)  # Add the new row to the list/table
```

We now move on to a different class, IO. This is a class for performing input and output for the script.



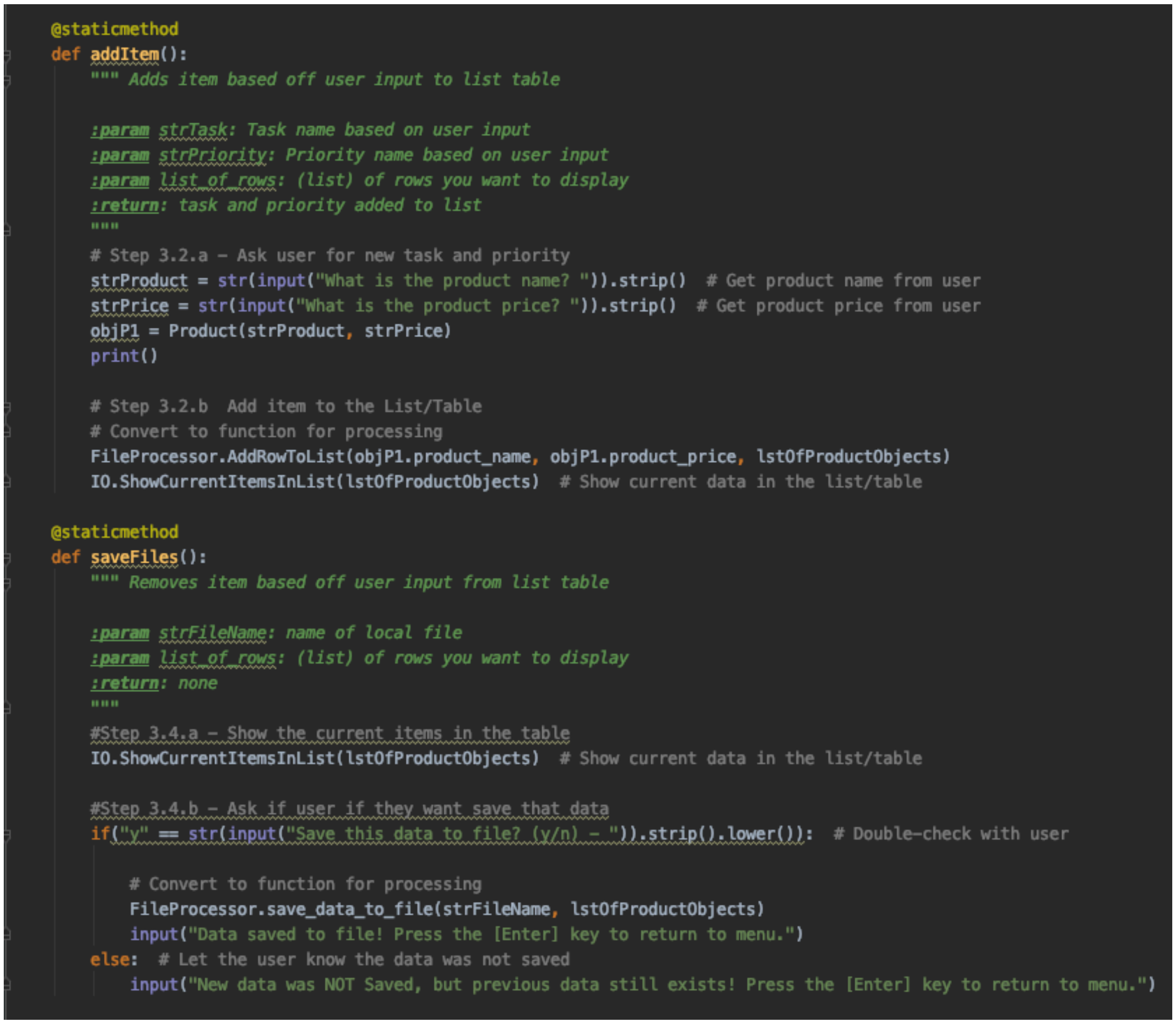
```
# Processing  ------------------------------------------------------------- #

class IO:
    """ A class for performing Input and Output """
    pass
    @staticmethod
    def OutputMenuItems():
        """  Print a menu of choices to the user
        :return: nothing
        """
        print('''
        Menu of Options
        1) Show current data
        2) Add a new product.
        3) Save Data to File
        4) 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 4] - ")).strip()
        print()  # Add an extra line for looks
        return choice

    @staticmethod
    def ShowCurrentItemsInList(list_of_rows):
        """ Shows the current items in the list of dictionaries rows
        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current products and their corresponding prices are: *******")
        for row in list_of_rows:
            print(row["Product"] + " (" + row["Price"] + ")")
        print("*******************************************************************")
        print()
```

Our first function establishes our main menu of the script with our displayed choices the user will see each time they launch the script or finish a task within the script.

Our second function (InputMenuChoice) is a basic function which gathers input from the user using an input function to ask them what choice from our menu would they like to do. Once chosen, the function returns the choice and calls the appropriate function in the script to perform that choice.

The third function pictured here is called ShowCurrentItemsInList and is used very frequently throughout our script. It is the main function used for choice 1 to show the current data, but also is called each time a user finishes one of the pre-defined tasks (i.e, adding a new item). This function is called using the syntax IO.ShowCurrentItemsInList(lstOfProductObjects).

Moving on with our IO class, we have the next 2 functions with the first being addItem(). It takes the code from choice 2 and moves it up to our IO class section. Since this function is just running code and not passing anything, no arguments have to be passed. For this function, we are gathering user input for what they would like to add to the list. This function is also calling 1 other function from our FileProcessor class (AddRowToList) to actually run the code and do the processing.

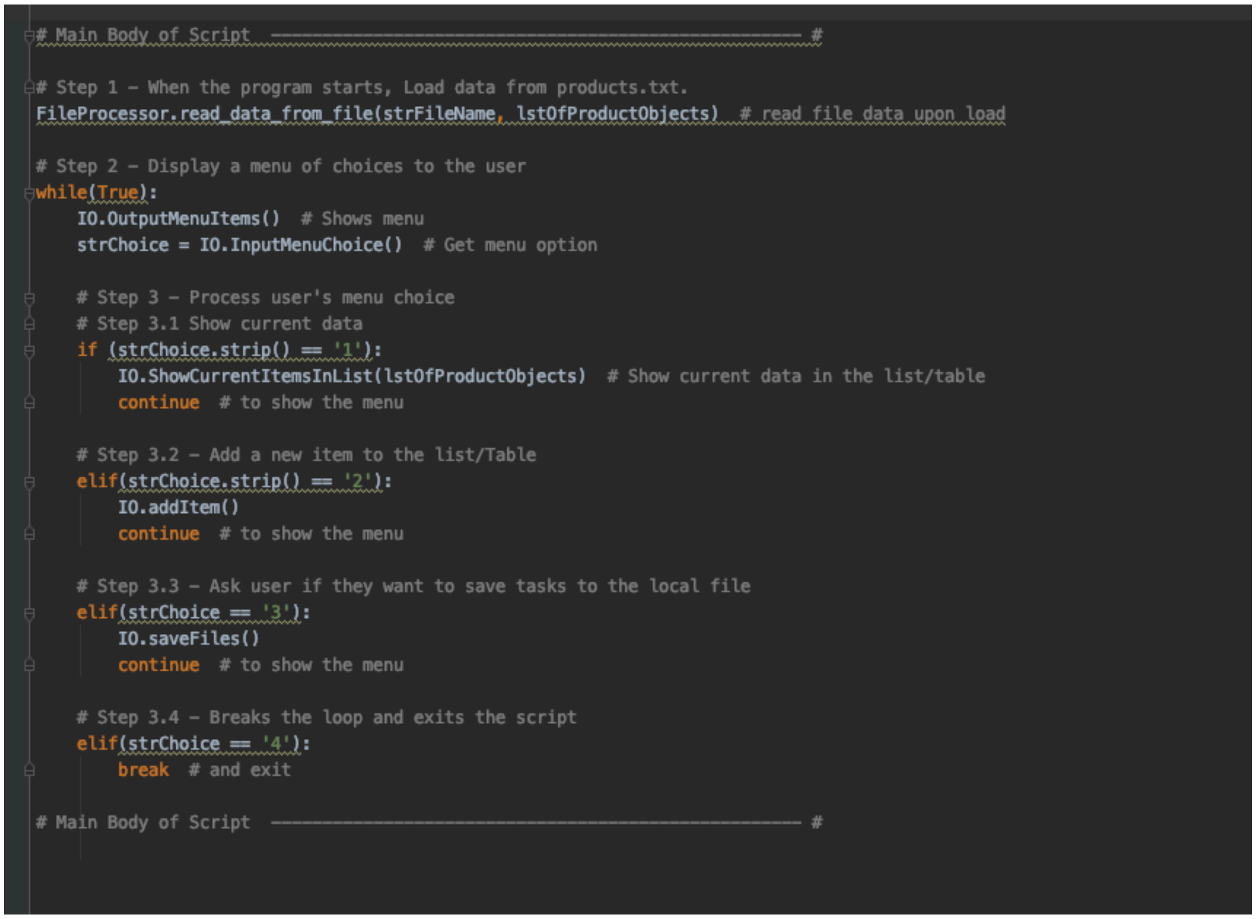


```
    @staticmethod
    def addItem():
        """ Adds item based off user input to list table

        :param strTask: Task name based on user input
        :param strPriority: Priority name based on user input
        :param list_of_rows: (list) of rows you want to display
        :return: task and priority added to list
        """
        # Step 3.2.a - Ask user for new task and priority
        strProduct = str(input("What is the product name? ")).strip()  # Get product name from user
        strPrice = str(input("What is the product price? ")).strip()  # Get product price from user
        objP1 = Product(strProduct, strPrice)
        print()

        # Step 3.2.b  Add item to the List/Table
        # Convert to function for processing
        FileProcessor.AddRowToList(objP1.product_name, objP1.product_price, lstOfProductObjects)
        IO.ShowCurrentItemsInList(lstOfProductObjects)  # Show current data in the list/table

    @staticmethod
    def saveFiles():
        """ Removes item based off user input from list table

        :param strFileName: name of local file
        :param list_of_rows: (list) of rows you want to display
        :return: none
        """
        #Step 3.4.a - Show the current items in the table
        IO.ShowCurrentItemsInList(lstOfProductObjects)  # Show current data in the list/table

        #Step 3.4.b - Ask if user if they want save that data
        if("y" == str(input("Save this data to file? (y/n) - ")).strip().lower()):  # Double-check with user

            # Convert to function for processing
            FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
            input("Data saved to file! Press the [Enter] key to return to menu.")
        else:  # Let the user know the data was not saved
            input("New data was NOT Saved, but previous data still exists! Press the [Enter] key to return to menu.")
```

Our addItem() function takes advantage of our new Product class that we have introduced in this assignment. Because they are indirect objects and not static, we able to assign them as so for processing. We assign an object file (objP1) to the object values of each input as it's entered. Once the function is over, it adds them to the list and memory and is able to loop back again and use the same object variables to assign new values.

For our second function we have saveFiles(). This is called with choice 3 from the menu when our user would like to save the current data in memory to a local txt file. To do the actual processing, we use the FileProcessor.WriteListDataToFile(strFileName, lstOfProductObjects) function from the FileProcessor class and then provide the user with some instructions to advance further in the script.

The script now gets to the main body of the script. In this section, we will see the fruits of our clean up labor and see just how easy it is to read what the script actually does.



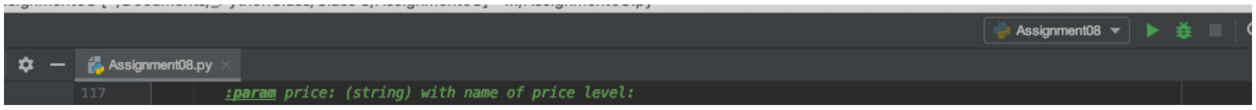
```
# Main Body of Script  ------------------------------------------------------ #

# Step 1 - When the program starts, Load data from products.txt.
FileProcessor.read_data_from_file(strFileName, lstOfProductObjects)  # read file data upon load

# Step 2 - Display a menu of choices to the user
while(True):
    IO.OutputMenuItems()  # Shows menu
    strChoice = IO.InputMenuChoice()  # Get menu option

    # Step 3 - Process user's menu choice
    # Step 3.1 Show current data
    if (strChoice.strip() == '1'):
        IO.ShowCurrentItemsInList(lstOfProductObjects)  # Show current data in the list/table
        continue  # to show the menu

    # Step 3.2 - Add a new item to the list/Table
    elif(strChoice.strip() == '2'):
        IO.addItem()
        continue  # to show the menu

    # Step 3.3 - Ask user if they want to save tasks to the local file
    elif(strChoice == '3'):
        IO.saveFiles()
        continue  # to show the menu

    # Step 3.4 - Breaks the loop and exits the script
    elif(strChoice == '4'):
        break  # and exit

# Main Body of Script  ------------------------------------------------------ #
```

You will probably notice that the code in this section is actually quite minimal. For each choice off the menu of choices, the elif for each choice now just calls a function to run. I decided when making this script that this was my end goal. I wanted the main body of the script to be as clean as possible. By doing this, I was able to move all of the actual code into organized sections of my script to making reading the script overall much easier.

The only function we haven't mentioned yet is choice 4, which when chosen just breaks the loop and exits the script.

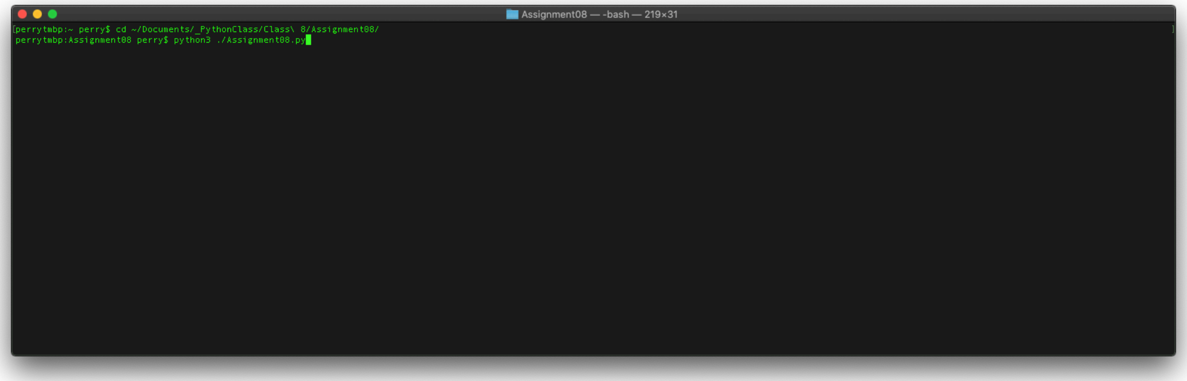
Launching and Saving the Script

We are using PyCharm to run a script for an assignment. You can press the play button in the upper right corner or hit run from the menu bar to run the script. Below is a screenshot of how I run it from PyCharm:



```
:param price: (string) with name of price level:
```

When launching from the shell, since I am using Python3 on my Mac and macOS can have more than one version of Python installed, I have to make sure to designate Python3 when I call the script to run. Also because of the way I saved the path to the txt file that is going to be written, I have to run the script from the directory I want the txt file saved to. Otherwise, I would just need to define the path in the script. Below is a screenshot of how I call the script to run from terminal:

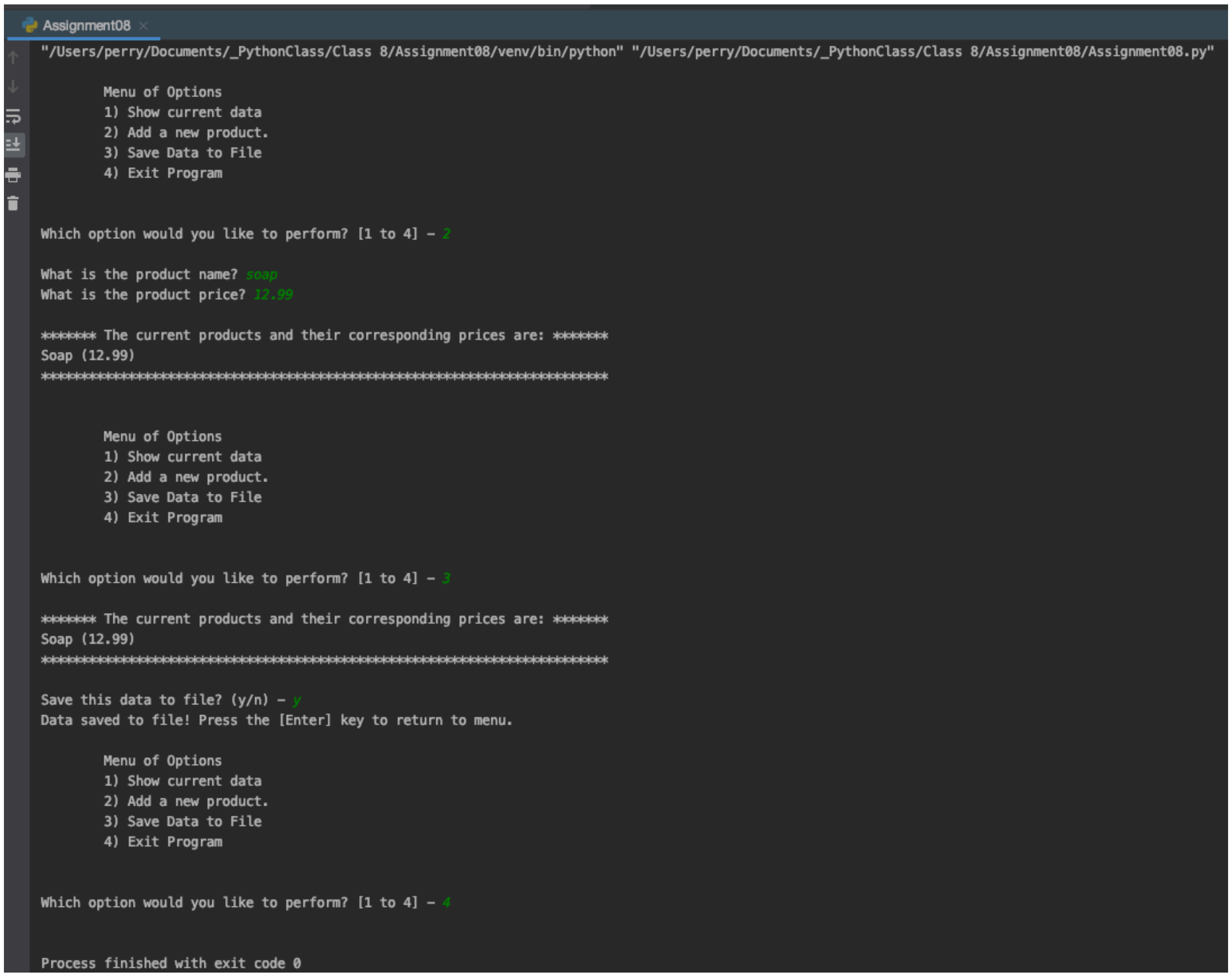


```
perrytmbp:~ perry$ cd ~/Documents/_PythonClass/Class\ 8/Assignment08/
perrytmbp:Assignment08 perry$ python3 ./Assignment08.py
```

The script is saved as a .py file (for python script) in my class directory: /Users/perry/Documents/_PythonClass/Class 6/Assignment08/Assignment08.py

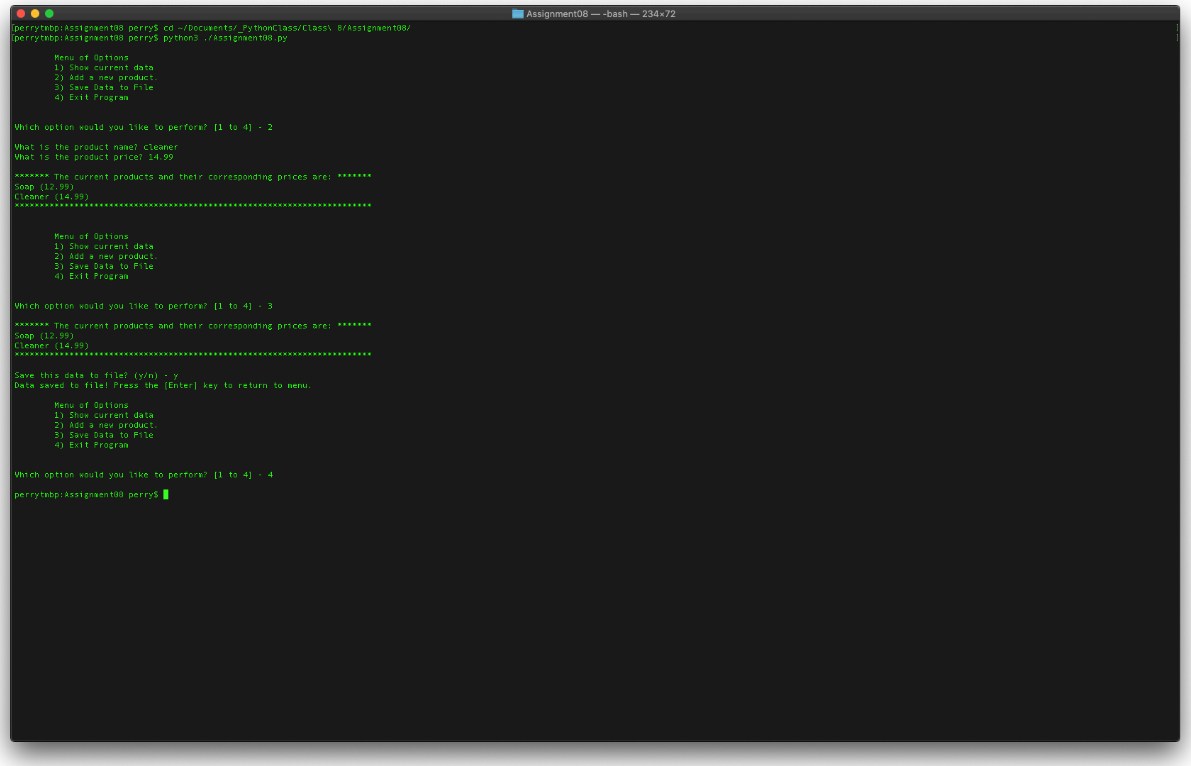
Output of the Final Script

Please see below for a final output screenshot of my Python script for assignment 8. First, the output while running in PyCharm:



```
"/Users/perry/Documents/_PythonClass/Class 8/Assignment08/venv/bin/python" "/Users/perry/Documents/_PythonClass/Class 8/Assignment08/Assignment08.py"

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


Which option would you like to perform? [1 to 4] - 2

What is the product name? soap
What is the product price? 12.99

******* The current products and their corresponding prices are: *******
Soap (12.99)
*************************************************************************


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


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

******* The current products and their corresponding prices are: *******
Soap (12.99)
*************************************************************************

Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

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


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


Process finished with exit code 0
```

Output from running the same script from the shell (terminal) in macOS:



```
perrytmbp:Assignment08 perry$ cd ~/Documents/_PythonClass/Class\ 8/Assignment08/
perrytmbp:Assignment08 perry$ python3 ./Assignment08.py

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


Which option would you like to perform? [1 to 4] - 2

What is the product name? cleaner
What is the product price? 14.99

******* The current products and their corresponding prices are: *******
Soap (12.99)
Cleaner (14.99)
************************************************************************



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


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

******* The current products and their corresponding prices are: *******
Soap (12.99)
Cleaner (14.99)
************************************************************************

Save this data to file? (y/n) - y
Data saved to file! Press the [Enter] key to return to menu.

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


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

perrytmbp:Assignment08 perry$ 
```

Final Summary

During this eighth assignment, I was able to demonstrate how to incorporate object classes to assign value and definition. I was able to class on those classes later to right the name and price the user chosen via their input and wrote it to memory. If the user chose, I was also able to write these values to a local file.