Chrissy Henderson

December 6, 2020

IT FDN 110 A Au 20: Foundations Of Programming: Python

Assignment 08

https://github.com/chrissyhend1/IntroToProg-Python-Mod08

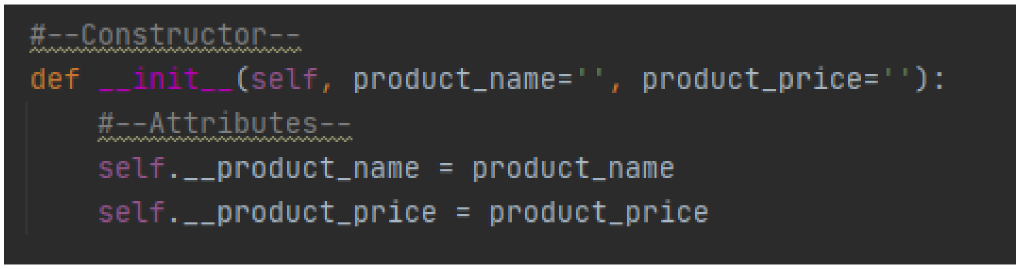
Demonstrating Classes and DocStrings in a Python Program

Introduction

In this module, we learned about the components of a class and DocStrings. Classes group similar data and functions together in one place with a name to denote its purpose. In this assignment, I created two classes to help complete a file reading and writing program.

Class Product

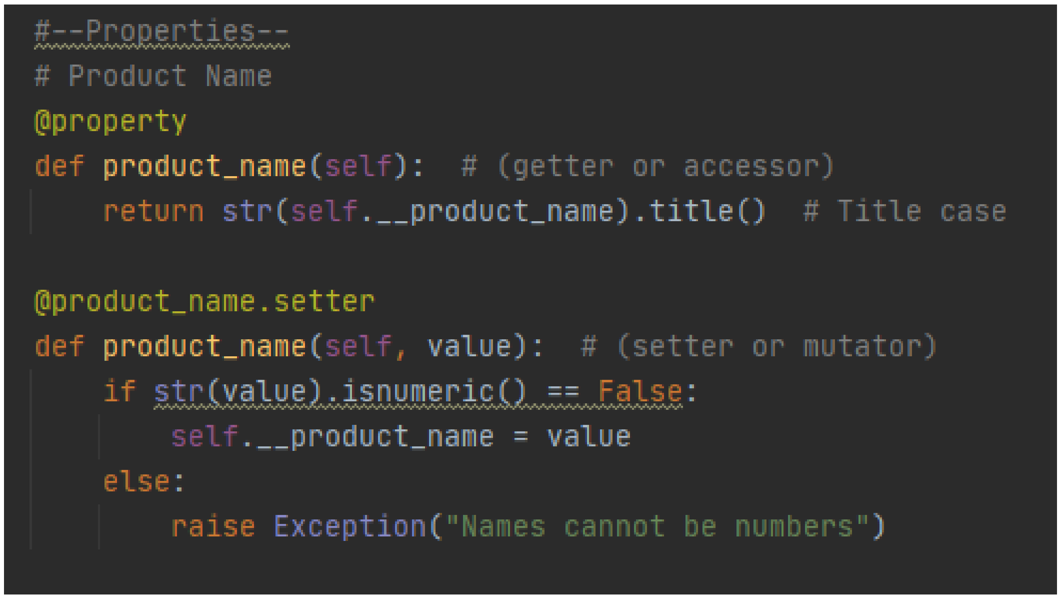
The first class works to set the initial values about a product including its name and price. In the Constructor section, these two variables are created using an "init" method with a duder before and after the "init" function word (Figure 1). When this class is referenced, Python will automatically run this method. The self keyword is important to include in that it is referring to the data found in this particular object instance inside and outside the Product class. The Attributes section sets up the names that the variables will be referred to throughout the class as well.



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

Figure 1. The Constructor which creates the product name and price variable

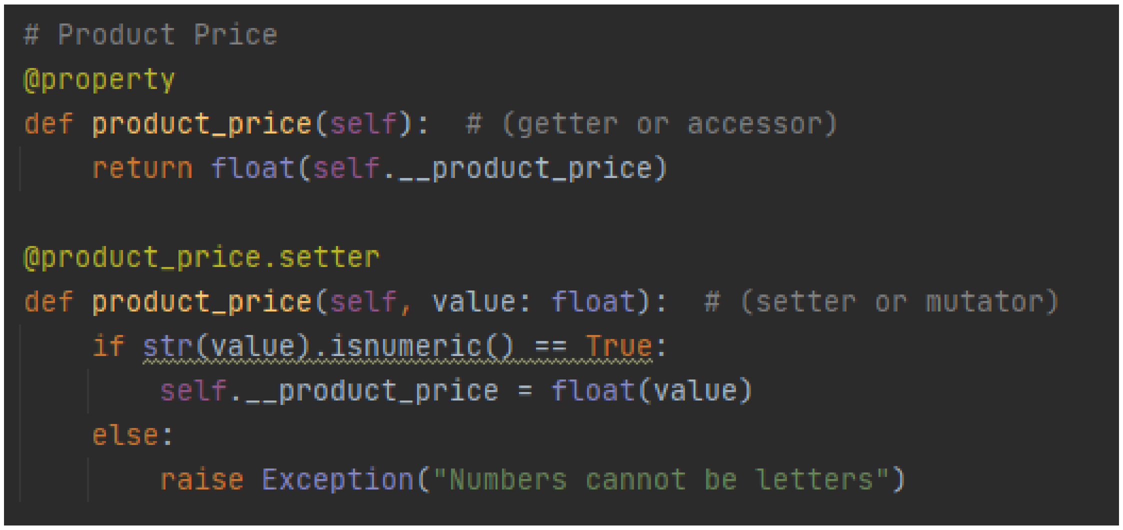
Next comes the Properties set-up for the name of the product. The getter will be the first to set up and runs a function to retrieve the product name and then return it to the program in title case (Figure 2). The setter in the second part of the Properties section will take the value of the variable and raise an exception if the name is numeric.



```python
#--Properties--
# Product Name
@property
def product_name(self):  # (getter 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")
```

Figure 2. Getter and setter set up for the product name variable

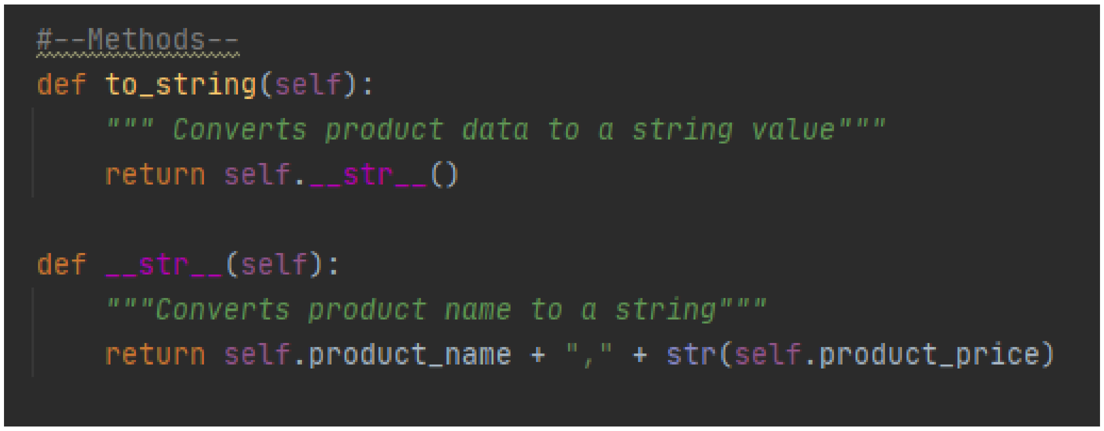
Ths process is repeated for the product price variable except this variable will be set up as a float variable rather than a string (Figure 3).



```python
# Product Price
@property
def product_price(self):  # (getter or accessor)
    return float(self.__product_price)

@product_price.setter
def product_price(self, value: float):  # (setter or mutator)
    if str(value).isnumeric() == True:
        self.__product_price = float(value)
    else:
        raise Exception("Numbers cannot be letters")
```

Figure 3. Getter and setter set up for the product price variable

Next in the Methods section, two functions will be created: one will convert product data to a string value while the other is a built in function that will be called on to return a product name and the price with a comma between them (Figure 4).



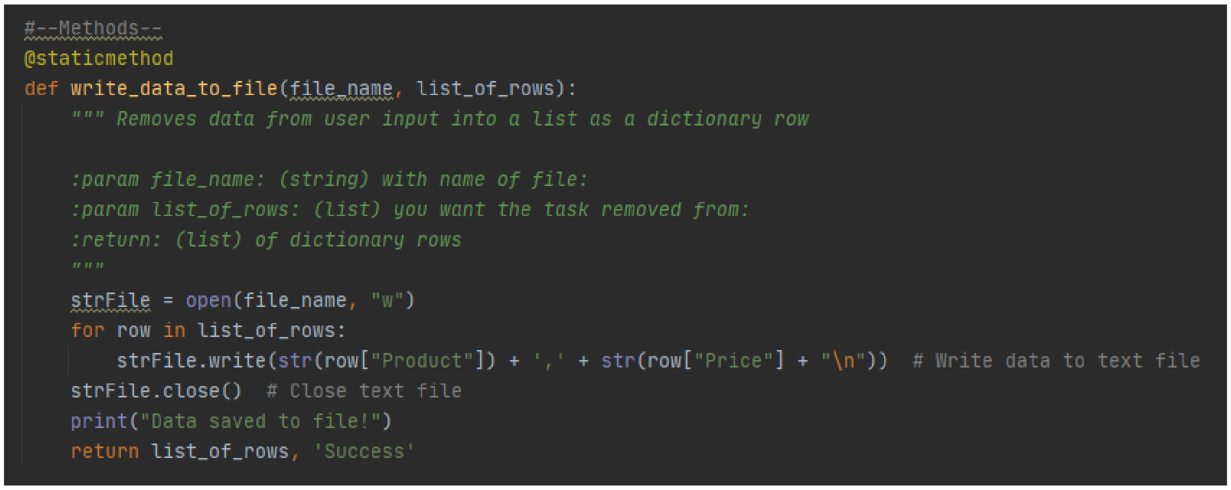
```
#--Methods--
def to_string(self):
    """ Converts product data to a string value"""
    return self.__str__()

def __str__(self):
    """Converts product name to a string"""
    return self.product_name + "," + str(self.product_price)
```

Figure 4. Methods section of the Product class that converts product data to a string type

Class FileProcessor

The class FileProcessor creates functions to save data to a file, read data from a file, and add data to a list. This class is made up of exclusively methods. The first function requires a file name and the "list_of_rows" that the user wants written to the file (Figure 5). When run, the function will open the file that has the name of the variable "file_name" in write mode, will format the "list_of_rows" data, and will then write it to the file in the file variable created earlier. If the file does not exist, the program will create it when the file is opened.

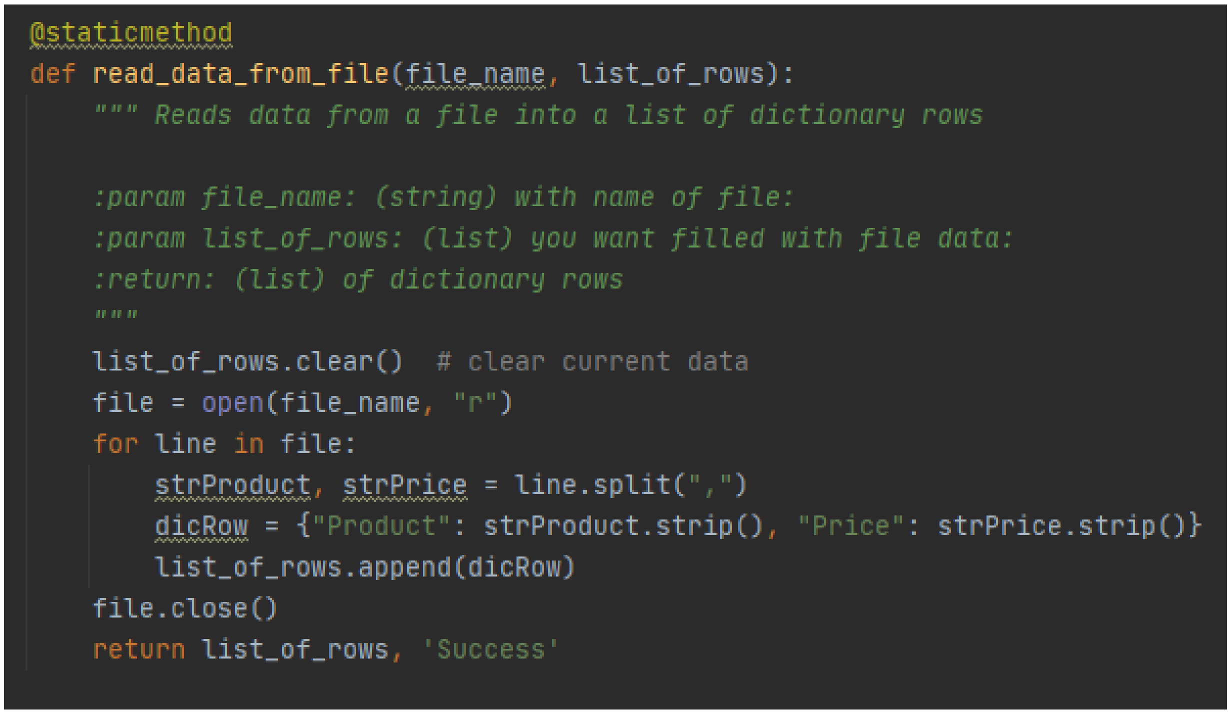
After this, the file is closed, and the program prints "Data saved to file!" to let the user know that the data was saved. I made sure to put this line after the "write" and "close" lines so that if something goes wrong with writing the file, the user does not see the printed statement before this error. The function returns the "list_of_rows" on a successful run.



```
#--Methods--
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Removes data from user input into a list as a dictionary row

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want the task removed from:
    :return: (list) of dictionary rows
    """
    strFile = open(file_name, "w")
    for row in list_of_rows:
        strFile.write(str(row["Product"]) + ',' + str(row["Price"] + "\n"))  # Write data to text file
    strFile.close()  # Close text file
    print("Data saved to file!")
    return list_of_rows, 'Success'
```

Figure 5. The function in the FileProcessor class that writes data to a file

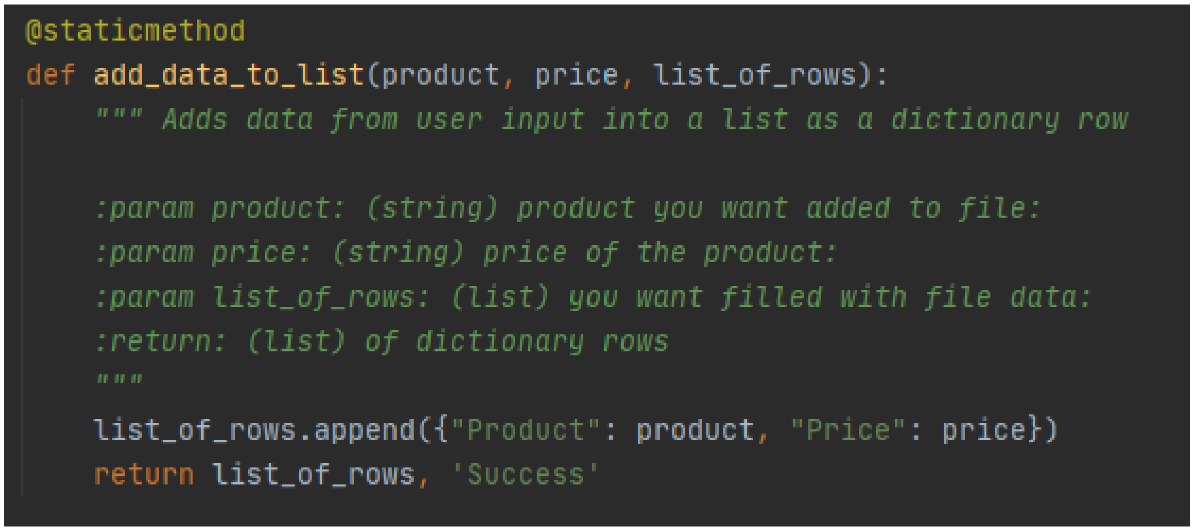
The next function to include was written to load in a text file as a table that could be referenced later in the program (Figure 6). I first wanted to tell the program to load in the file and then run through the file line by line using a loop. For each line, the loop would read in the data as a string, convert it to a dictionary line, and append the data as dictionary data to the "list_of_rows" object before closing the file.



```python
@staticmethod
def read_data_from_file(file_name, list_of_rows):
    """ 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
    """
    list_of_rows.clear()  # clear current data
    file = open(file_name, "r")
    for line in file:
        strProduct, strPrice = line.split(",")
        dicRow = {"Product": strProduct.strip(), "Price": strPrice.strip()}
        list_of_rows.append(dicRow)
    file.close()
    return list_of_rows, 'Success'
```

Figure 6. The function in the FileProcessor class that reads data from a file and loads it into a table

The last function in this class is meant to append data to a list. When run, the function is going to look for a variable declared as “product”, “price”, and “list_of_rows”. With those three variables, the program will append a dictionary row of the “product” and “price” variables to the “list_of_rows” variable and return the “list_of_rows” to the program on a successful run. When it returns a variable, it does not print it to the user, but just keeps that variable for the program to call on in whatever the next step is. Additionally, I added some parameter descriptors at the start of the function to help the user figure out what each variable was.

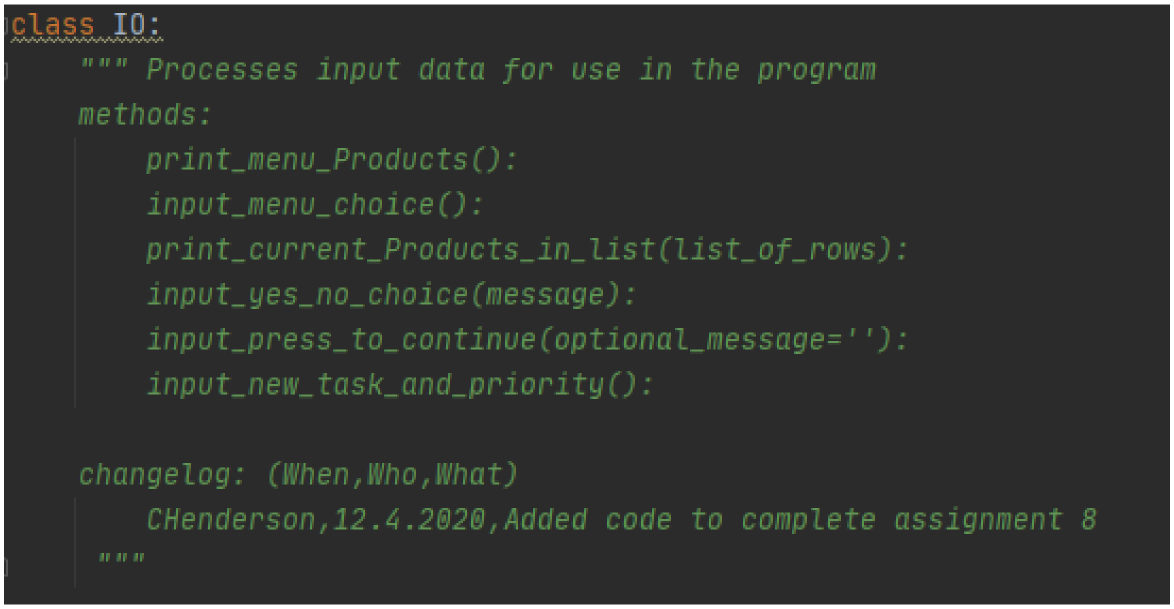


```
@staticmethod
def add_data_to_list(product, price, list_of_rows):
    """ Adds data from user input into a list as a dictionary row

    :param product: (string) product you want added to file:
    :param price: (string) price of the product:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    list_of_rows.append({"Product": product, "Price": price})
    return list_of_rows, 'Success'
```

Figure 7. The function in the FileProcessor class that appends data to a list

Class IO

The Class IO is meant to process input data for use in the program. I added a DocString to this class to further describe it to the user what the purpose of the class is and present the different methods that are present in the class as well as who was the last person to change the code and when (Figure 8).



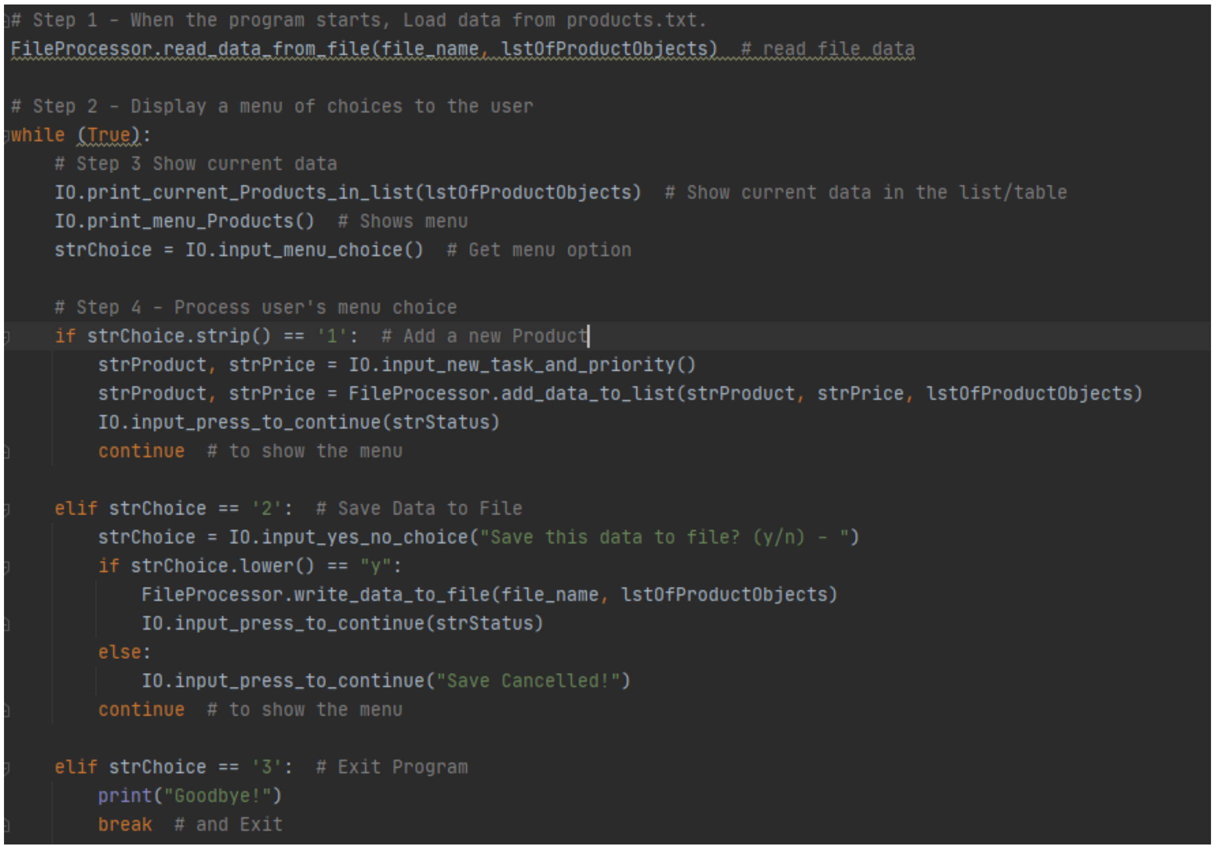
```
class IO:
    """ Processes input data for use in the program
    methods:
        print_menu_Products():
        input_menu_choice():
        print_current_Products_in_list(list_of_rows):
        input_yes_no_choice(message):
        input_press_to_continue(optional_message=''):
        input_new_task_and_priority():

    changelog: (When,Who,What)
        CHenderson,12.4.2020,Added code to complete assignment 8
    """
```

Figure 8. The DocString of the Class IO

I further added code into this class that was previously written from assignment 6 to show the menu to the user, get the user's choice, show the current data from the file to the user, and add code to get product data from the user.

Main Script Body

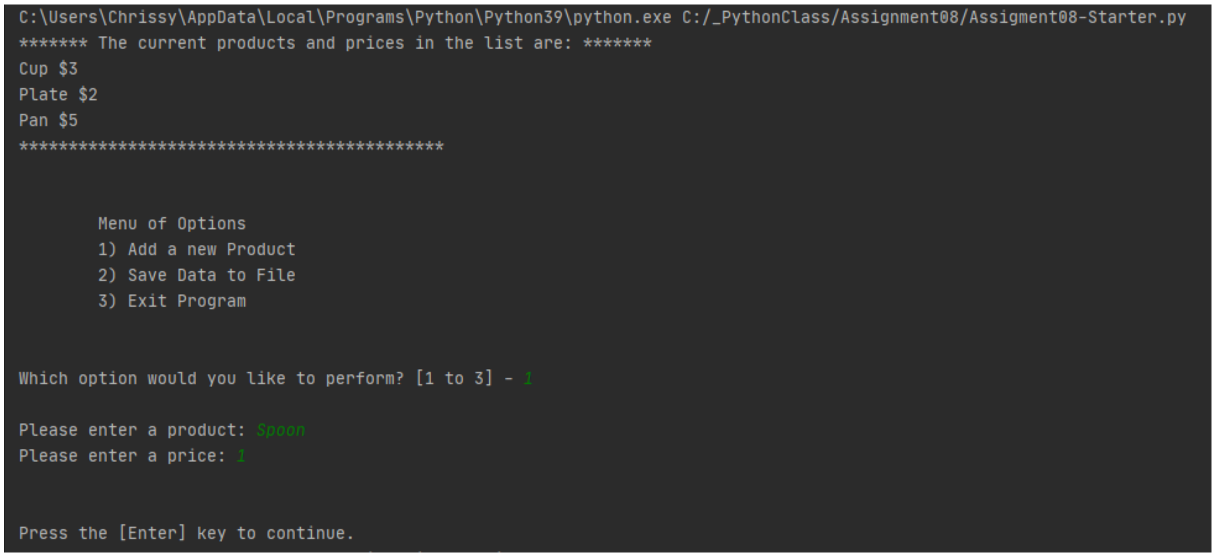
The main script of the body is where the program will run and pull in the classes and functions that have been created in the previous steps (Figure 9). Three different menu items are presented to the user after showing the current products in the file: adding a new product and price, saving the data to a file, and exiting the program.



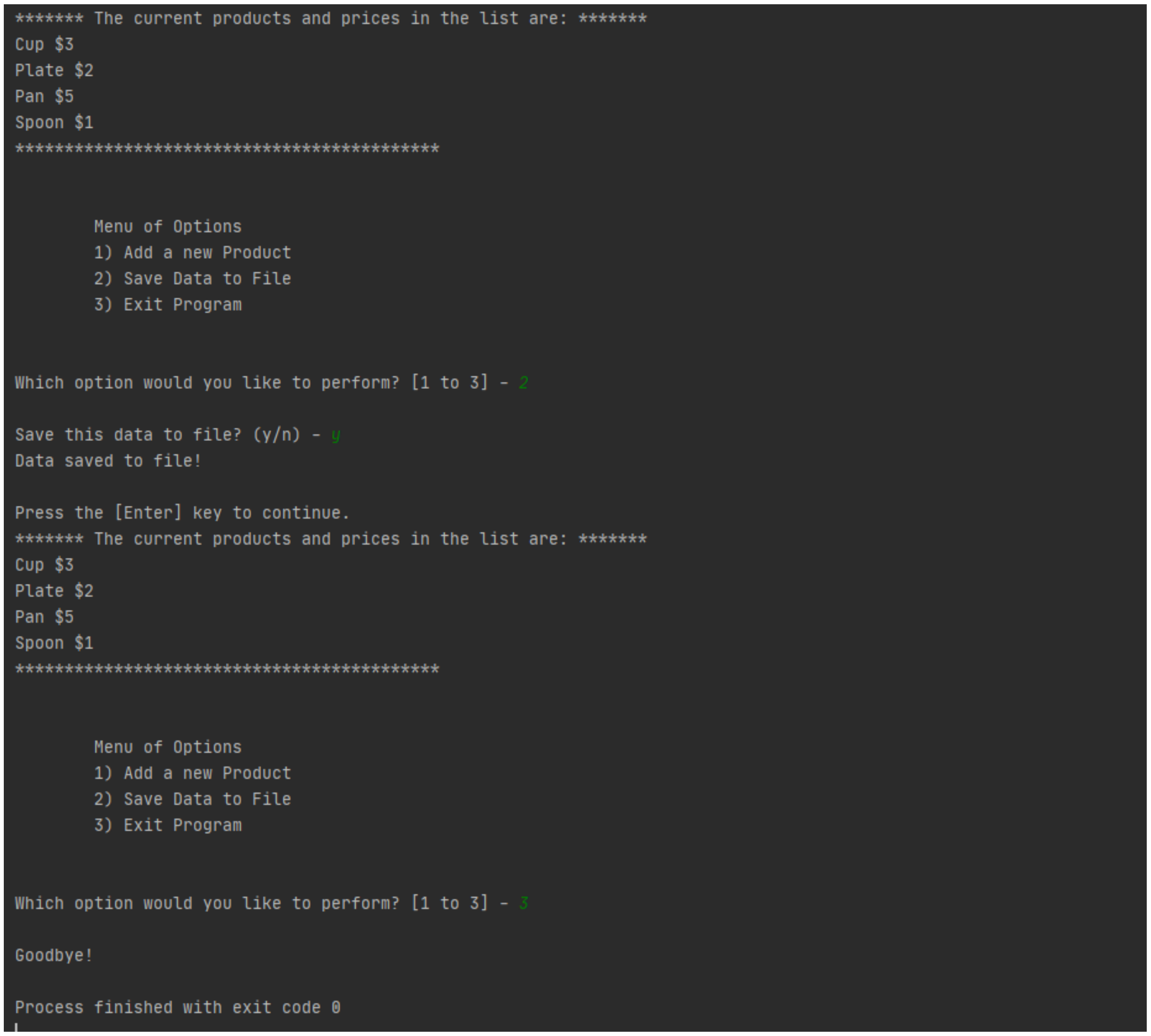
```python
# Step 1 - When the program starts, Load data from products.txt.
FileProcessor.read_data_from_file(file_name, lstOfProductObjects)  # read file data

# Step 2 - Display a menu of choices to the user
while (True):
    # Step 3 Show current data
    IO.print_current_Products_in_list(lstOfProductObjects)  # Show current data in the list/table
    IO.print_menu_Products()  # Shows menu
    strChoice = IO.input_menu_choice()  # Get menu option

    # Step 4 - Process user's menu choice
    if strChoice.strip() == '1':  # Add a new Product
        strProduct, strPrice = IO.input_new_task_and_priority()
        strProduct, strPrice = FileProcessor.add_data_to_list(strProduct, strPrice, lstOfProductObjects)
        IO.input_press_to_continue(strStatus)
        continue  # to show the menu

    elif strChoice == '2':  # Save Data to File
        strChoice = IO.input_yes_no_choice("Save this data to file? (y/n) - ")
        if strChoice.lower() == "y":
            FileProcessor.write_data_to_file(file_name, lstOfProductObjects)
            IO.input_press_to_continue(strStatus)
        else:
            IO.input_press_to_continue("Save Cancelled!")
        continue  # to show the menu

    elif strChoice == '3':  # Exit Program
        print("Goodbye!")
        break  # and Exit
```

Figure 9. The main body of the script that presents three menu choices to the user

The product list program was successfully executed in PyCharm IDE and in the Windows command line, respectively (Figure 10 and 11).

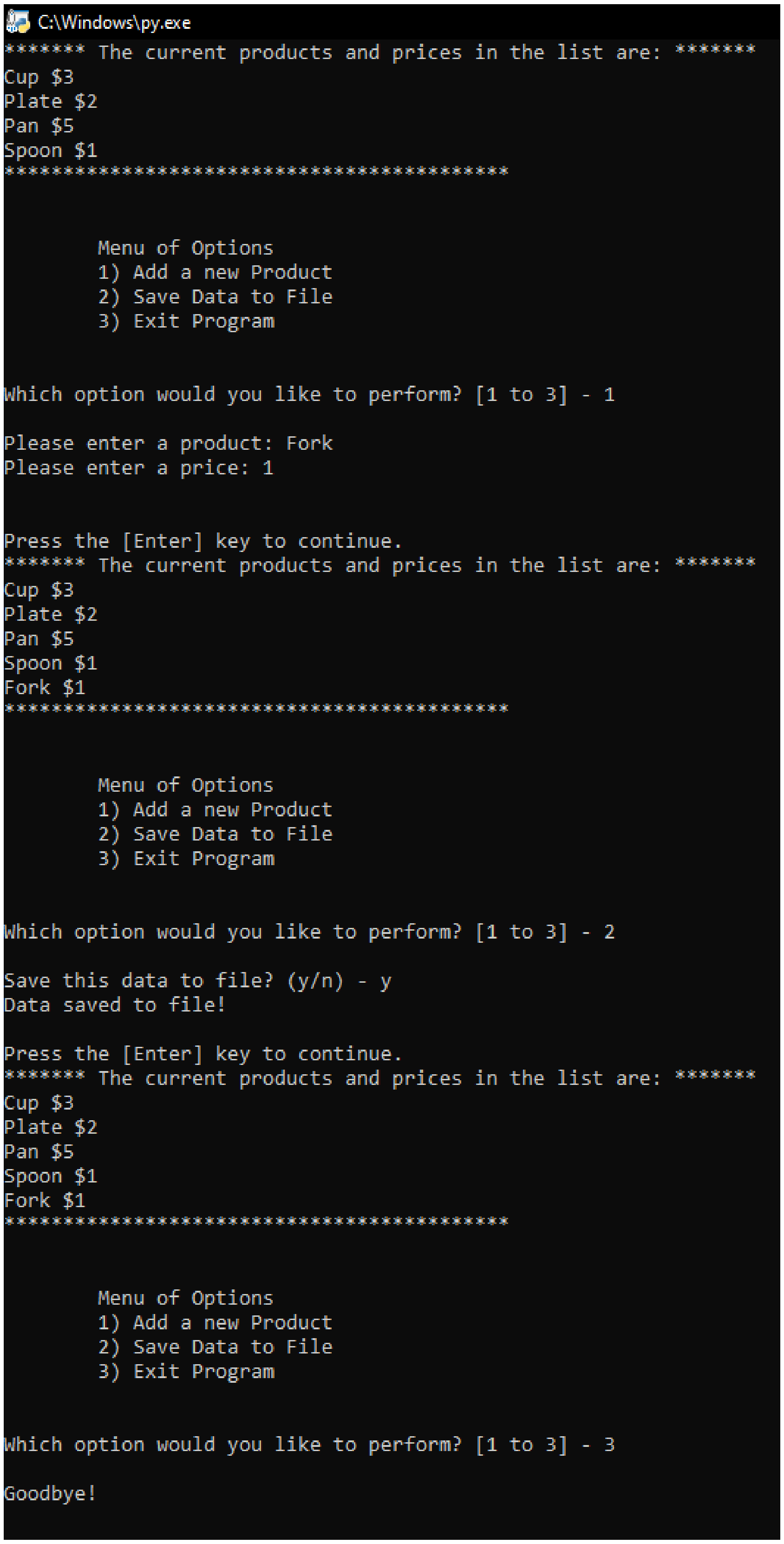


```
C:\Users\Chrissy\AppData\Local\Programs\Python\Python39\python.exe C:/_PythonClass/Assignment08/Assigment08-Starter.py
******* The current products and prices in the list are: *******
Cup $3
Plate $2
Pan $5
*******************************************


        Menu of Options
        1) Add a new Product
        2) Save Data to File
        3) Exit Program


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

Please enter a product: Spoon
Please enter a price: 1


Press the [Enter] key to continue.
```



```
******* The current products and prices in the list are: *******
Cup $3
Plate $2
Pan $5
Spoon $1
*******************************************


        Menu of Options
        1) Add a new Product
        2) Save Data to File
        3) Exit Program


Which option would you like to perform? [1 to 3] - 2

Save this data to file? (y/n) - y
Data saved to file!

Press the [Enter] key to continue.
******* The current products and prices in the list are: *******
Cup $3
Plate $2
Pan $5
Spoon $1
*******************************************


        Menu of Options
        1) Add a new Product
        2) Save Data to File
        3) Exit Program


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

Goodbye!

Process finished with exit code 0
```

Figure 10. The program working in PyCharm IDE



```
C:\Windows\py.exe
******* The current products and prices in the list are: *******
Cup $3
Plate $2
Pan $5
Spoon $1
*******************************************



        Menu of Options
        1) Add a new Product
        2) Save Data to File
        3) Exit Program


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

Please enter a product: Fork
Please enter a price: 1


Press the [Enter] key to continue.
******* The current products and prices in the list are: *******
Cup $3
Plate $2
Pan $5
Spoon $1
Fork $1
*******************************************



        Menu of Options
        1) Add a new Product
        2) Save Data to File
        3) Exit Program


Which option would you like to perform? [1 to 3] - 2

Save this data to file? (y/n) - y
Data saved to file!

Press the [Enter] key to continue.
******* The current products and prices in the list are: *******
Cup $3
Plate $2
Pan $5
Spoon $1
Fork $1
*******************************************



        Menu of Options
        1) Add a new Product
        2) Save Data to File
        3) Exit Program


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

Goodbye!
```

Figure 11. The program working in Windows command line

Summary

For this assignment, I was able to add in code to create two classes that contain functions to run a program. The program ultimately takes user input of a product and price, saves the data to a file, and exits the program. Classes are a great way to cleanly organize a group of functions that are similar in their task.