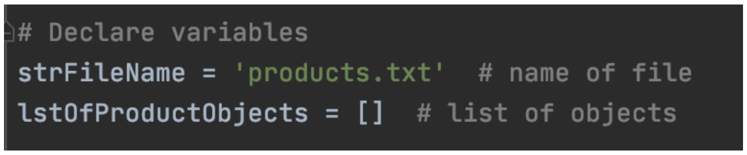
Meg Giddings
June 7, 2020
Foundations of Programming (Python)
Assignment08
GitHub: https://github.com/mgidd/IntroToProg-Python-Mod08

Working with Objects and Classes

Introduction

This paper provides an overview of a basic program that utilizes several classes to perform data storage, processing, and presentation. The data is stored in object instances of a product class and managed with properties and methods. The processing class is used to perform actions on the data, including reading from and saving to a file, and the presentation class captures user input and displays program output. All of these classes and their functions are called in the main body of the script.

After the header, the first section of the script declares the variables to be used in the program (Figure 1).

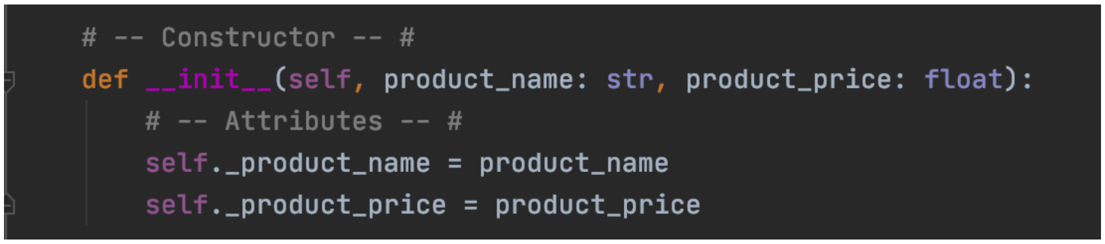


```
# Declare variables
strFileName = 'products.txt'  # name of file
lstOfProductObjects = []  # list of objects
```

Figure 1: Declaring variables to be used in script

Product Class

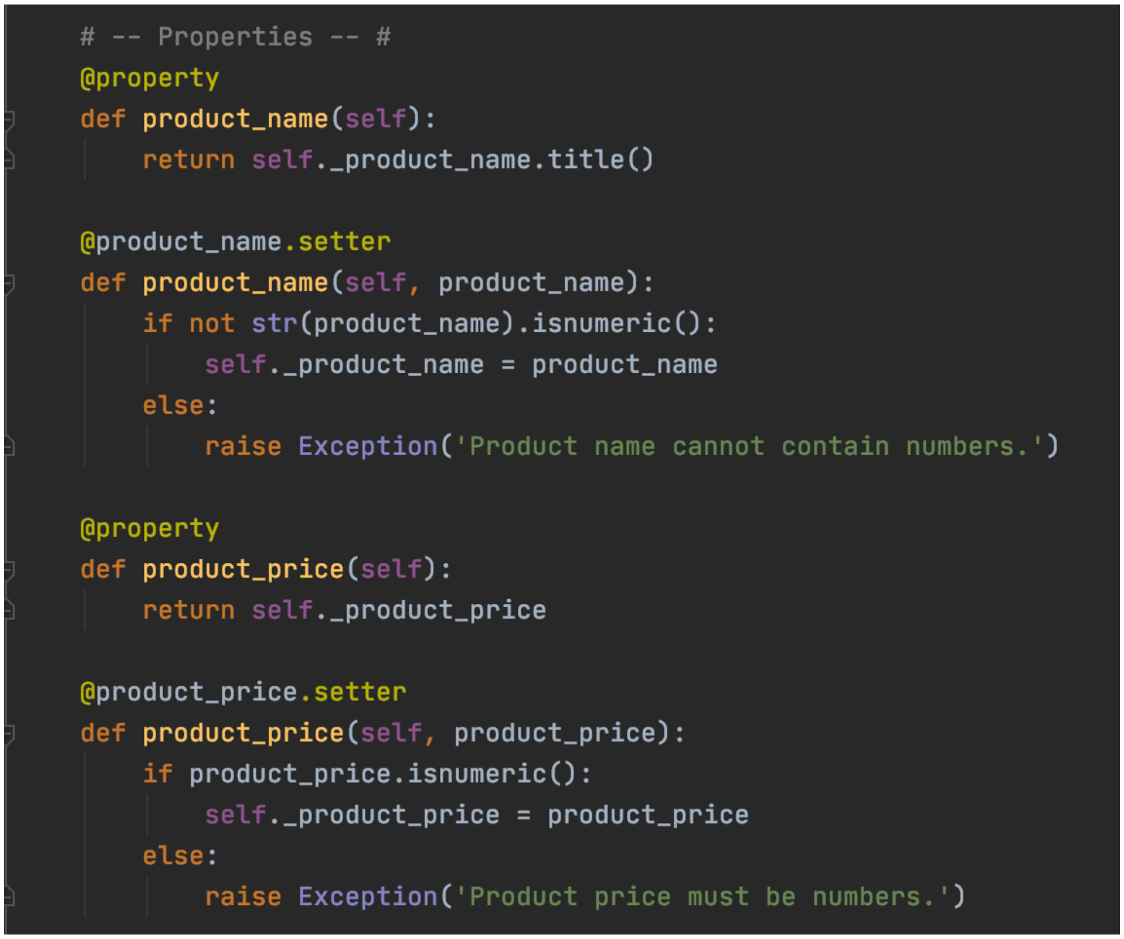
The first class in this script is Product, which stores information about product data. In this case, the product has two attributes, a name and a price. I define these in the class's constructor, including type hints for the name and price parameters (Figure 2).



```
    # -- Constructor -- #
    def __init__(self, product_name: str, product_price: float):
        # -- Attributes -- #
        self._product_name = product_name
        self._product_price = product_price
```

Figure 2: Defining constructor and attributes in Product class

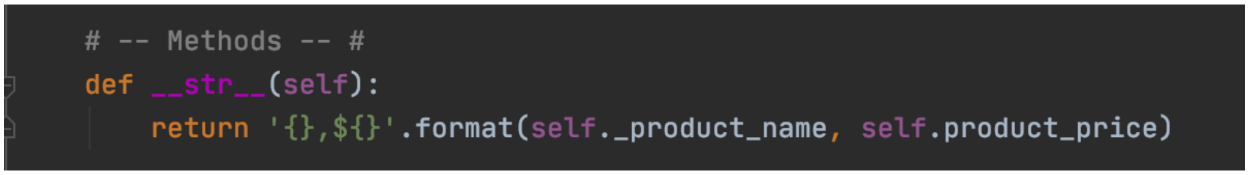
Next, I define the properties of the class – a getter and a setter for both the product name and product price (Figure 3). I include some error handling in the setter properties to ensure that the product name is non-numeric and the product price is only numeric.



```python
# -- Properties -- #
@property
def product_name(self):
    return self._product_name.title()

@product_name.setter
def product_name(self, product_name):
    if not str(product_name).isnumeric():
        self._product_name = product_name
    else:
        raise Exception('Product name cannot contain numbers.')

@property
def product_price(self):
    return self._product_price

@product_price.setter
def product_price(self, product_price):
    if product_price.isnumeric():
        self._product_price = product_price
    else:
        raise Exception('Product price must be numbers.')
```

Figure 3: Defining properties of Product class

Last in the Product class, I define a method for returning the class's data as a string (Figure 4). This overrides the built-in "__str__()" method and allows the program to return data in a custom format.



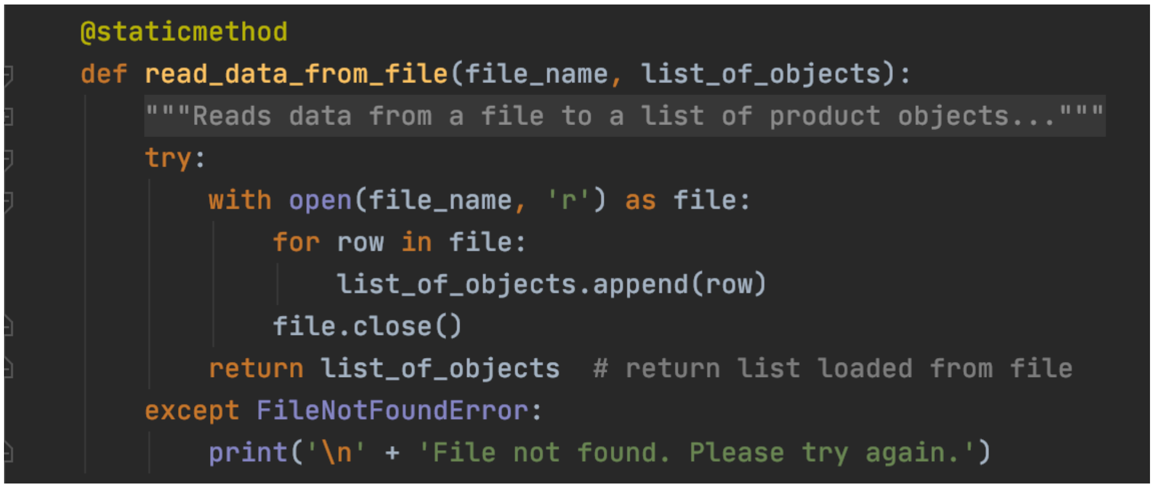
```python
# -- Methods -- #
def __str__(self):
    return '{},${}'.format(self._product_name, self.product_price)
```

Figure 4: Defining string method in Product class

Processing Class

The next class in this program is the FileProcessor class, which includes functions for processing data to and from a file.

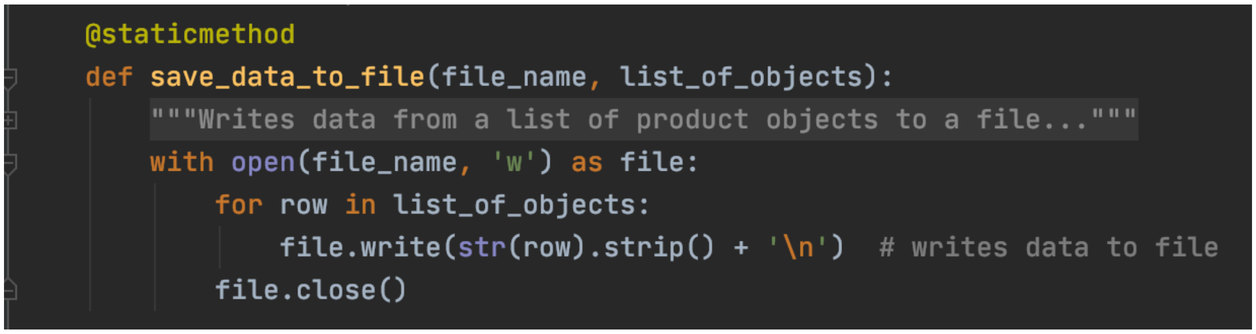
To read data from a file, I use a for loop to append each file row to the list of product objects in the program's memory (Figure 5). This function also includes some error handling for when the program tries to read from a nonexistent file, although because the file is defined in the program and not captured as input from the user, it is unlikely this error will occur, unless the file is deleted.



```python
    @staticmethod
    def read_data_from_file(file_name, list_of_objects):
        """Reads data from a file to a list of product objects..."""
        try:
            with open(file_name, 'r') as file:
                for row in file:
                    list_of_objects.append(row)
                file.close()
            return list_of_objects  # return list loaded from file
        except FileNotFoundError:
            print('\n' + 'File not found. Please try again.')
```

Figure 5: Defining function to read data from a file in FileProcessor class

To write data to a file, I again use a for loop to write each row of the list of product objects to the file, as string data (Figure 6).



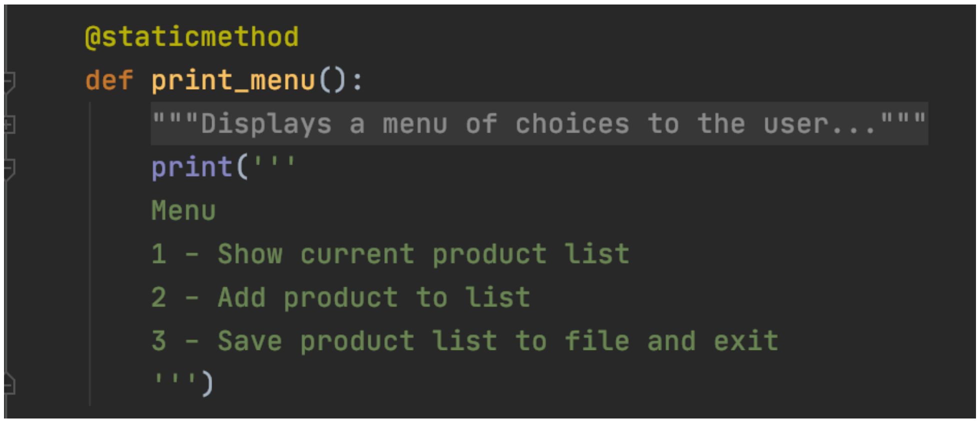
```python
    @staticmethod
    def save_data_to_file(file_name, list_of_objects):
        """Writes data from a list of product objects to a file..."""
        with open(file_name, 'w') as file:
            for row in list_of_objects:
                file.write(str(row).strip() + '\n')  # writes data to file
            file.close()
```

Figure 6: Defining function to write data to a file in FileProcessor class

Input/Output Class

The next class in this program is the IO class, which defines functions for capturing user input and displaying program output.

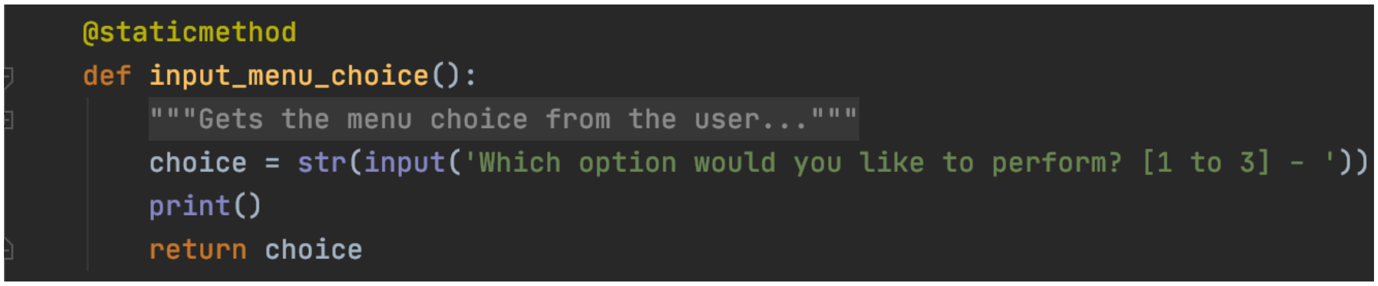
The simplest function prints out a short menu of options to the user (Figure 7).



```python
    @staticmethod
    def print_menu():
        """Displays a menu of choices to the user..."""
        print('''
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit
        ''')
```

Figure 7: Defining function to print menu of options in IO class

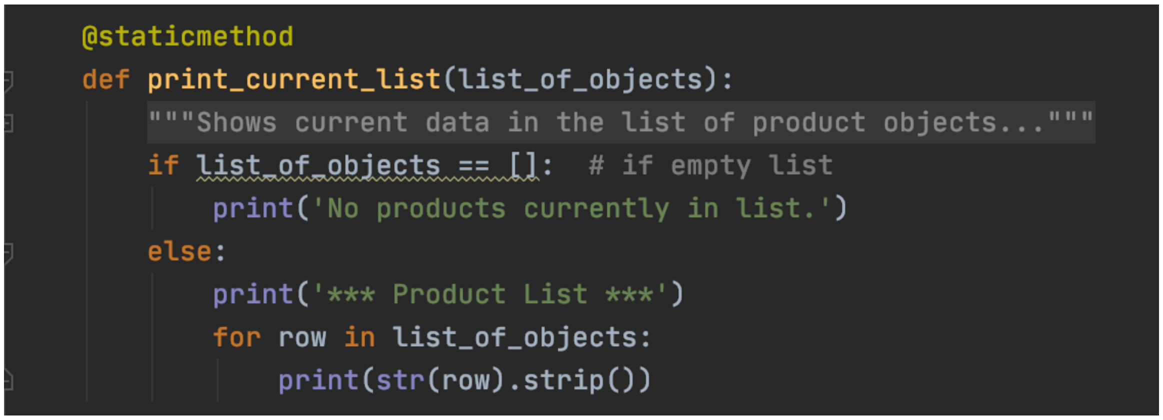
Another function captures the user's choice from the menu of options as a string (Figure 8).



```
    @staticmethod
    def input_menu_choice():
        """Gets the menu choice from the user..."""
        choice = str(input('Which option would you like to perform? [1 to 3] - '))
        print()
        return choice
```

Figure 8: Defining function to get user's menu choice in IO class

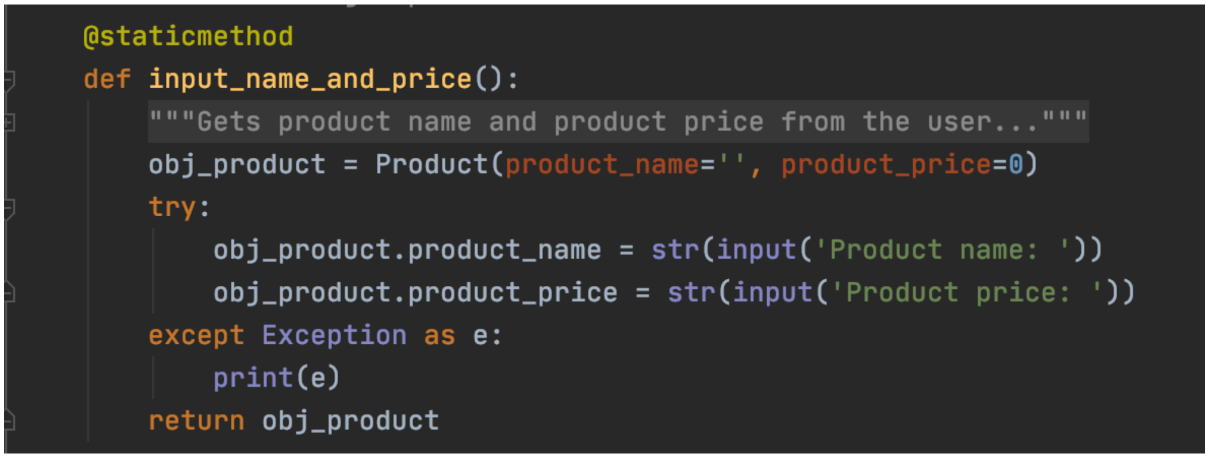
A third function displays the current list of product objects in memory (Figure 9). If there are no objects currently in the list, the user is notified as such.



```
    @staticmethod
    def print_current_list(list_of_objects):
        """Shows current data in the list of product objects..."""
        if list_of_objects == []:  # if empty list
            print('No products currently in list.')
        else:
            print('*** Product List ***')
            for row in list_of_objects:
                print(str(row).strip())
```

Figure 9: Defining function to print current list of product objects in IO class

The final function in class IO captures the user's input for a product name and price (Figure 10). These inputs are assigned to an object instance of the Product class.

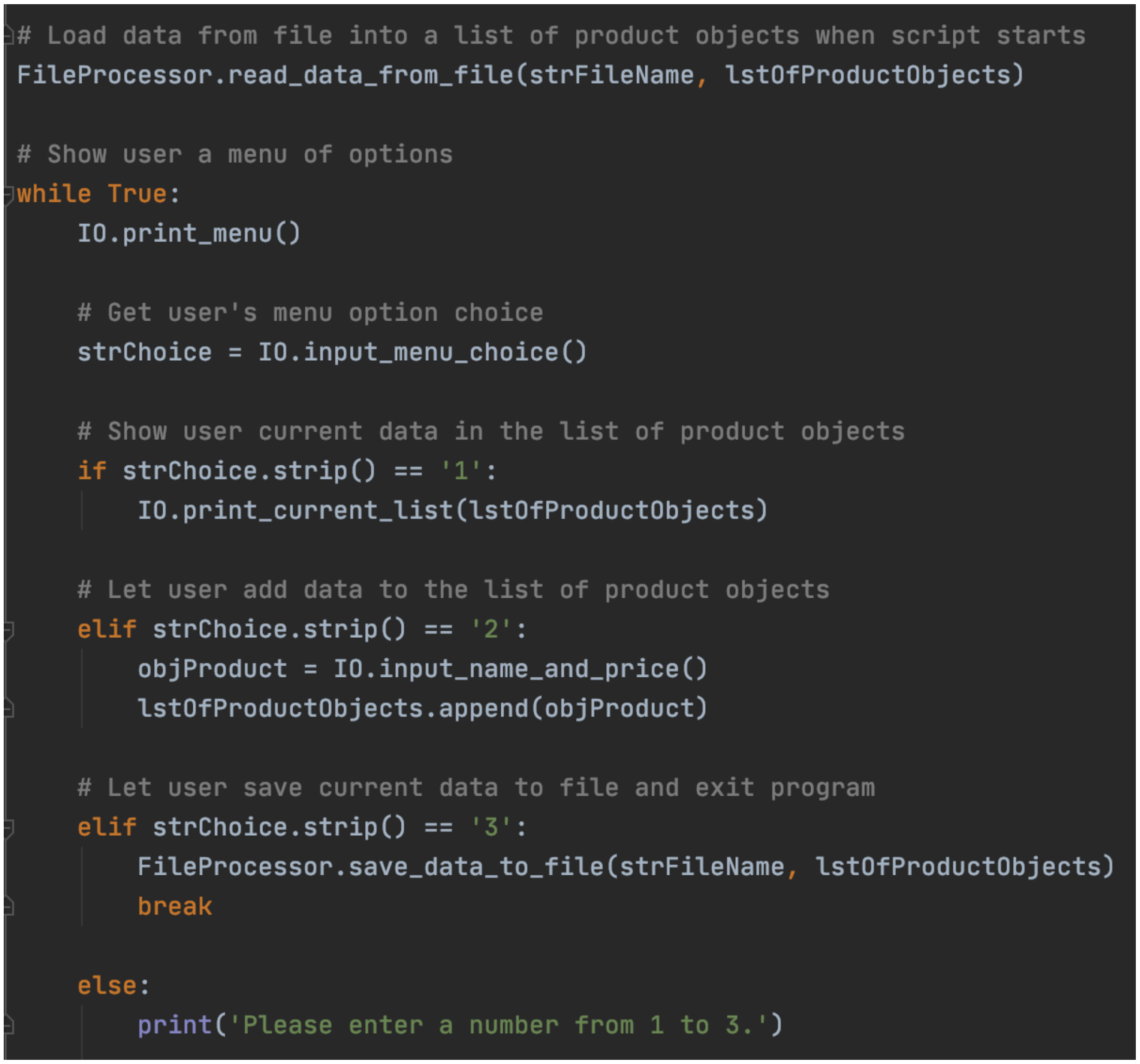


```
    @staticmethod
    def input_name_and_price():
        """Gets product name and product price from the user..."""
        obj_product = Product(product_name='', product_price=0)
        try:
            obj_product.product_name = str(input('Product name: '))
            obj_product.product_price = str(input('Product price: '))
        except Exception as e:
            print(e)
        return obj_product
```

Figure 10: Defining function to get user's input for a product name and price in IO class

Main Body

The main body of the script calls functions of the previously defined classes to perform the program (Figure 11). When the program loads, data is loaded from a file into the list of product objects. Then, a while loop ensures the program repeatedly prints the menu of options following each time the user makes a choice.

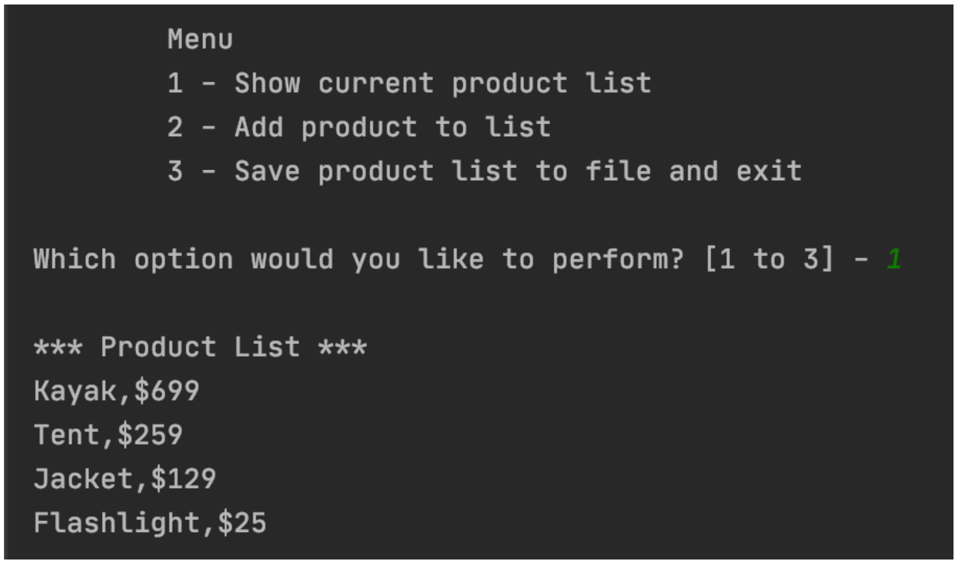


```python
# Load data from file into a list of product objects when script starts
FileProcessor.read_data_from_file(strFileName, lstOfProductObjects)

# Show user a menu of options
while True:
    IO.print_menu()

    # Get user's menu option choice
    strChoice = IO.input_menu_choice()

    # Show user current data in the list of product objects
    if strChoice.strip() == '1':
        IO.print_current_list(lstOfProductObjects)

    # Let user add data to the list of product objects
    elif strChoice.strip() == '2':
        objProduct = IO.input_name_and_price()
        lstOfProductObjects.append(objProduct)

    # Let user save current data to file and exit program
    elif strChoice.strip() == '3':
        FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
        break

    else:
        print('Please enter a number from 1 to 3.')
```

Figure 11: Main body of script

Running the Program

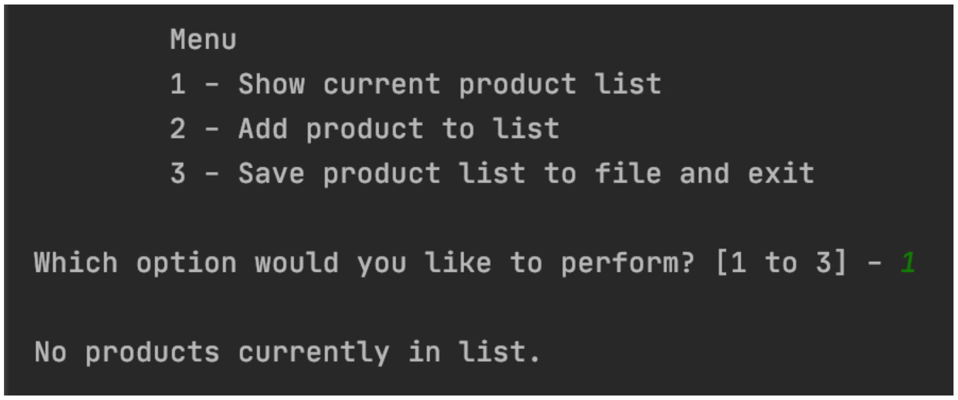
When the user selects option 1, the current product list is displayed (Figure 12).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

*** Product List ***
Kayak,$699
Tent,$259
Jacket,$129
Flashlight,$25
```

Figure 12: Option 1 (show current list) running in PyCharm

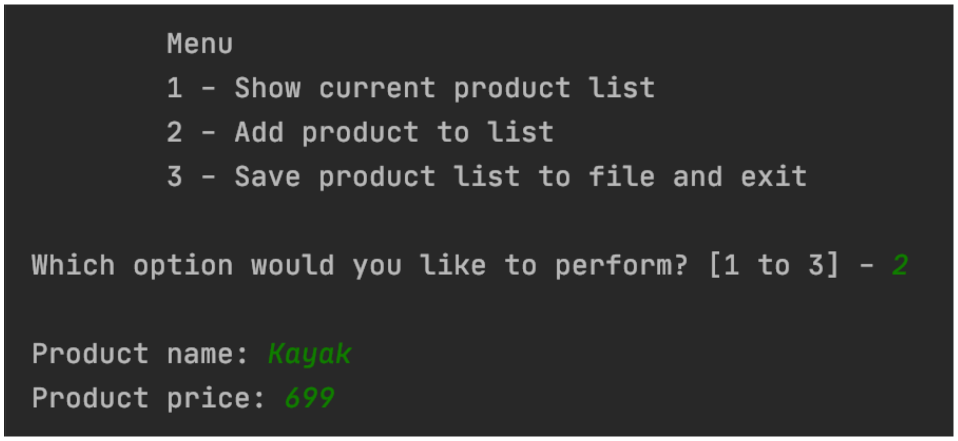
However, if there are no products currently in the list, the user is notified (Figure 13).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

No products currently in list.
```

Figure 13: Option 1 (show current list) running in PyCharm when list of product objects is empty

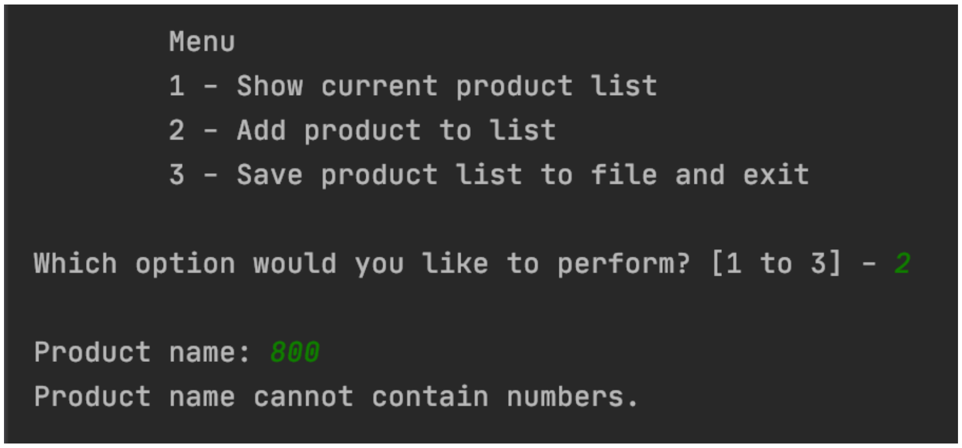
When the user selects option 2, they are prompted to enter a name and a price for the product they want to add to the list (Figure 14).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

Product name: Kayak
Product price: 699
```

Figure 14: Option 2 (add product to list) running in PyCharm

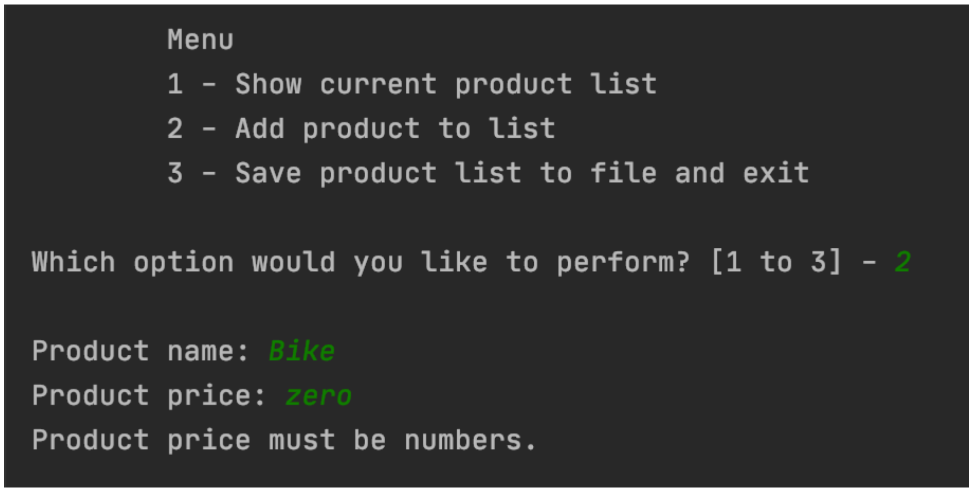
However, if the product name is numeric, the user will be notified (Figure 15).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

Product name: 800
Product name cannot contain numbers.
```

Figure 15: Option 2 (add product to list) running in PyCharm

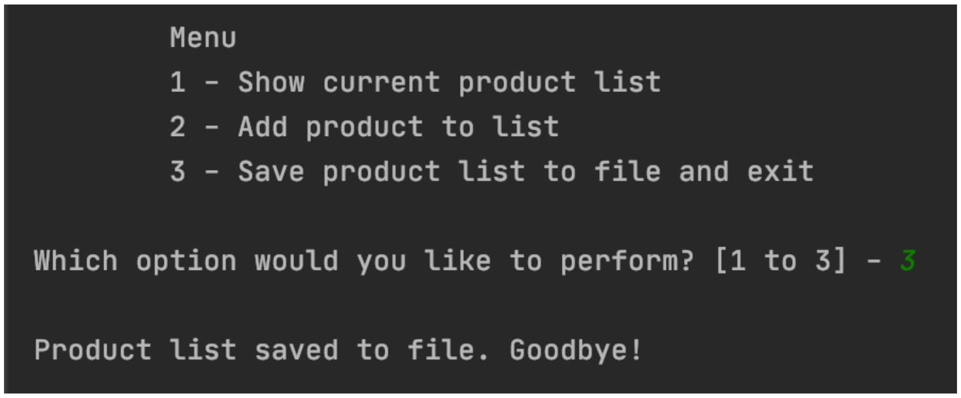
And if the product price is not numeric, the user will similarly be notified (Figure 16).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

Product name: Bike
Product price: zero
Product price must be numbers.
```

Figure 16: Option 2 (add product to list) running in PyCharm

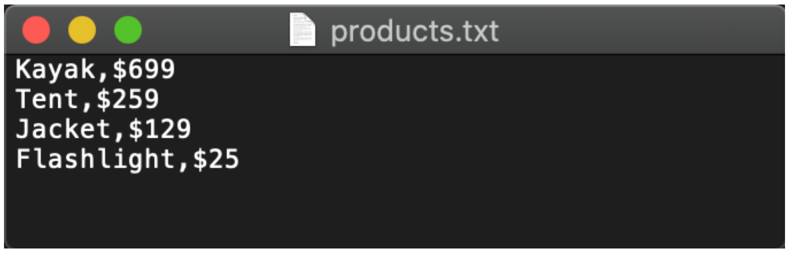
If the user selects option 3, the product list is written to a file (the same file from which data is loaded when the program initializes), the user is notified, and the program ends (Figure 17).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

Product list saved to file. Goodbye!
```

Figure 17: Option 3 (save product list to file and exit) running in PyCharm

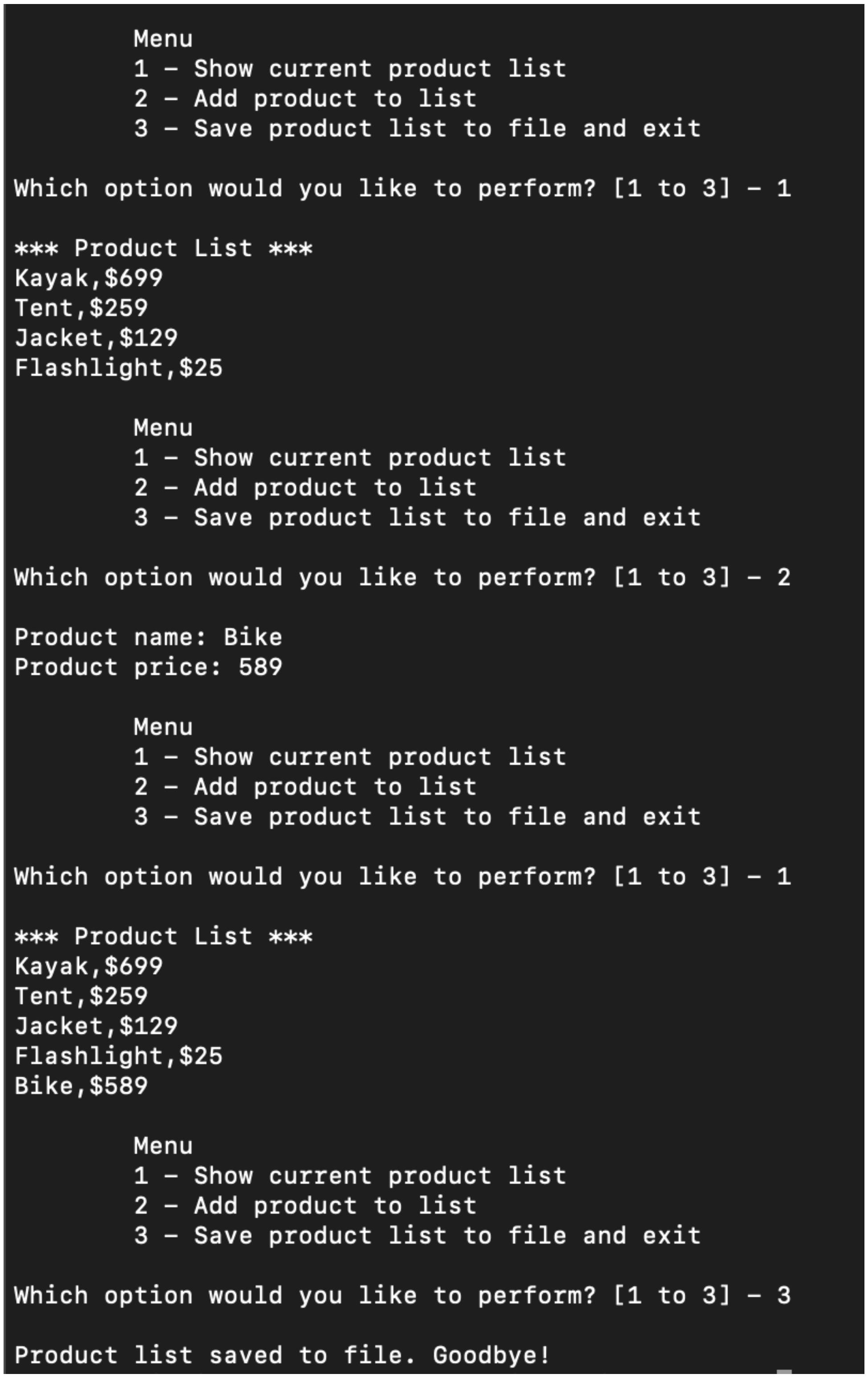
To make sure that the list of product objects actually saved, I checked the relative file (Figure 18).



```
Kayak,$699
Tent,$259
Jacket,$129
Flashlight,$25
```

Figure 18: Verifying data saved to file

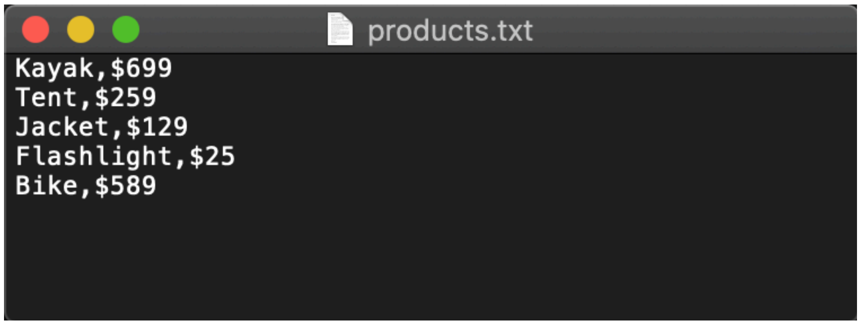
After running this program in PyCharm, I also ran it in Terminal (Figure 19).



```
        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

*** Product List ***
Kayak,$699
Tent,$259
Jacket,$129
Flashlight,$25

        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

Product name: Bike
Product price: 589

        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

*** Product List ***
Kayak,$699
Tent,$259
Jacket,$129
Flashlight,$25
Bike,$589

        Menu
        1 - Show current product list
        2 - Add product to list
        3 - Save product list to file and exit

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

Product list saved to file. Goodbye!
```

Figure 19: Options 1, 2 and 3 running in Terminal

And I again checked the relative file to verify that the list data saved as expected (Figure 20).



```
Kayak,$699
Tent,$259
Jacket,$129
Flashlight,$25
Bike,$589
```

Figure 20: Verifying data saved to file

Summary

The script described in this paper utilizes classes and objects to store and work with data in a "separation of concerns" framework. The program allows a user to view their current list of products, add product data to the list, and save that list of product data to a file, which can then be re-accessed the next time the user runs the program.

While it was challenging to wrap my head around the concept of objects and how the constructor and property methods work, I can absolutely understand now why object-oriented programming has become such a useful technique for working with data sets. I am looking forward to learning more about how to use encapsulation and abstraction to build better programs.