Kate Beverly

August 31, 2022

Foundations of Programming: Python

Assignment 08

https://github.com/kbev12/IntroToProg-Python-Mod08

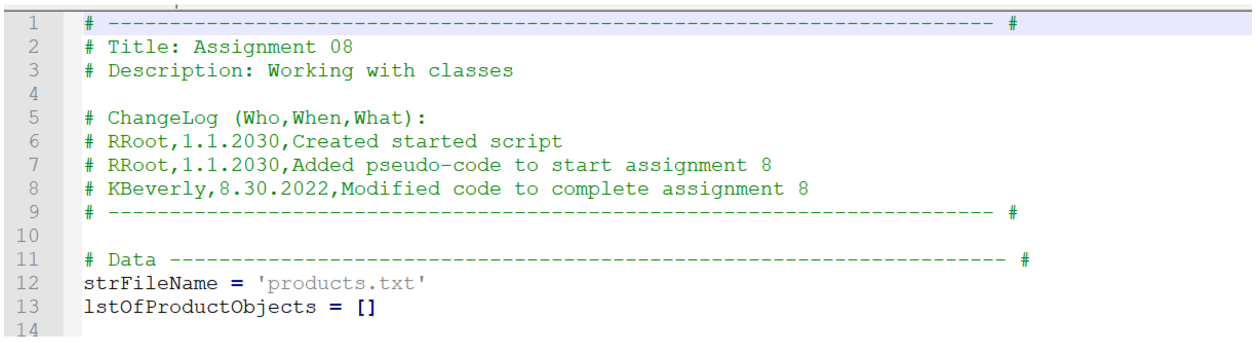
Product List Script

Introduction

For assignment 08 for the class Foundations of Programming: Python I worked on a python script to manage a product list. The script reads from a file to memory and then gives the user the option to display the information, add a product, write to the file, and end the program.

Creating the Script

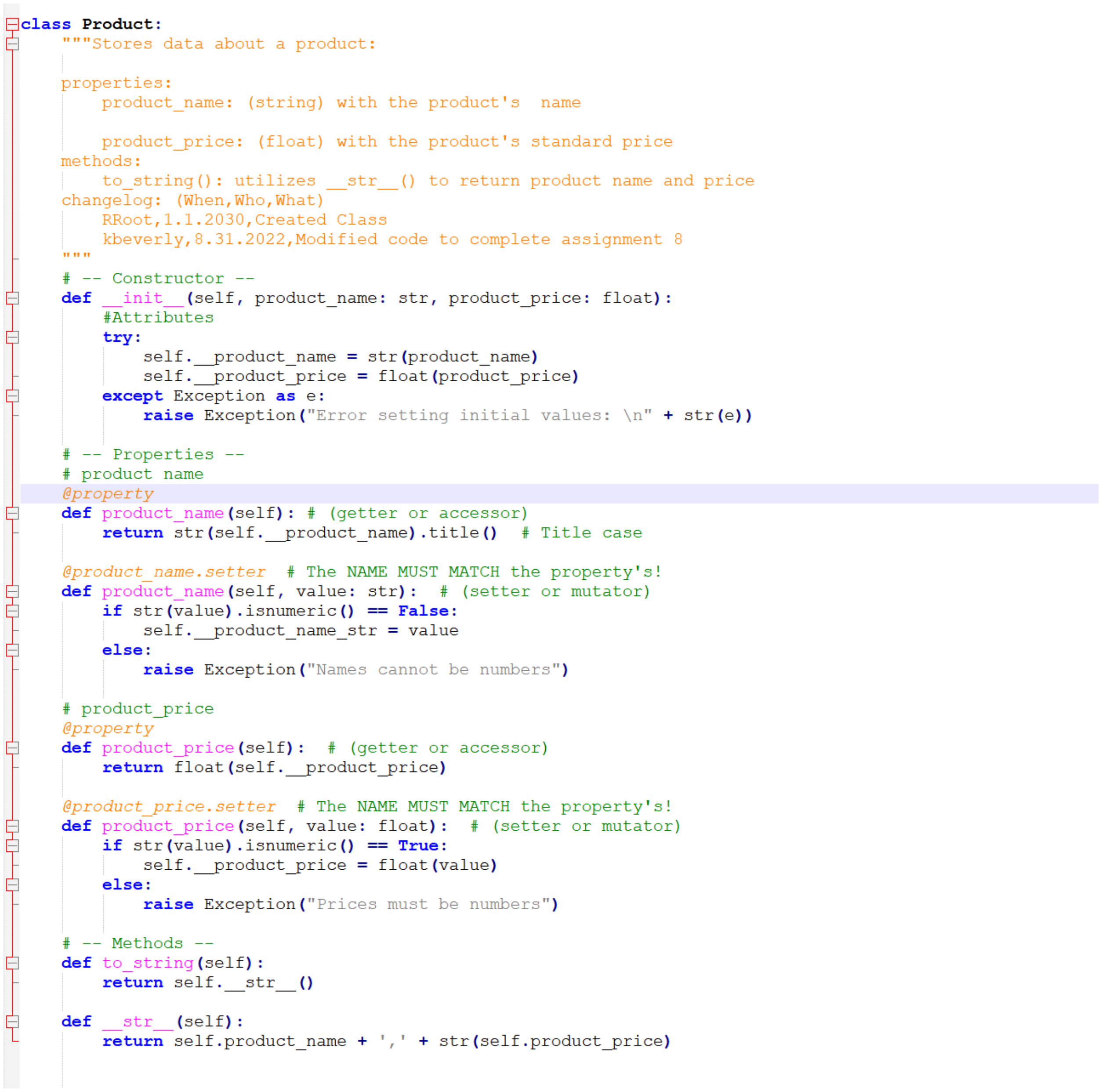
The script begins with declaring the global variables (**Figure 1**).



```
# ------------------------------------------------------------------------ #
# 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
# KBeverly,8.30.2022,Modified code to complete assignment 8
# ------------------------------------------------------------------------ #

# Data -------------------------------------------------------------------- #
strFileName = 'products.txt'
lstOfProductObjects = []

```

Figure 1. Declaring the program variables

The script enters the first class named Product (Figure 2). The Product class stores the data about a product. The first function is the constructor to set the initial values for product_name and product_price. The getters and the setters are created for product_name and product_price. The setters for each have the formatting code. The respective setters are validating the data inputted to confirm it is the correct data type. Property_name checks is the data type entered is numeric and if so, it moves onto the else statement in the and raises an exception for the error handling. If it was not numeric it sets the input as the value. The setter for property_price has similar error handling but wants the data type to be numeric.



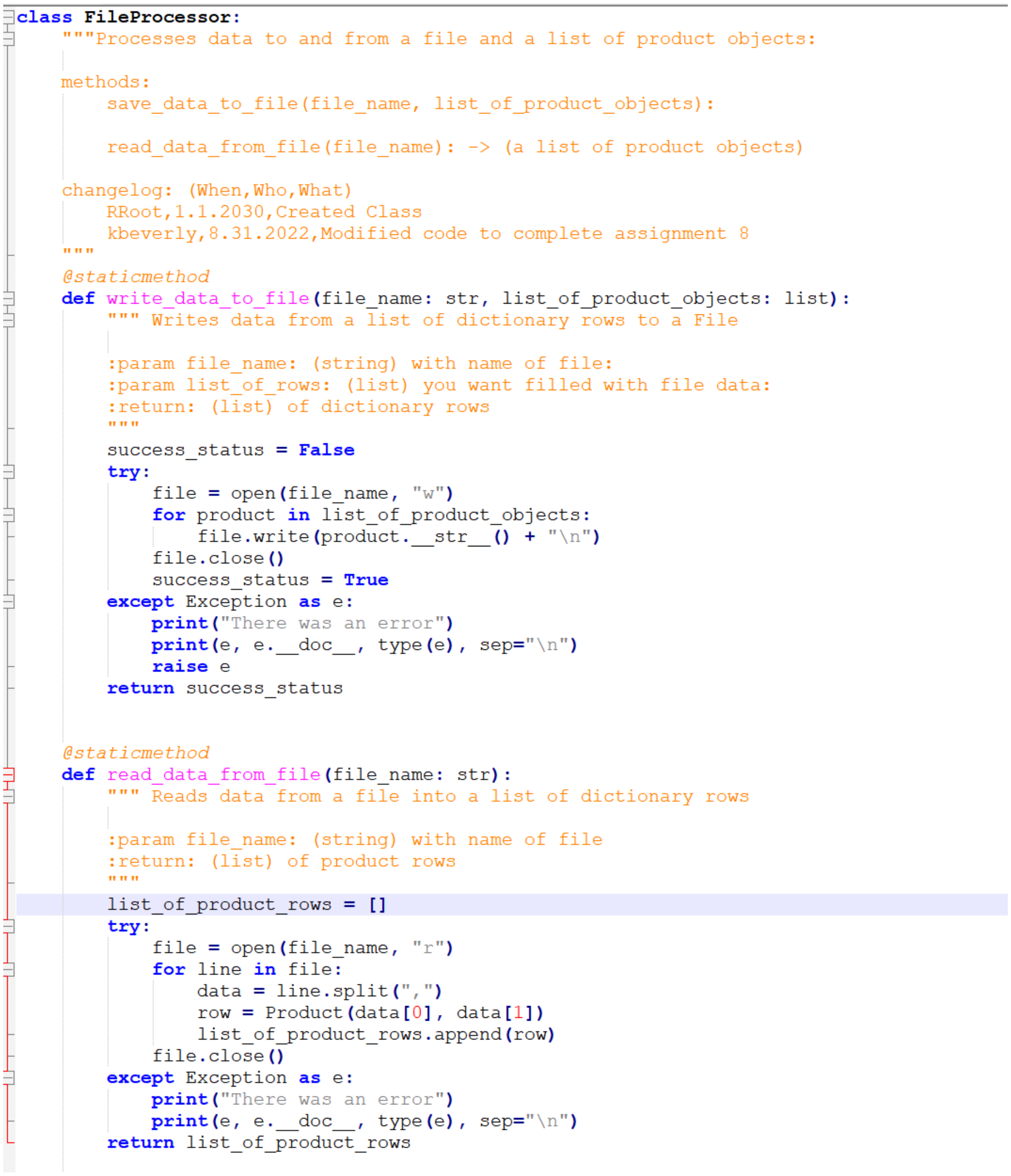
```python
class Product:
    """Stores data about a product:

    properties:
        product_name: (string) with the product's  name

        product_price: (float) with the product's standard price
    methods:
        to_string(): utilizes __str__() to return product name and price
    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        kbeverly,8.31.2022,Modified code to complete assignment 8
    """
    # -- Constructor --
    def __init__(self, product_name: str, product_price: float):
        #Attributes
        try:
            self.__product_name = str(product_name)
            self.__product_price = float(product_price)
        except Exception as e:
            raise Exception("Error setting initial values: \n" + str(e))

    # -- Properties --
    # product name
    @property
    def product_name(self): # (getter or accessor)
        return str(self.__product_name).title()  # Title case

    @product_name.setter  # The NAME MUST MATCH the property's!
    def product_name(self, value: str):  # (setter or mutator)
        if str(value).isnumeric() == False:
            self.__product_name_str = value
        else:
            raise Exception("Names cannot be numbers")

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

    @product_price.setter  # The NAME MUST MATCH the property's!
    def product_price(self, value: float):  # (setter or mutator)
        if str(value).isnumeric() == True:
            self.__product_price = float(value)
        else:
            raise Exception("Prices must be numbers")

    # -- Methods --
    def to_string(self):
        return self.__str__()

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

Figure 2 Product class. Stores data about a product

The next class FileProcessor contains the functions to process data to and from a file and the list of property data in memory. (Figure 3)

The first function write_data_to_file takes in a variable that represents a file name, which is a string, and a list of product objects, which is a list. It initializes the variable *success_status* and sets it to False. It uses a try/catch to open the *file_name* (which was passed as a variable) in write mode. It loops through the *list_of_product_objects* (also passed in) and writes each row to the *file_name* file. If this was successful *success_status* is now set to True and the updated *success_status* is returned. If it was not successful it would hit the exception and throw an error.

The second function in the FileProcessor class read_data_from_file takes in the string file_name. It creates an empty list called list_of_product_rows. It enters a try/catch where, if successful, opens file_name in read mode and for every line in the file splits the line at the comma and appends each row to the previously empty list list_of_product_rows and that list is returned. If it was not successful it would hit the exception and throw an error.



```python
class FileProcessor:
    """Processes data to and from a file and a list of product objects:

    methods:
        save_data_to_file(file_name, list_of_product_objects):

        read_data_from_file(file_name): -> (a list of product objects)

    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        kbeverly,8.31.2022,Modified code to complete assignment 8
    """
    @staticmethod
    def write_data_to_file(file_name: str, list_of_product_objects: list):
        """ Writes data from a list of dictionary rows to a File

        :param file_name: (string) with name of file:
        :param list_of_rows: (list) you want filled with file data:
        :return: (list) of dictionary rows
        """
        success_status = False
        try:
            file = open(file_name, "w")
            for product in list_of_product_objects:
                file.write(product.__str__() + "\n")
            file.close()
            success_status = True
        except Exception as e:
            print("There was an error")
            print(e, e.__doc__, type(e), sep="\n")
            raise e
        return success_status


    @staticmethod
    def read_data_from_file(file_name: str):
        """ Reads data from a file into a list of dictionary rows

        :param file_name: (string) with name of file
        :return: (list) of product rows
        """
        list_of_product_rows = []
        try:
            file = open(file_name, "r")
            for line in file:
                data = line.split(",")
                row = Product(data[0], data[1])
                list_of_product_rows.append(row)
            file.close()
        except Exception as e:
            print("There was an error")
            print(e, e.__doc__, type(e), sep="\n")
        return list_of_product_rows
```

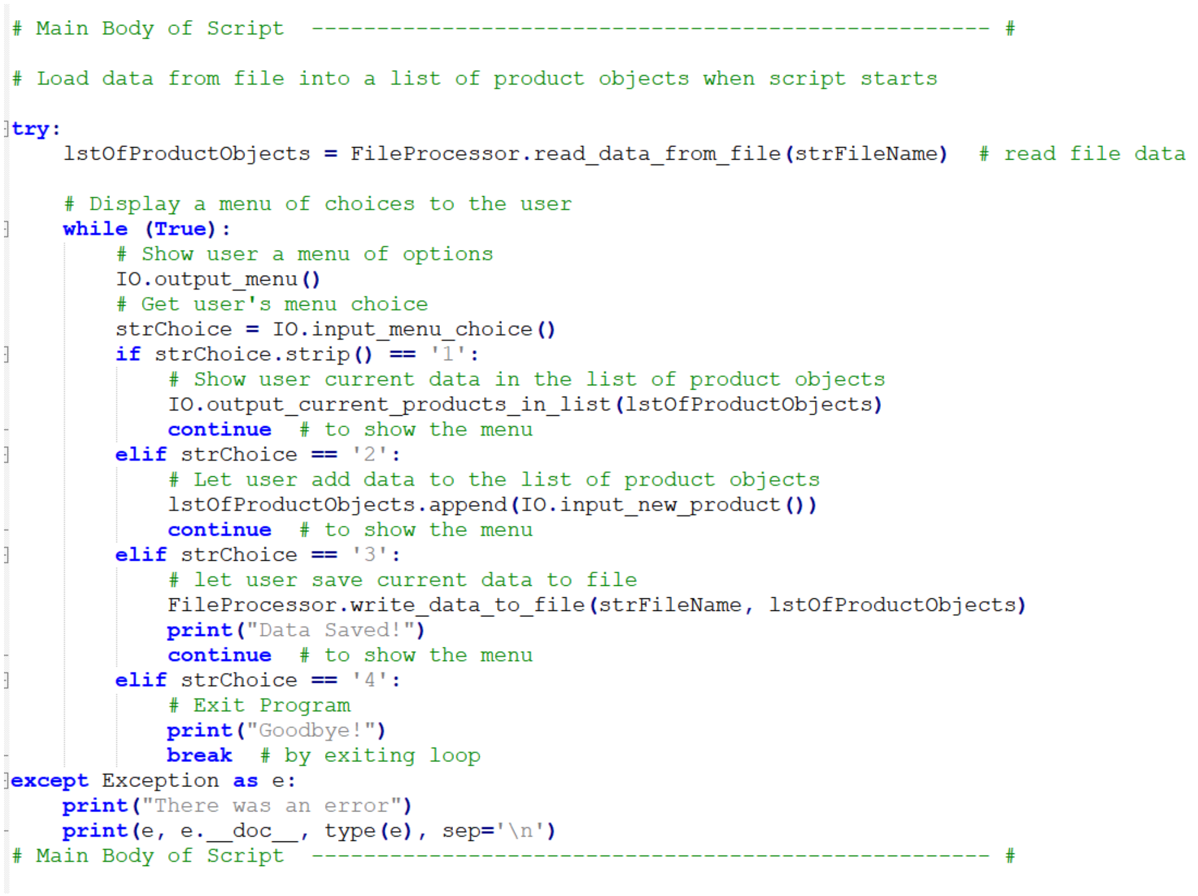
Figure 3 Class FileProcessor. Processes data to and from a file and a list of products

The last class is class IO which contains the input and output functions. (Figure 4) The first function output_menu_items displays a menu of choices to the user using a print statement. The next function input_menu_choice requests input from the user on which menu choice they would like and assigns it to the variable *choice* and returns that variable. The third function, output_current_products_in_list takes in the list_of_rows and prints each of the rows. The last function input_new_product requests the user to input a product and assigns it to the variable product. It then requests the user to input the product's price and assigns it to the variable price. A Product object (reference to the Product class) is instantiated where the variable *product* is assigned to product_name and *price* is assigned to product_price and set as the variable *p*. The variable *p* is returned. If any errors occur, it hits the exception and throws an error.

```python
# Presentation (Input/Output)  -------------------------------------------- #
class IO:
    """ A class for performing Input and Output
     methods:
     print_menu_items():
     print_current_list_items(list_of_rows):
     input_product_data():
     changelog: (When,Who,What)
     RRoot,1.1.2030,Created Class
    kbeverly, 8.31.2022, added functions"""

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

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

    @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

    @staticmethod
    def output_current_products_in_list(list_of_rows: list):
        """ 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 are: *******")
        for row in list_of_rows:
            print(row.product_name + " (" + str(row.product_price) + ")")
        print("*******************************************")
        print()  # Add an extra line for looks

    @staticmethod
    def input_new_product():
        """  Gets product and product price to be added to the list

        :return: object with input data
        """
        try:
            product = str(input("What is the product? ")).strip()
            price = float(input("What is the product's price? "))
            print()
            p = Product(product_name=product, product_price=price)
        except Exception as e:
            print(e)
        return p
# Presentation (Input/Output)  -------------------------------------------- #
```

Figure 4. Class IO. Performs the input and output functions

Lastly, we see the control flow of the script, the main body. (Figure 5) It runs as a try/catch where if successful it loads the data from the file into a list of product objects right when the script starts. It calls the output_menu function from the IO class to show the user the menu options. It then calls IO.input_menu_choice to get the user's menu choice and assigns it to the variable *strChoice*. If the choice is 1 it calls IO.output_current_products_in_list to show the current products in the list. If the choice is 2 it calls IO. input_new_product to append the data to the list and add a product. IF 3 is selected FileProcessor.write_data_to_file is called to save the current data to a file. If 4, then the program ends. If any available option but 4 is selected after the function runs it continues to show the menu. If the initial run is not successful, the exception is run and an error is displayed.

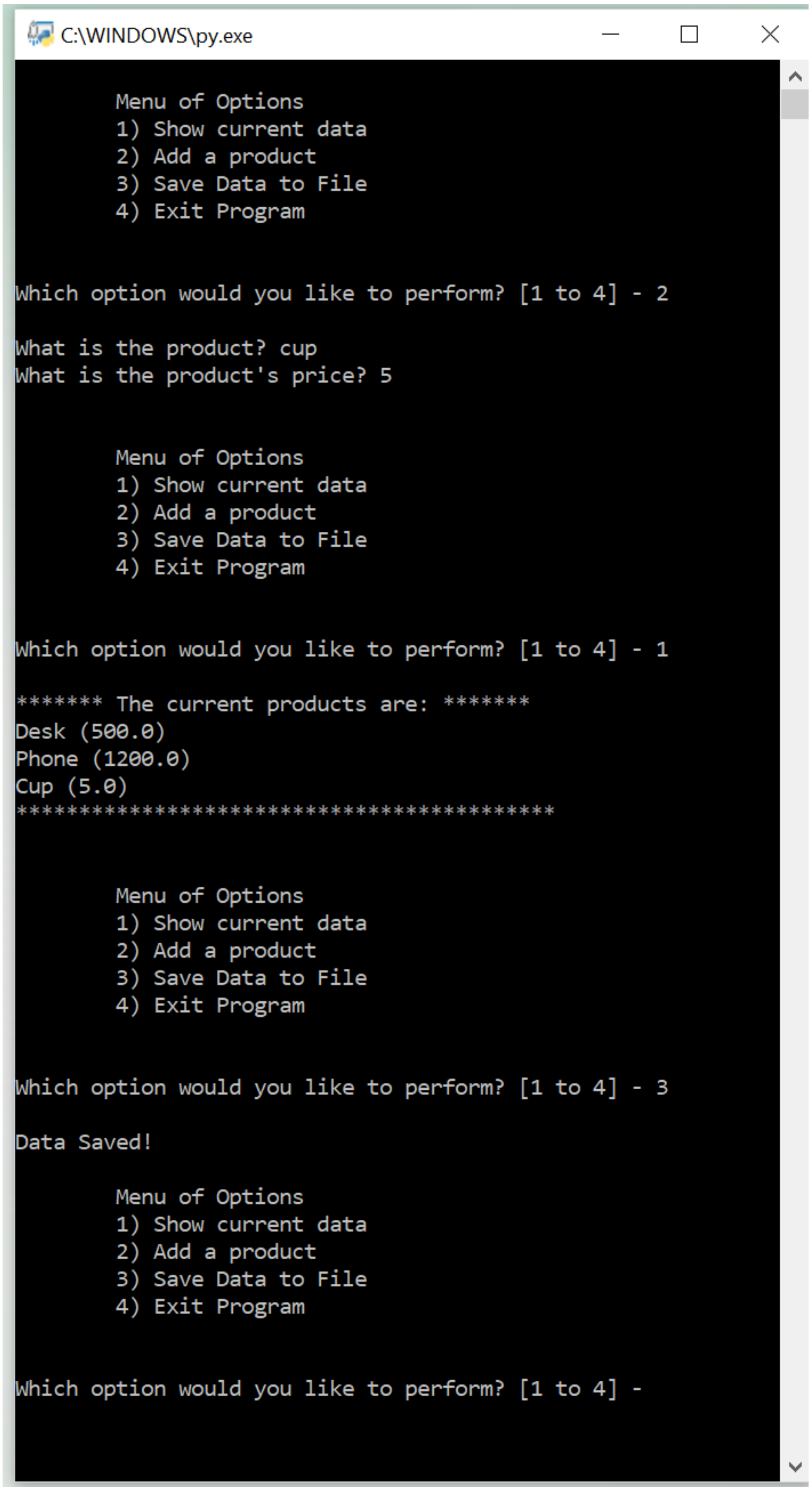


```
# Main Body of Script  ---------------------------------------------------- #

# Load data from file into a list of product objects when script starts

try:
    lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)  # read file data

    # Display a menu of choices to the user
    while (True):
        # Show user a menu of options
        IO.output_menu()
        # Get user's menu choice
        strChoice = IO.input_menu_choice()
        if strChoice.strip() == '1':
            # Show user current data in the list of product objects
            IO.output_current_products_in_list(lstOfProductObjects)
            continue  # to show the menu
        elif strChoice == '2':
            # Let user add data to the list of product objects
            lstOfProductObjects.append(IO.input_new_product())
            continue  # to show the menu
        elif strChoice == '3':
            # let user save current data to file
            FileProcessor.write_data_to_file(strFileName, lstOfProductObjects)
            print("Data Saved!")
            continue  # to show the menu
        elif strChoice == '4':
            # Exit Program
            print("Goodbye!")
            break  # by exiting loop
except Exception as e:
    print("There was an error")
    print(e, e.__doc__, type(e), sep='\n')
# Main Body of Script  ---------------------------------------------------- #
```

Figure 5. Main body of the script

Testing the Script

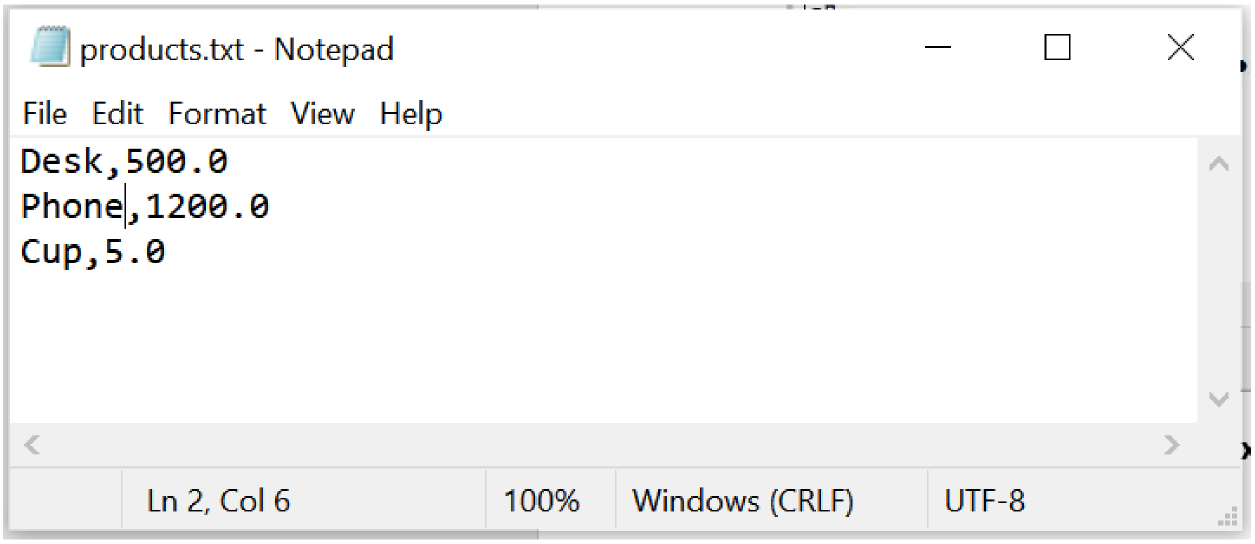
I tested the program in both PyCharm and in the command prompt and it successfully completed using both (**Figure 5**).


C:\WINDOWS\py.exe

```
        Menu of Options
        1) Show current data
        2) Add a product
        3) Save Data to File
        4) Exit Program


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

What is the product? cup
What is the product's price? 5


        Menu of Options
        1) Show current data
        2) Add a product
        3) Save Data to File
        4) Exit Program


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

******* The current products are: *******
Desk (500.0)
Phone (1200.0)
Cup (5.0)
*******************************************


        Menu of Options
        1) Show current data
        2) Add a product
        3) Save Data to File
        4) Exit Program


Which option would you like to perform? [1 to 4] - 3

Data Saved!

        Menu of Options
        1) Show current data
        2) Add a product
        3) Save Data to File
        4) Exit Program


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

Figure 5. Testing the program

The script successfully ran and updated the text file todo.txt (**Figure 6**).



```
Desk,500.0
Phone,1200.0
Cup,5.0
```

Figure 6. Updated products.txt

Summary

For Foundations of Programming: Python seventh assignment I worked on a python script that reads from a file to memory and then gives the user the option to display the information, add a product, write to the file, and end the program.