

Creating Custom Classes with Objects

Introduction

This assignment was by far the most difficult that I've tried to code to date! The concepts always seem to be rather logical and straight forward, but it always starts to get really tricky when it comes time to apply it to a program. The assignment already contained pseudo-coding to start. It was a matter of scripting the classes within each of the Separation of Concerns (Data, Processing and Presentation) and calling the classes coding within the Main program to complete the assignment.

Creating a Class with Custom Functions

A programmer can create a class in order to build objects within custom functions (better known as **methods** when used inside of a class). These objects or **instances**, are created so that they can have the same structure as defined by the class, but be unique unto themselves. Much like you can have two cats with two different names and personalities.

Scripting the Data

The data section of the SoC (Separation of Concerns) had a class name of Product. Here we construct our objects and then define their properties. The program consists of a Product and its Price contained as a row of data. First, they had to be initiated as objects. This is done in the **constructor** component of the class. Once initiated as their own unique selves (the keyword '**self**' is used when defining the objects within the code. Besides creating the objects, we also scripted that they should be saved as 'private' methods for internal processing only. This is done under the subheading of **attributes** of our constructor component. Usually, if there are no attributes defined under the constructor, you indicate them as a **field** above the constructor components.

Now that we have our objects, we'll want to define their **properties** so we can set the data and get the data. When setting the property, we pass in a value to the method. When getting the property, we add code to format the field or attributes data.

Scripting the Processing

For this Class, we used the **@staticmethod** to call up the methods(functions) to both add a new product with its price and to save them to the file. Because we'll be calling up these directives in the main program by the class name and not by the object name, we use the static method. When calling up an object, we need to define the 'self' parameter within the script of the method.

Scripting the Presentation

Here we are creating the inputs for the user and the outputs they'll be calling on. This class will be all static

methods except for one: Displaying the text file to the user. On this method, I saved some coding space and had the method open the file as an object and return that information to the user. In the main part of the program, the file is loaded and shown to the user and then used again as option 1 for the user to call upon if they add to the file and want to see for themselves.

Calling on the Classes and Objects

Now we can call on the classes and objects created to run our program. To call on a class, the programmer enters the class name followed by a period and then the name of the method(function) and then the valid variable values if necessary within the parentheses at the end. For the reading of the file class, a variable (f) is assigned to the class followed by parentheses. Then directly afterwards we use the print command and use the variable followed by a period, the method name and the valid variables that we pass inside the argument.

The following images illustrate the successful running of the program.

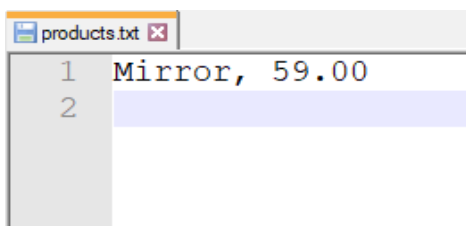


Figure 8.1: Text File that loads upon starting of program

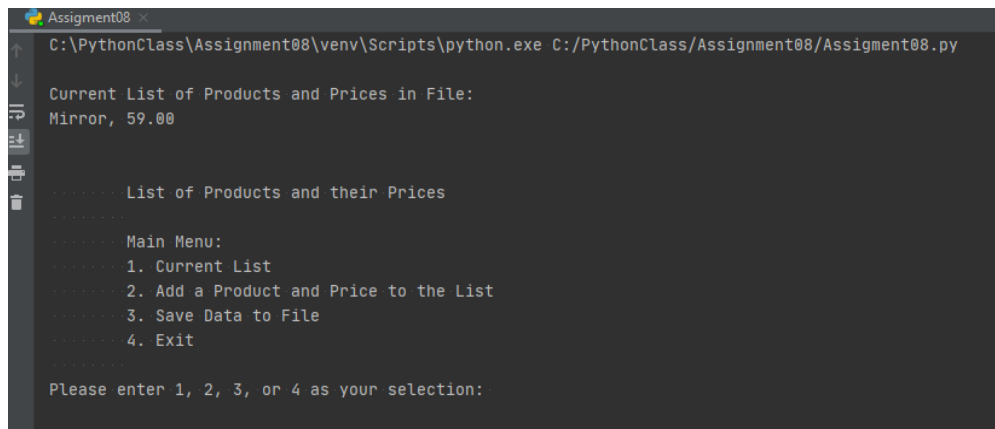


Figure 8.2: Program Load in PyCharm

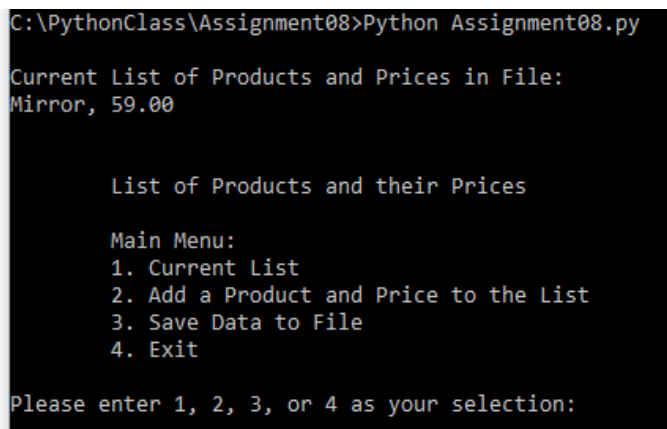


Figure 8.3: Program Load in Command Line Prompt Window

```
Please enter 1, 2, 3, or 4 as your selection: 2
2
Enter a Product: Table
What is the Price?: 99.00

----- List of Products and their Prices
-----
----- Main Menu:
----- 1. Current List
----- 2. Add a Product and Price to the List
----- 3. Save Data to File
----- 4. Exit
-----
Please enter 1, 2, 3, or 4 as your selection: 
```

Figure 8.4: Add Data in PyCharm

```
Please enter 1, 2, 3, or 4 as your selection: 2
2
Enter a Product: Table
What is the Price?: 99.00

----- List of Products and their Prices
-----
----- Main Menu:
----- 1. Current List
----- 2. Add a Product and Price to the List
----- 3. Save Data to File
----- 4. Exit
-----
Please enter 1, 2, 3, or 4 as your selection: 
```

Figure 8.5: Add Data in Command Line Prompt

```
Please enter 1, 2, 3, or 4 as your selection: 3
3
Data Saved!

----- List of Products and their Prices
-----
----- Main Menu:
----- 1. Current List
----- 2. Add a Product and Price to the List
----- 3. Save Data to File
----- 4. Exit
-----
Please enter 1, 2, 3, or 4 as your selection: 
```

Figure 8.6: Save Data in PyCharm

```

Please enter 1, 2, 3, or 4 as your selection: 3
3
Data Saved!

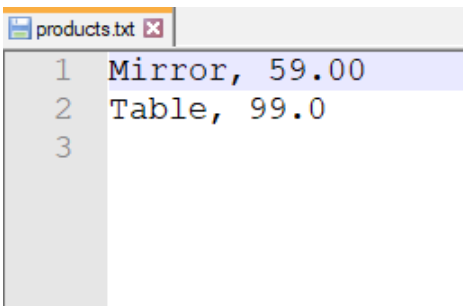
    List of Products and their Prices

    Main Menu:
    1. Current List
    2. Add a Product and Price to the List
    3. Save Data to File
    4. Exit

Please enter 1, 2, 3, or 4 as your selection:

```

Figure 8.7: Save Data in Command Line Prompt



```

products.txt
1 Mirror, 59.00
2 Table, 99.0
3

```

Figure 8.8: Updated Text File

```

Please enter 1, 2, 3, or 4 as your selection: 4
4

Process finished with exit code 0

```

Figure 8.9: Exit Program in PyCharm

```

Please enter 1, 2, 3, or 4 as your selection: 4
4

C:\PythonClass\Assignment08>

```

Figure 8.7: Exit Program in Command Line Prompt

Summary

It wouldn't be overstating things by saying this assignment "kicked my butt". But after many hours of trial and error, I finally arrived at a program that successfully ran using the Custom Classes and their objects that I scripted. Phew!