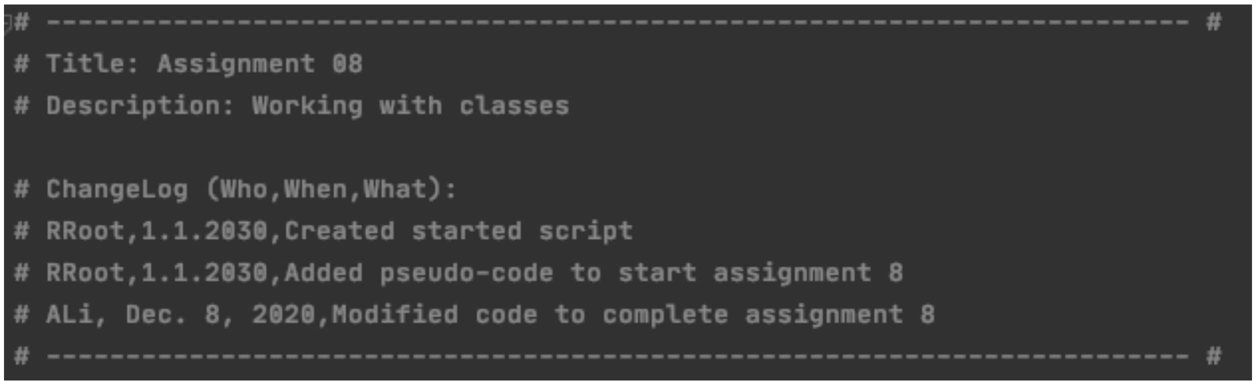
Andrew Li
Dec. 8, 2020
Foundations of Programming: Python
Assignment08
https://github.com/idrew4u/ITFnd100-Mod08

Class Object: Product List Python Script

Introduction
In this assignment, we will review object classes. A starter code has been provided with pseudo-code, outlining what the program should accomplish. After interpreting the pseudo-code, we were asked to provide a script that asks the user to select a choice from the menu that: 1) shows current list, 2) inputs new products to list, 3) Saves the new data to file, and 4) exits the program.

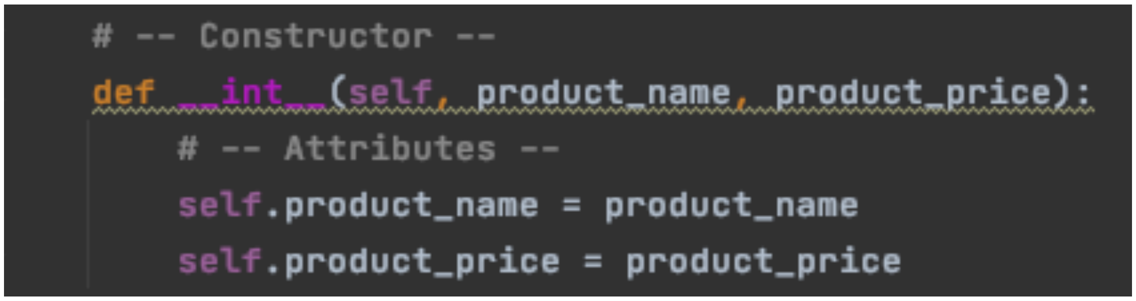
Creating the Script
Script Header
The first thing we should do is write a header for the script. The header is the first thing you see and gives us a basic idea what the script is trying to accomplish. It asks for the title of the program, a description of what it does, and logs changes to the script (Figure 1). The logs should include who made the change, the date it was changed, and what was changed in the script. In this situation, the originator already started the script header, we just had to modify the information to allow other viewers we made updates and changes to the script.



```
# ------------------------------------------------------------------------ #
# 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
# ALi, Dec. 8, 2020,Modified code to complete assignment 8
# ------------------------------------------------------------------------ #
```

Figure 1: The header includes the title, description, and log changes.

Constructors and Attributes
The first thing we did with the script after updating the header was set up out constructor and defined it’s attributes (Figure 2). Constructors are special methods, or functions, that runs automatically when the object from the class is created. Attributes are an invisble field that holds the internal data. Once we assign the constructors and attributes, we have to set up the properties.

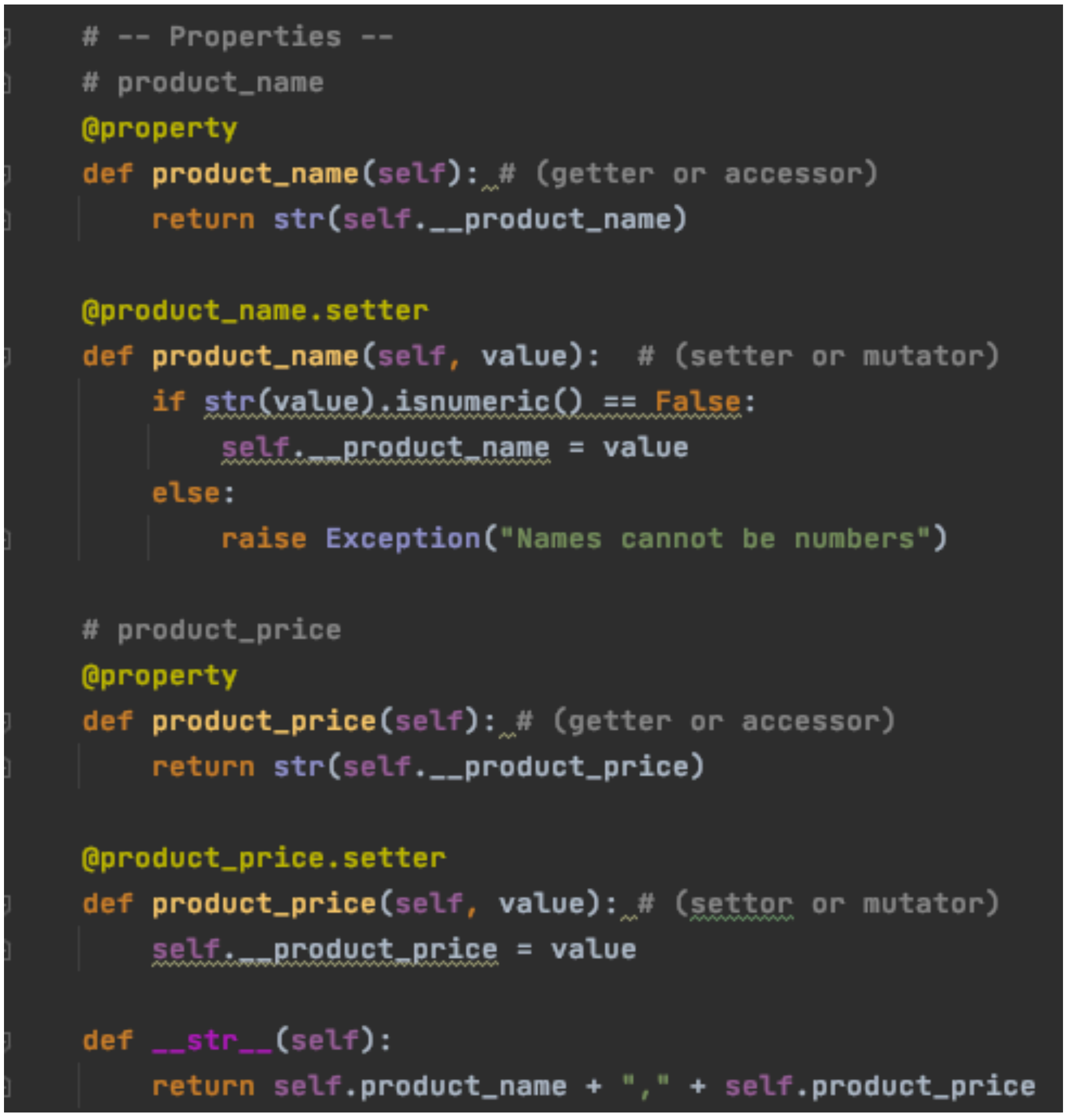


```python
    # -- Constructor --
    def __int__(self, product_name, product_price):
        # -- Attributes --
        self.product_name = product_name
        self.product_price = product_price
```

Figure 2: Defining the constructor and attributes.

Properties

Properties are functions used to manage attribute data. They typically come in pairs, getters(accessors) and setters(mutators). In our case, we used them to "get" the property names, and then we "set" their values (Figure 3).



```python
    # -- Properties --
    # product_name
    @property
    def product_name(self): # (getter or accessor)
        return str(self.__product_name)

    @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): # (getter or accessor)
        return str(self.__product_price)

    @product_price.setter
    def product_price(self, value): # (settor or mutator)
        self.__product_price = value

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

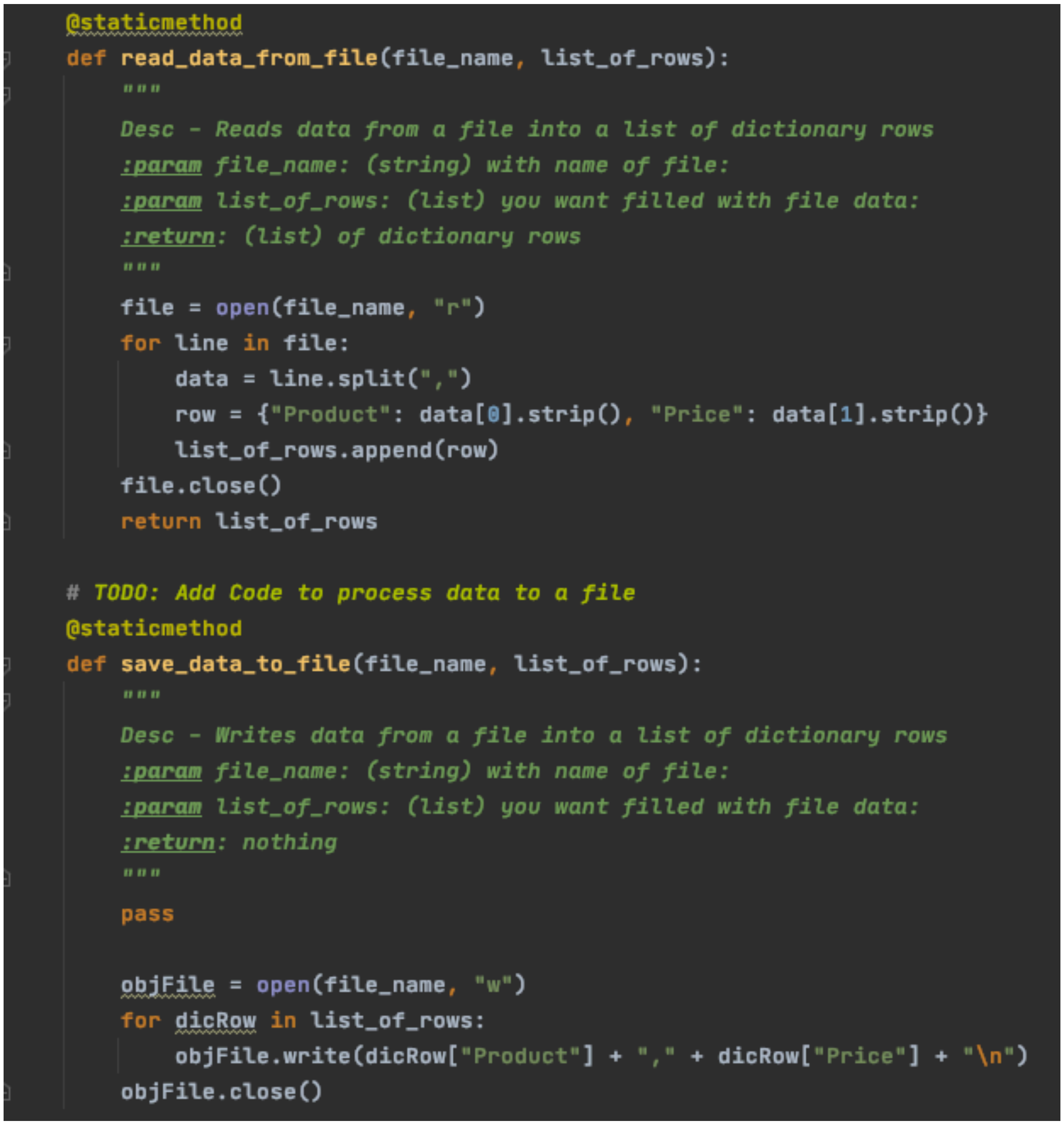
Figure 3: Using getters and setters to "get" the property names and "set" their values.

Methods

Methods are functions inside of the a class that helps us organize the processing statements into named groups. This is where we get to use what we have previously built upon in prior assignments.

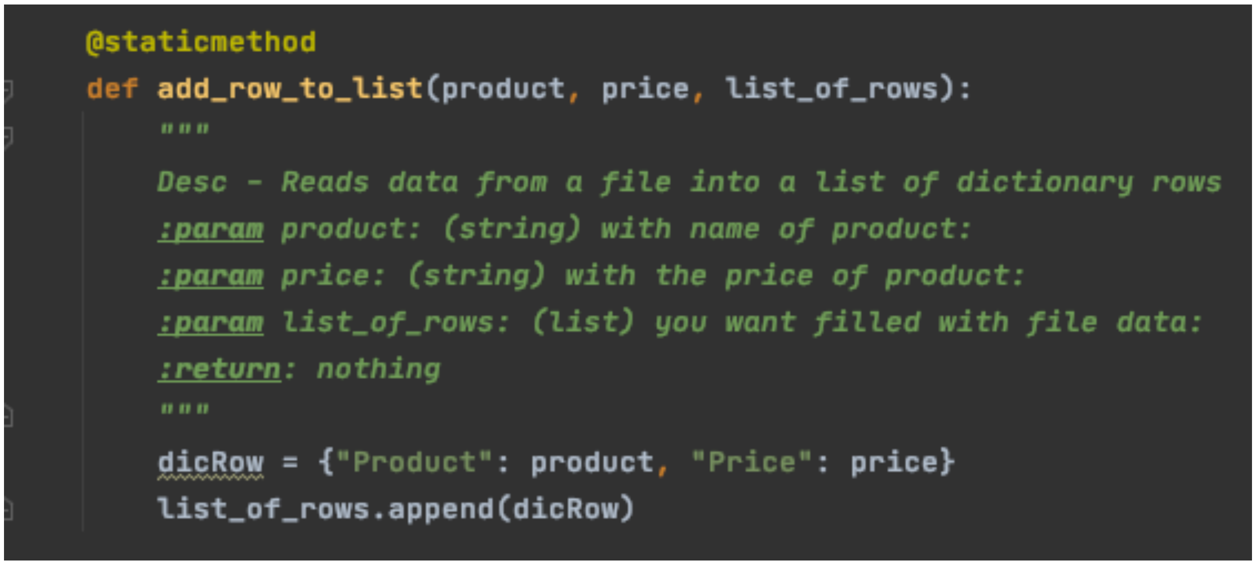
FileProcessor Class

In the FileProcessor Class, we process the data to and from a file. It was outlined that we should include a method or function to read the data from a file, and to save the data to the file (Figure 4). We also want to be able to append the file and add products to the list (Figure 5).



```
    @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

    # TODO: Add Code to process data to a file
    @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:
            objFile.write(dicRow["Product"] + "," + dicRow["Price"] + "\n")
        objFile.close()
```

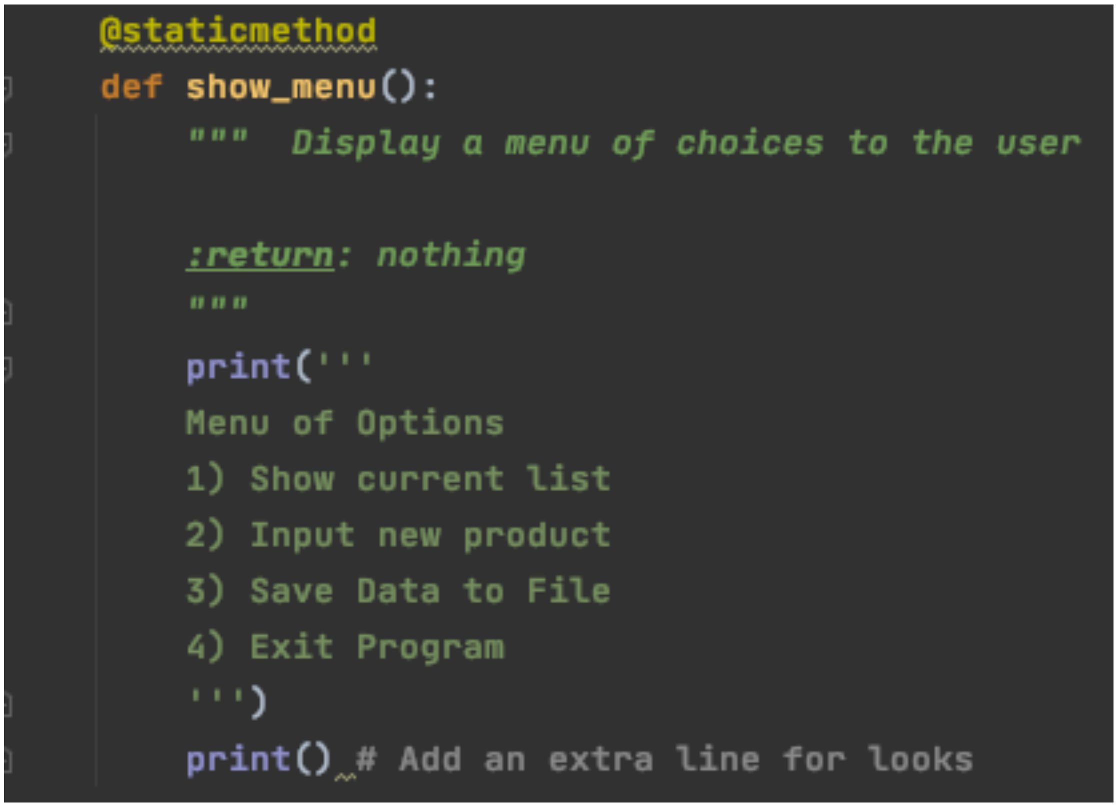
Figure 4: Methods to read and save data to file.



```
    @staticmethod
    def add_row_to_list(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 the price of product:
        :param list_of_rows: (list) you want filled with file data:
        :return: nothing
        """
        dicRow = {"Product": product, "Price": price}
        list_of_rows.append(dicRow)
```

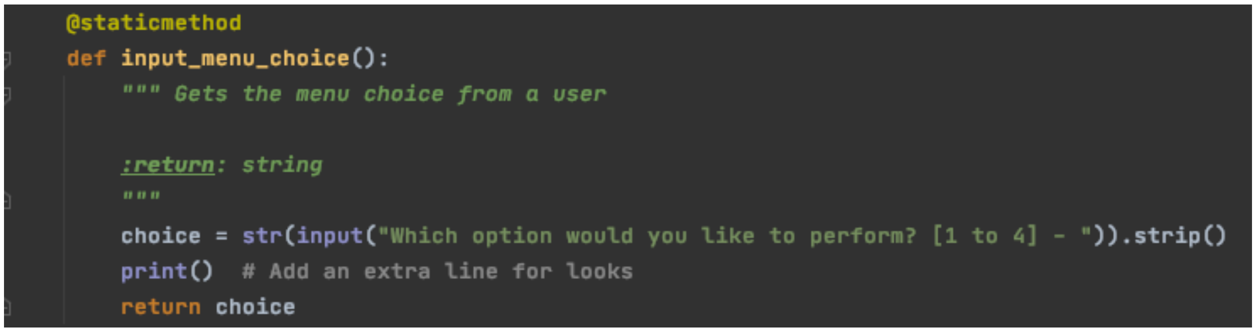
Figure 5: Method to append or add entries to the file.

IO Class
For the code, the IO class does most of the grunt work. This area is where we display the menu (Figure 6), gather input from the user (Figure 7 and 9), print the list of things gathered (Figure 8), and in the end, save the information (Figure 10).



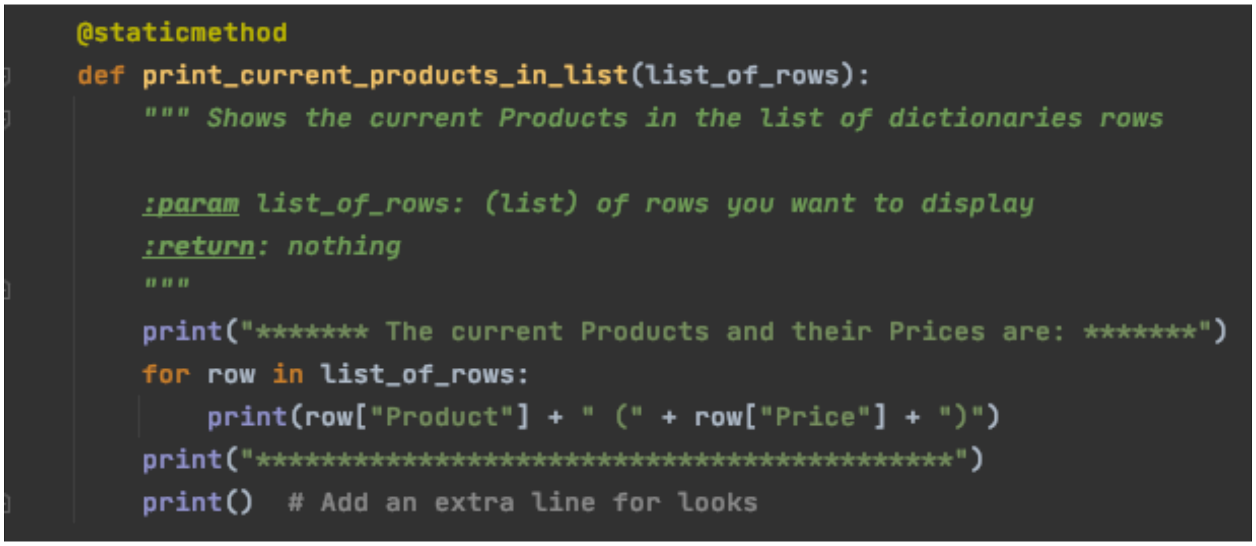
```
    @staticmethod
    def show_menu():
        """  Display a menu of choices to the user

        :return: nothing
        """
        print('''
        Menu of Options
        1) Show current list
        2) Input new product
        3) Save Data to File
        4) Exit Program
        ''')
        print() # Add an extra line for looks
```

Figure 6: Displaying menu to user.



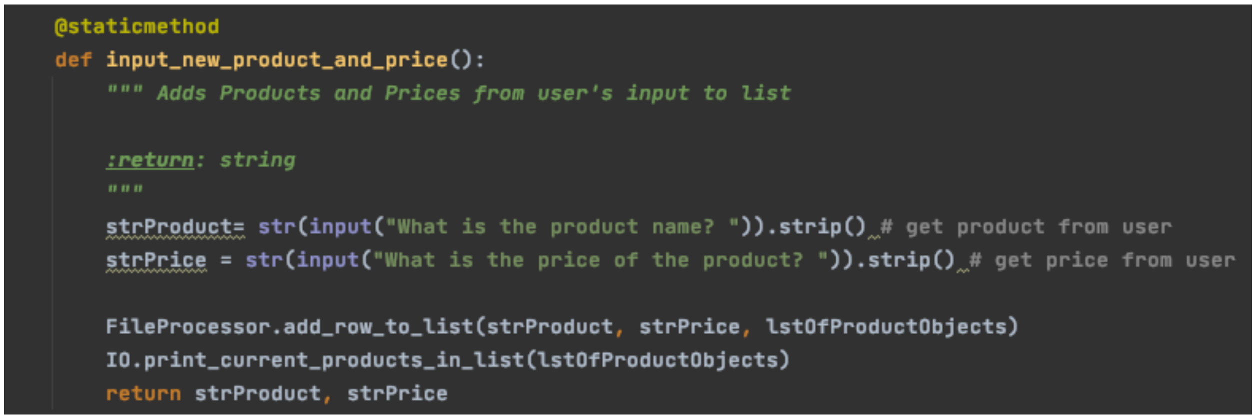
```python
    @staticmethod
    def input_menu_choice():
        """ Gets the menu choice from a user

        :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
```

Figure 7: Gathering menu choice from user.



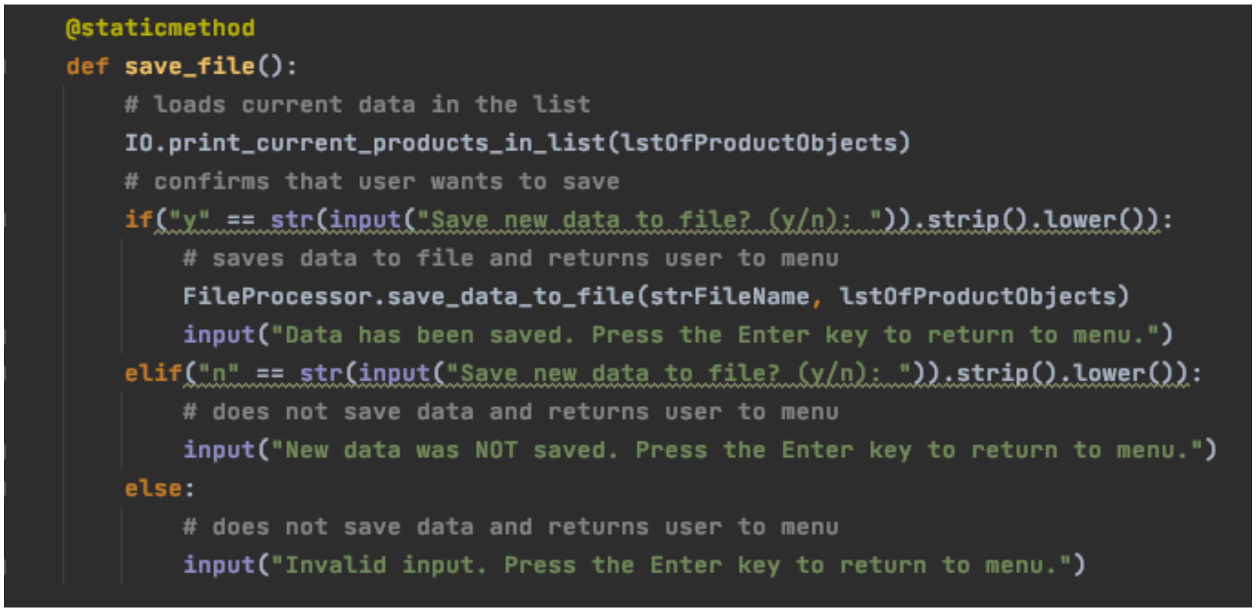
```python
    @staticmethod
    def print_current_products_in_list(list_of_rows):
        """ Shows the current Products in the list of dictionaries rows

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
        """
        print("******* The current Products and their Prices are: *******")
        for row in list_of_rows:
            print(row["Product"] + " (" + row["Price"] + ")")
        print("*******************************************")
        print()  # Add an extra line for looks
```

Figure 8: Displaying current products and prices on the list.



```python
    @staticmethod
    def input_new_product_and_price():
        """ Adds Products and Prices from user's input to list

        :return: string
        """
        strProduct= str(input("What is the product name? ")).strip() # get product from user
        strPrice = str(input("What is the price of the product? ")).strip() # get price from user

        FileProcessor.add_row_to_list(strProduct, strPrice, lstOfProductObjects)
        IO.print_current_products_in_list(lstOfProductObjects)
        return strProduct, strPrice
```

Figure 9: Gathering additional entries for products and their prices to add to the list.

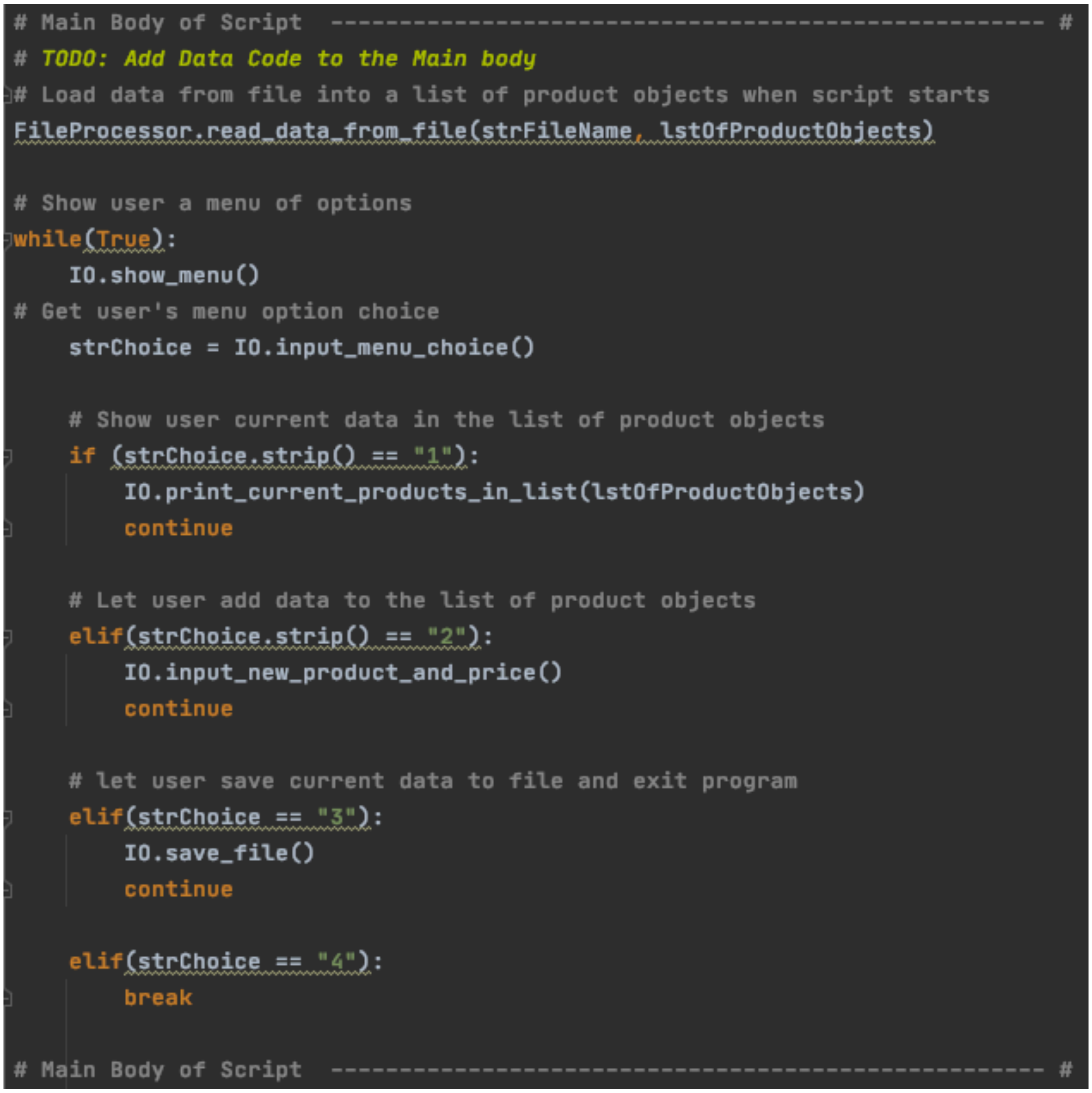


```
@staticmethod
def save_file():
    # loads current data in the list
    IO.print_current_products_in_list(lstOfProductObjects)
    # confirms that user wants to save
    if("y" == str(input("Save new data to file? (y/n): ")).strip().lower()):
        # saves data to file and returns user to menu
        FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
        input("Data has been saved. Press the Enter key to return to menu.")
    elif("n" == str(input("Save new data to file? (y/n): ")).strip().lower()):
        # does not save data and returns user to menu
        input("New data was NOT saved. Press the Enter key to return to menu.")
    else:
        # does not save data and returns user to menu
        input("Invalid input. Press the Enter key to return to menu.")
```

Figure 10: Saving the new entries to a file.

Body of Script

Then it comes to the body of the script. The body of the script basically organizes everything for us. It combines everything we have written into one place. We use this portion to refer back to the FileProcessor and IO classes to make the script run the way we want it to (Figure 11).

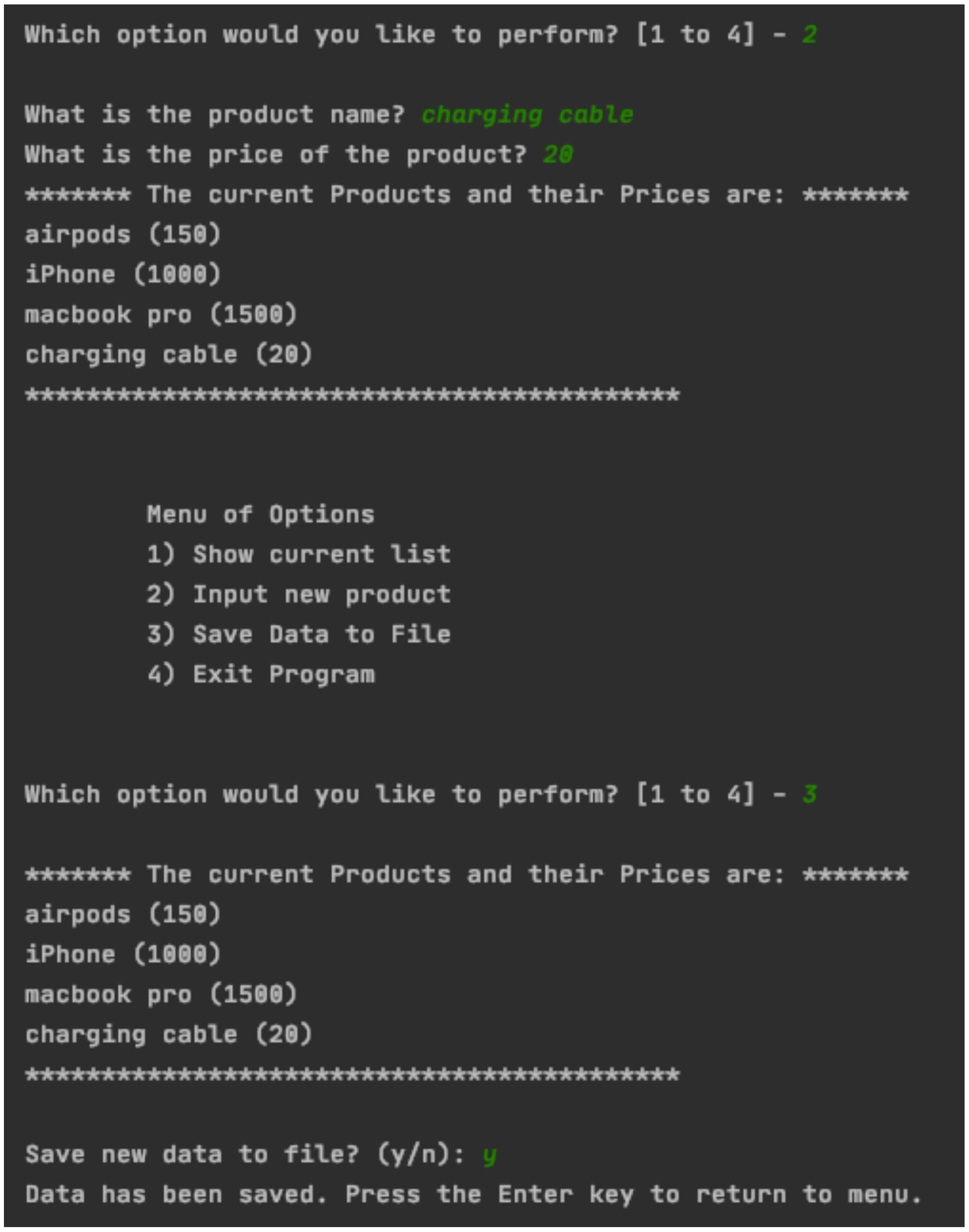


```
# Main Body of Script  ---------------------------------------------------- #
# TODO: Add Data Code to the Main body
# Load data from file into a list of product objects when script starts
FileProcessor.read_data_from_file(strFileName, lstOfProductObjects)

# Show user a menu of options
while(True):
    IO.show_menu()
# Get user's menu option choice
    strChoice = IO.input_menu_choice()

    # Show user current data in the list of product objects
    if (strChoice.strip() == "1"):
        IO.print_current_products_in_list(lstOfProductObjects)
        continue

    # Let user add data to the list of product objects
    elif(strChoice.strip() == "2"):
        IO.input_new_product_and_price()
        continue

    # let user save current data to file and exit program
    elif(strChoice == "3"):
        IO.save_file()
        continue

    elif(strChoice == "4"):
        break

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

Figure 11: Mainly a combination of both FileProcessor and IO classes to build the body of the script.

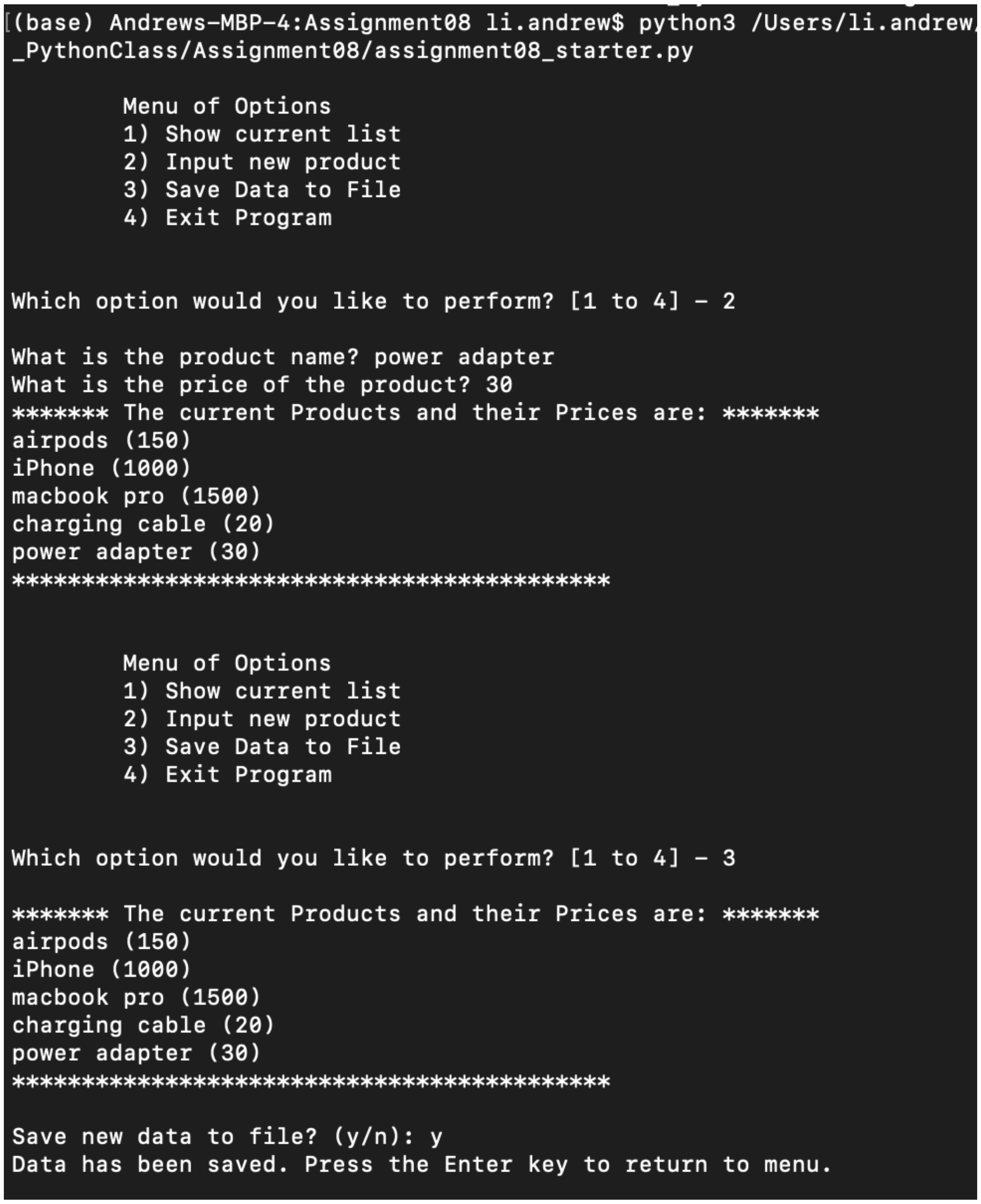
Running the Program

After creating the script, we verified the program is working properly by running it in PyCharm (Figure 12) and the terminal command window on macOS (Figure 13).



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

What is the product name? charging cable
What is the price of the product? 20
******* The current Products and their Prices are: *******
airpods (150)
iPhone (1000)
macbook pro (1500)
charging cable (20)
*******************************************



        Menu of Options
        1) Show current list
        2) Input 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 Prices are: *******
airpods (150)
iPhone (1000)
macbook pro (1500)
charging cable (20)
*******************************************

Save new data to file? (y/n): y
Data has been saved. Press the Enter key to return to menu.
```

Figure 12: The results after running our program in the PyCharm IDE.



```
[(base) Andrews-MBP-4:Assignment08 li.andrew$ python3 /Users/li.andrew
_PythonClass/Assignment08/assignment08_starter.py

        Menu of Options
        1) Show current list
        2) Input 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? power adapter
What is the price of the product? 30
******* The current Products and their Prices are: *******
airpods (150)
iPhone (1000)
macbook pro (1500)
charging cable (20)
power adapter (30)
*******************************************


        Menu of Options
        1) Show current list
        2) Input 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 Prices are: *******
airpods (150)
iPhone (1000)
macbook pro (1500)
charging cable (20)
power adapter (30)
*******************************************

Save new data to file? (y/n): y
Data has been saved. Press the Enter key to return to menu.
```

Figure 13: The results after running our program in the terminal command shell.

Summary

We were able to take a concept written by the originator and build a whole script around the outline. We were able to incorporate object classes and assign them value and definition. We were able to have the user choose from a menu, add input, and save the data to a file.