Nathaniel Smith

June 8, 2020

Foundations of Programming: Python (IT FDN 100 A Sp 20)

Assignment08

[https://github.com/10neg9/IntroToProg-Python-Mod08.git](https://github.com/10neg9/IntroToProg-Python-Mod08.git%20)

Python: Script Using Classes and Objects

# Introduction

In this paper I describe the code that I wrote for the Python script file Assignment08.py. The script uses classes to create objects and to call static methods. The purpose of this script is to create a list of class objects and store their attributes to a file. There are two attributes: a product name, and its price. In general, the script provides a user with a menu and depending on the user’s choice, it takes input from a user in the form of a product name and its price, adds that information to a list, prints that list when requested, and then saves that list to file when requested.

# Script Header

The script header in Figure 1 shows the title, description, and the changelog. There are other changelogs embedded in the docstrings of the classes. From the changelog you can see that I updated class Product, class FileProcessor, class IO, and main.



Figure 1. Assignment 08 Script Header

# Separation of Concerns

The script file is structured into 4 sections: data, processing, input/output, and main. The data section includes the variable declarations and the class Product. The processing section has a single class, FileProcessor. The input/output section also has a single class, IO. The main section calls the classes of the other sections to use their methods and to create objects from the Product class.

# Data – Variables

Figure 2 shows the variable declarations.



Figure 2. Variables

# Data – class Product Docstring

The data section has one class, Product. From the docstring shown in Figure 3, you can see that the Product class stores data about a product. It has two property methods: product\_name and product\_price. The \_\_init\_\_ method is used to initialize an object instance of the Product class when the object is created.



Figure 3. class Product docstring

# Data – class Product Constructor

Constructor is another name given to \_\_init\_\_ method. This method has two parameters: a string and a float. This is clearly shown in Figure 4. The constructor is executed when the Product class is called to create an object instance. It calls the property methods to create an object with two attributes: product\_name and product\_price.



Figure 4. Constructor Method of class Product

# Data – class Product Properties

In Figure 5 and Figure 6 you get a more detail look at the properties of Product. Both properties have a getter (indicated by the @ property decorator) and a setter (indicated by the @*method\_name*.setter decorator).

The product\_name getter returns the product’s name as a string. The product\_name setter, takes the value passed to it and assigns it to a hidden attribute, \_\_product\_name, in title case (note the two underscores preceding the attribute name).



Figure 5. product\_name Getter and Setter Properties

The product\_price getter returns the product’s price as a float value. The product\_price setter, takes the value passed to it and assigns it to a hidden attribute, \_\_product\_price.

This marks the end of the data section.



Figure 6. product\_price Getter and Setter Properties

# Processing – class FileProcessor Docstring

The processing section contains a single class, FileProcessor. As shown in the docstring in Figure 7, FileProcessor saves data to a file with the save\_data\_to\_file() method and copies data from a file into a list with the read\_data\_from\_file() method. Each item it reads from the file is stored as an object in a list.



Figure 7. FileProcessor Docstring

# Processing – Read Data from File

Figure 8 shows the code for reading data from file. The code takes the name of a file as an argument. It tries to open the file, and if the file exists it will open it in read mode. Otherwise, it will provide feedback to the user the file does not exist yet and the program is starting with an empty list. If the file does exists, each line of the file is iterated through, the lines are split into two variables, and these variables are used to create objects from the Product class. Each object is appended to a list. The file is closed after reading through all the lines and the list of objects is returned.



Figure 8. read\_data\_from\_file() Method

# Processing – Save Data to File

If the user wants to save the product list to file, the code shown in Figure 9 will be executed. This method takes a filename and a list name as arguments. It opens the file in write mode, and for each element in the list, it writes a new line to the file that includes the product’s name followed by a comma followed by the product’s price. The file is closed after writing is complete.

This marks the end of the processing section.



Figure 9. save\_data\_to\_file() Method

# Presentation – class IO Docstring

The presentation section includes a single class, IO. The docstring for IO is shown in Figure 10. IO contains 5 static methods.



Figure 10. class IO Docstring

# Presentation – Display Menu

Figure 11 shows the print\_menu\_Tasks() method. It is used to display a menu of 4 choices to the user.



Figure 11. Display Menu Method

# Presentation – Get User’s Menu Choice

Figure 12 shows the input\_menu\_choice() method. This method gets the menu choice from the user, strips it of whitespace, stores its value in the variable choice, and returns choice.



Figure 12. Menu Choice Method

# Presentation – Print the Product List

Figure 13 shows the print\_current\_product\_list() method. This takes a single argument of list type. The list contains the objects created from the Product class. A header is printed and below it each product and price, and below that a footer separator is printed. Note that the print statement is calling the Product getter properties to get the product name and product price of each object.



Figure 13. Method to Print Product List

# Presentation – Request Product and Price from User

Figure 14 shows the code for the input\_product\_and\_price() method. This method asks the user for a product name and checks if the name is valid. Then it asks for a price and checks if the price is valid. If it discovers invalid input exceptions are raised to provide feedback to the user that an invalid entry was made. If everything is on the up-and-up, the method returns a tuple containing the product’s name and price.



Figure 14. Method to Get Product and Price Input from User

# Presentation – Pause Program

The code in Figure 15 is used to pause the program at certain locations. It has one optional parameter, and if an argument is passed to this parameter, an optional message will be printed to screen. Otherwise, it prints a message asking the user to press the enter key to continue.



Figure 15. Method to Pause Program

# Main – Startup

When the script starts up, it checks for the file, reads the file if it exists, copies the data from the file (again only if the file already exists) to a list, prints the menu, and asks the user to make a choice from the four menu options. The code is shown in Figure 16.



Figure 16. Main Body Code Executed at Program Start

# Main – Menu Option 1 Chosen

The user’s menu option is checked against the 4 possible menu options using if-elif statements. If the user entered ‘1’ the product list will be printed, see Figure 17.



Figure 17. Print Current Product List

# Main – Menu Option 2 Chosen

If the user chooses option 2, the user will be asked to provide a product name and price. The inputs will be checked for validity, and if valid, an object will be created from the Product class. The object is appended to the list containing current list of products. See Figure 18.

If the user does not provide valid input for either the product name or price, friendly messages will be provided. These messages are handled through the try-except statement.



Figure 18. Add a Product to the List

# Main – Menu Option 3 Chosen

If the user chooses option 3, the list of products will be saved to a file. This code is shown in Figure 19.



Figure 19. Save Product List to File

# Main – Menu Option 4 Chosen

Figure 20 shows the code to exit the program if the user chooses option 4.



Figure 20. End the Program

# Main – Menu Option Invalid

The last statement in the script, shown in Figure 21, executes if the user did not choose a number from 1 to 4. A friendly message is displayed and the user has to press enter to continue at which point the menu is printed again and the user is asked to choose an option from the menu.



Figure 21. Give User Feedback About Invalid Selection

# Testing Script in PyCharm

I first tested the code in PyCharm. The startup is shown in Figure 22.

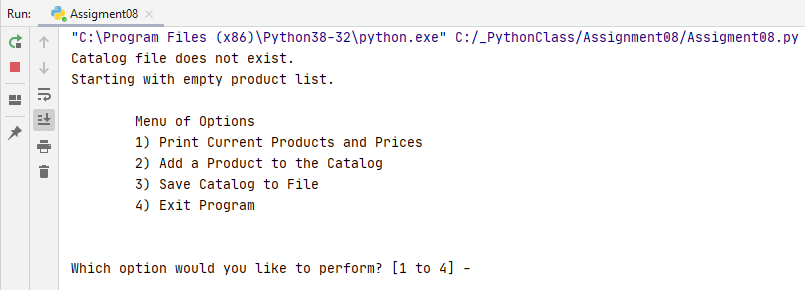


Figure 22. Assignment08.py Startup in PyCharm

Because the text file did not yet exist, the list started empty. I added an item to the list as shown in Figure 23.

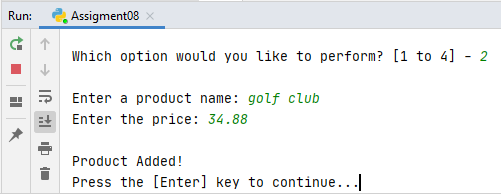


Figure 23. Added a Product

Then I printed the list with my newly added item, see Figure 24.

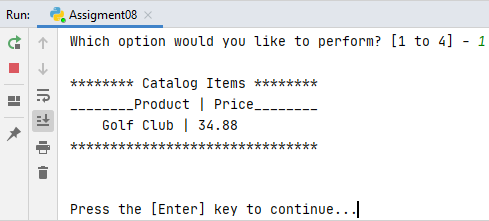


Figure 24. Print Product List

The list was saved to file, see Figure 25.

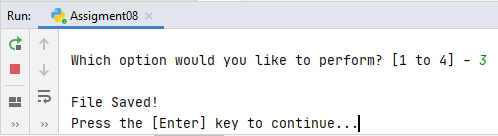


Figure 25. File Saved

I made an invalid menu selection, see Figure 26.

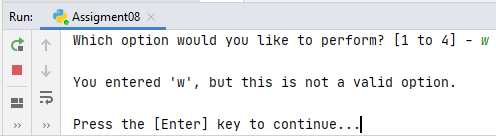


Figure 26. Invalid Selection

Time to exit the program, see Figure 27.

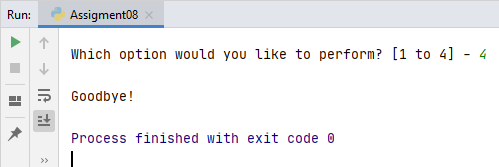


Figure 27. Exit Program

And finally I checked that the file was created with my product list as shown in Figure 28.

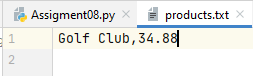


Figure 28. products.txt

# Testing Script in Command Prompt

After testing in PyCharm, I tested the code from Windows Command Prompt. The startup screen is shown in Figure 29.

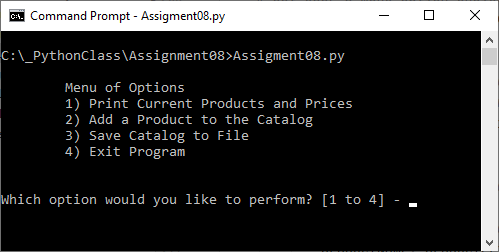


Figure 29. Assignment08.py Startup in Command Prompt

I added an item with option 2, see Figure 30.

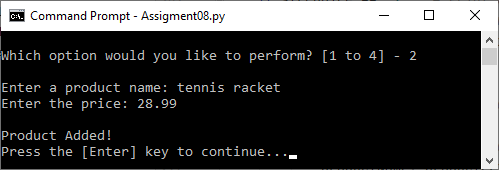


Figure 30. Added a Product

I saved the new list to file with option 3, see Figure 31.

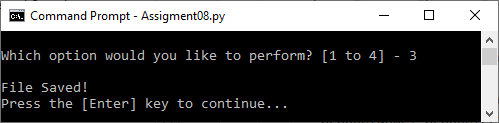


Figure 31. File Saved

I tested an invalid option, see Figure 32.

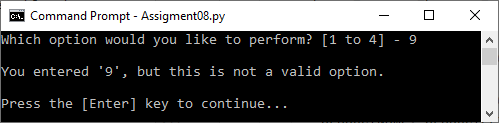


Figure 32. Invalid Selection

I printed the product list with option 1, see Figure 33.

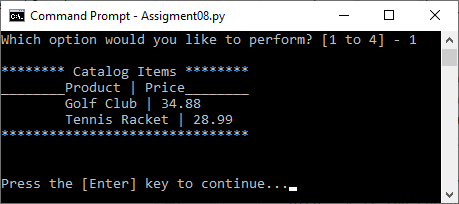


Figure 33. Print Product List

Time to exit the program, see Figure 34.

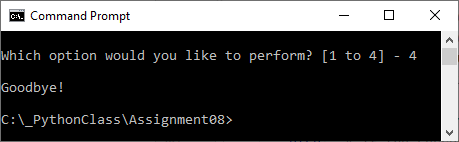


Figure 34. Exit Program

And finally, I checked that the code worked by opening the text file in Notepad. See Figure 35.

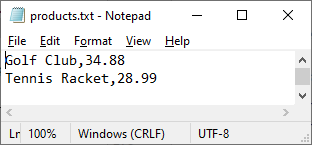


Figure 35. product.txt Text File

# Summary

In summary, I successfully wrote a program that creates object instances from classes. The objects were stored in a list until it was time to save the object attributes to a file. And when the file was opened, the file’s contents were used to create objects that were stored to a list. This code could be improved by adding an option to remove objects from the list.