**Theresa Ward**

December 8, 2020

IT FDN 110 A

Assignment 08

# https://github.com/ScrappySnacks/IntroToProg-Python-Mod08

## **Menu-driven script with three classes**

## Introduction

For assignment #8, we are asked to complete a menu-driven script with three classes, Product, FileProcessor and IO, that also includes error handling. This document will describe the steps taken to complete the script and ensure it works properly. As with past assignments PyCharm is used here for script development. Assignment documents can be found in the “ScrappySnacks” Mod08 repository: <https://github.com/ScrappySnacks/IntroToProg-Python-Mod08>

## Script Development

The intent of the script is to first load data from file into a list of product objects and display a menu to the user that can execute the following tasks: 1) Add a product object, 2) show data in the current list of product objects, 3) save the list of product objects to file and 4) exit the script. In addition, the script handles multiple exceptions. Separation of concerns and the standard class pattern are used to structure the script. Additionally, each class begins with a DocString that includes additional notes and description of each class.

## Product Class

This class organizes data for each product. The class defines a constructor and two properties, a setter and getter, for both the product\_name and product\_price attributes. The product\_name getter property converts the name to string data type and title case. The product\_price getter property is converted to float data type. The setter properties contain a “test” to ensure that a string and number is assigned to product\_name and product\_price, respectively. If the values do not pass the data type test, an exception is raised. An example of the code is shown for product\_name in Figure 1.

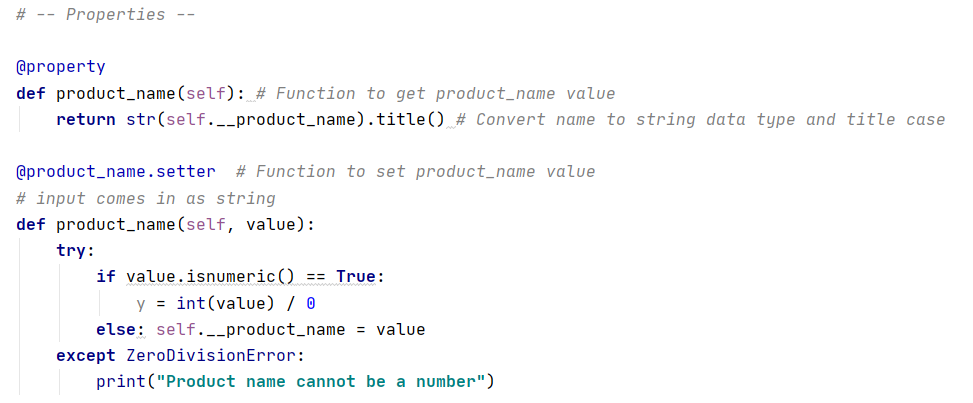


Figure 1. Example of properties for product\_name. This script triggers an exception if the input is a number.

## FileProcessor Class

The FileProcessor Class reads data from file and stores data in a list of product objects. It also saves a list of product objects to a file that the user specifies. FileProcessor includes two static methods to accomplish this: read\_data\_from\_file() and save\_data\_to\_file(). The read\_data\_from\_file() method loops through the file contents, converts each row of data into a Product object, and appends this row to the lstOfProductObjects list object.

## IO Class

This class contains several static methods. The first is talk(), which prints the contents of one Product object. The second method is user\_input(). This was a challenging method to write, as I needed to apply the getter property to the user input. I opted to create an object instance, which is populated with placeholder values, “null” and 0. As the user provides data, the getter property will raise exceptions if the input is not as expected (string and numeric data types). Once the new object is defined, it is appended to the lstOfProductObjects list object. Figure 2 shows the script written to handle user input. It is noted that, if an exception is raised during input, the default values are captured in lstOfProductObjects. An improvement on this script would be to add functionality to delete a product object.

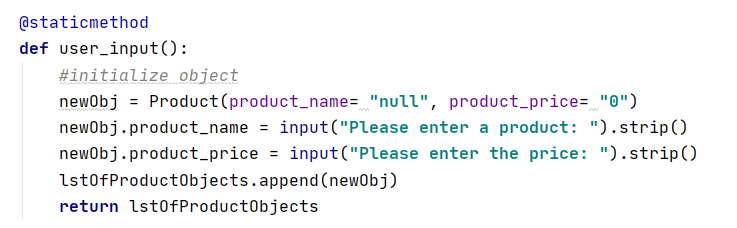


Figure 2. Static method for capturing user input. Default values are used to create an object instance.

The next method, user\_choice(), solicits the menu choice. And lastly, print\_menu\_options(), prints the list of menu options.

## Main Script Body

The main body of the script contains two *while* loops set to Boolean True. The first *while* loop sets up a try / except block to capture user input for file location. If the file is not found, an exception is raised.

The second *while* loop also contains a try / except block. Within the try / except block, the user is requested to choose an option to add a new product and price, show current data, save data to file or exit the program. If the user enters a string instead of a number, then an exception is raised via the ValueError class. Further, if the user enters a value other than 1 – 4, a warning is printed.

If the user enters “1,” the script prompts the user for input. If the user enters “2,” the script will call the talk() method and print the current contents of lstOfProductObjects. If the user enters “3,” the script will ask for the file name for saving content and then calls the save\_data\_to\_file() within the FileProcessor class. If the user enters “4,” the program ends. Figure 3 shows an example of a “happy” script without any exceptions raised. Figure 4 shows the output when saved. Figure 5 shows an example of output with exceptions raised. The script does run successfully from the command window as shown in Figure 6.

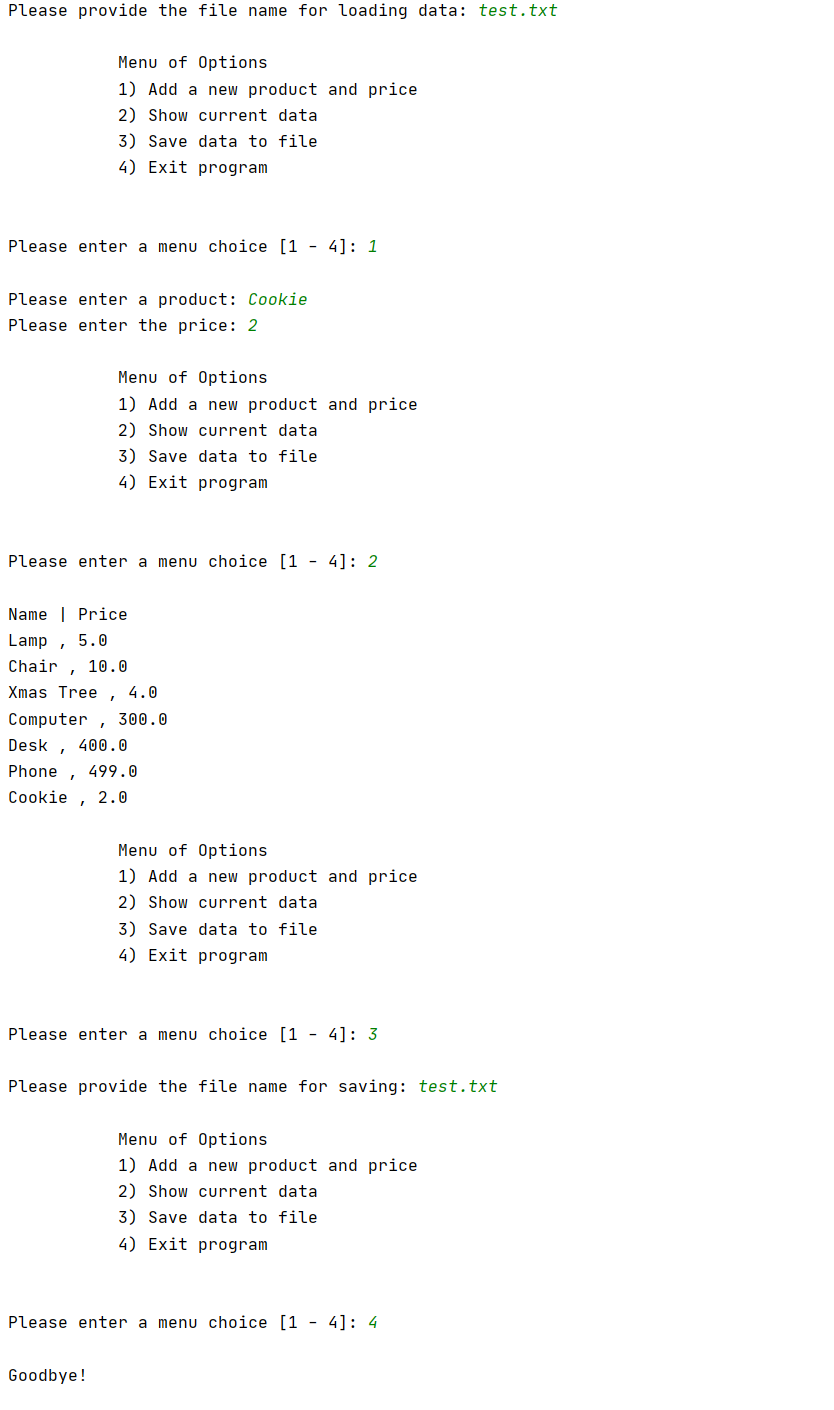


Figure 3. PyCharm output without exceptions raised

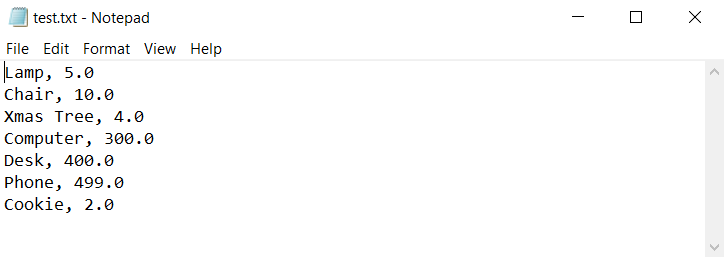


Figure 4. Saved data to file

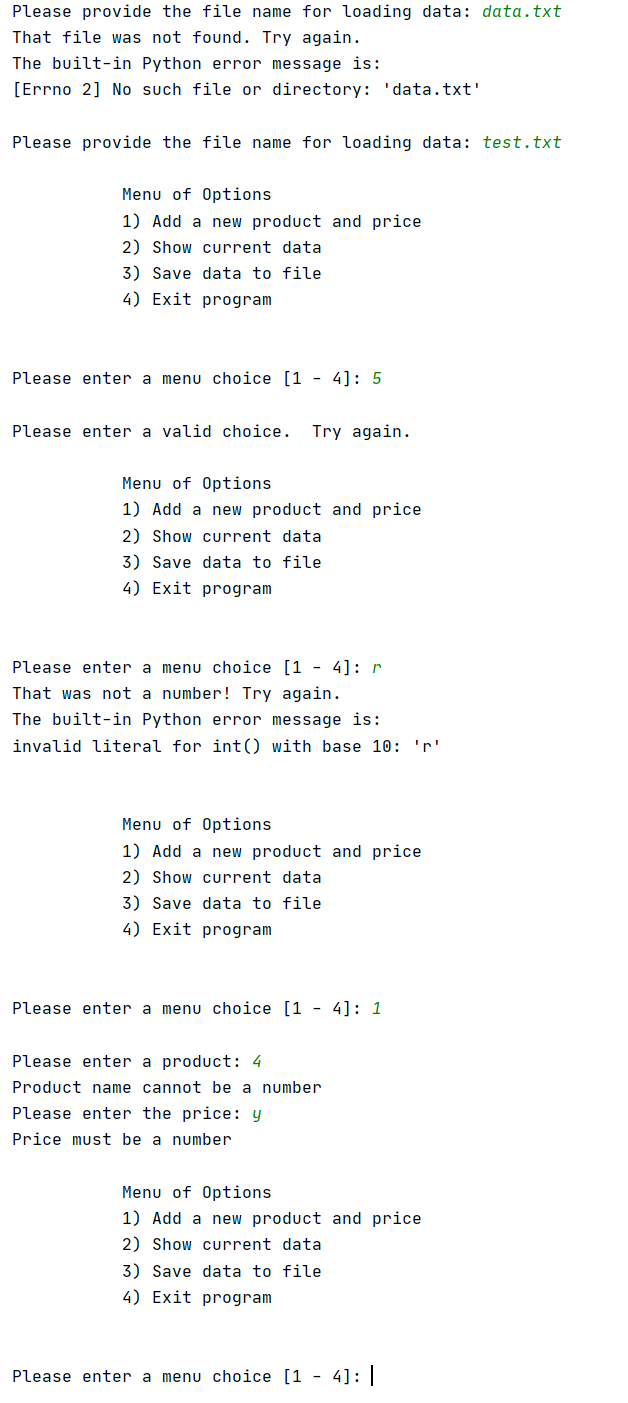


Figure 5. PyCharm output with exceptions raised

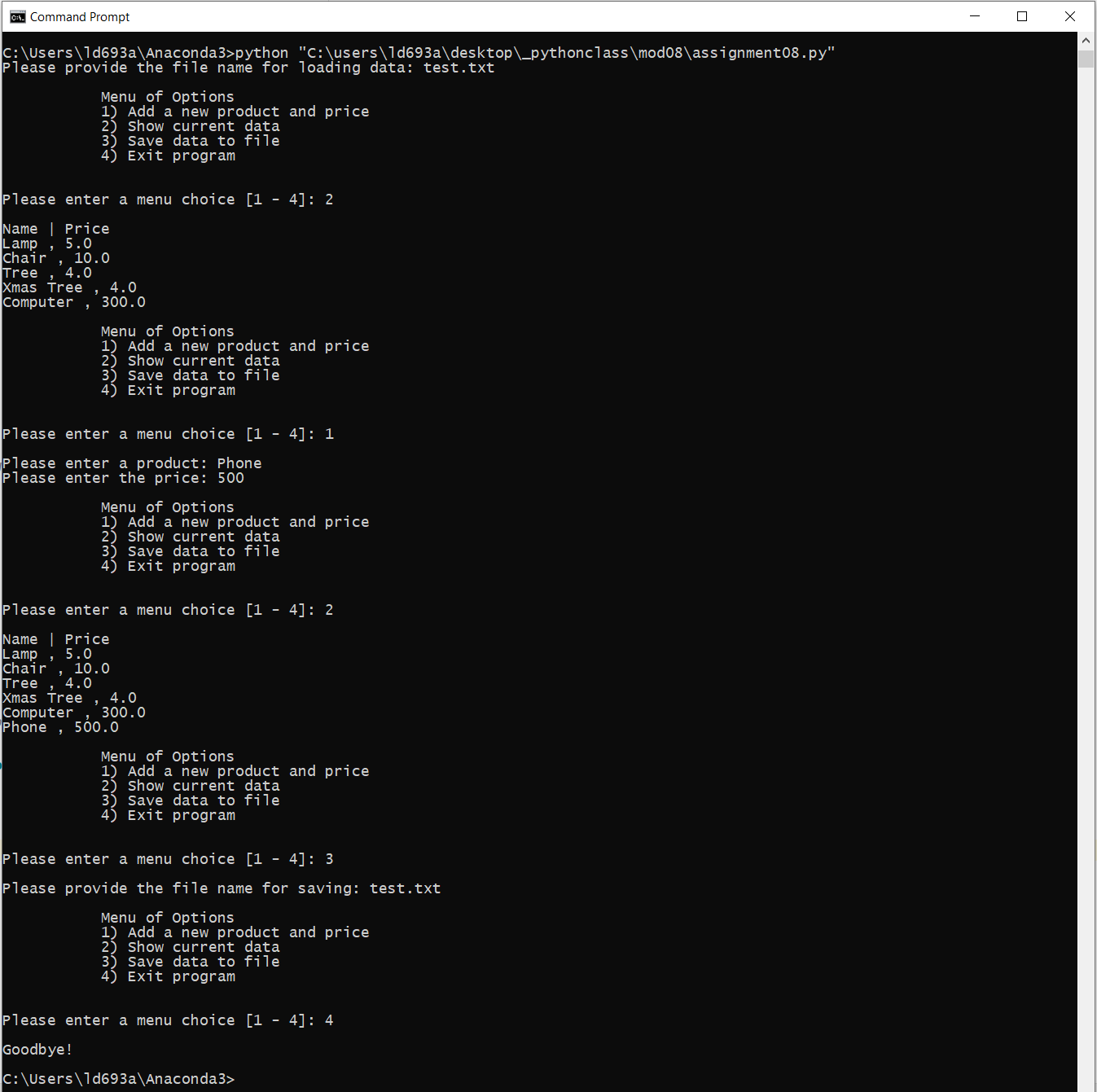


Figure 6. Output via command window

## Summary

In summary, I was successful in completing a script that meets the requirements of Module 08. This assignment was especially challenging. The ability to write efficient classes and handle exceptions will take practice.