Craig Muth

12/7/2022

IT FDN 110 A Au 22: Foundations Of Programming: Python

Assignment 08

Classes and Objects

Introduction

The goal of this project is to build a custom class to collect, hold, and report data about products. Names and prices will be retrieved from a text file, before creating instances of the class created for each product. Also, a user will be able to add display the existing data, add new products, and save the data back to a text file.

A partial program has been provided to start from. The necessary structure exists, but operational code has been omitted in several places so it will not run as it currently exists. So we will need to map the logic for the existing code, identify where it is lacking, and develop the logic and code to make it work.

Designing the program

Examining the starter file, the logic flow looks like this - with areas where code is missing is indicated.

- Data
    - declare variables and constants
    - class Product
        - *MISSING CODE*
- Processing
    - class FileProcesser
    - Process data from a file
        - *MISSING CODE*
    - Process data to a file
        - *MISSING CODE*
- Presentation (Input/Output)
    - class IO
        - def print_menu_items
            - print list of options to screen
        - Get user's choice
            - *MISSING CODE*
        - Show the current data from the file to user
            - *MISSING CODE*
        - Get product data from the user
            - *MISSING CODE*
- Main Body of Script
    - Load data from file into a list of product objects when script starts

        - *MISSING CODE*
    - Show user a menu of options
        - *MISSING CODE*
    - Get user’s menu option choice
        - *MISSING CODE*
        - Show user current data in the list of product objects
            - *MISSING CODE*
        - Let user add data to the list of product objects
            - *MISSING CODE*
        - Let user save current data to file and exit program
            - *MISSING CODE*

As the program is already divided into classes, we will address the updates for each class separately to make it easier to follow the changes as they are introduced.

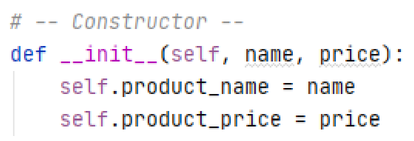
The Product class

First, we will create a Product class to define a data structure that will serve as an example from which Objects (instances of the Product class) will be built. (*Python Classes and Objects,* (September 8, 2022). Retrieved December 6, 2022, from https://www.geeksforgeeks.org/python-classes-and-objects/) (External link). Within the class, the code will include a constructor to initialize the object’s attributes, properties to modify those attributes, and methods to control how the objects interact with the main program.

Since the goal is to create a simple list of products and their price, the only interaction we will control is the output format when the Class data is called for display. So the logic for the definition of the class looks something like this:

- class Product
    - Receive name and price from main program
    - Initiate the attributes with the given values
    - Process the user given data to match the data type and format of the attributes
    - Define how the attributes will be displayed

Each object should have a unique combination of name and price when created, so a parameterized constructor will be used (*Constructor in Python with Examples*, (n.d.). Retrieved December 6, 2022, from https://pythongeeks.org/constructor-in-python/) (External Link). By defining the `__init__()` method inside the class and using the argument `self`, we can assign the values given by the user to the memory address of the object we are creating like this:



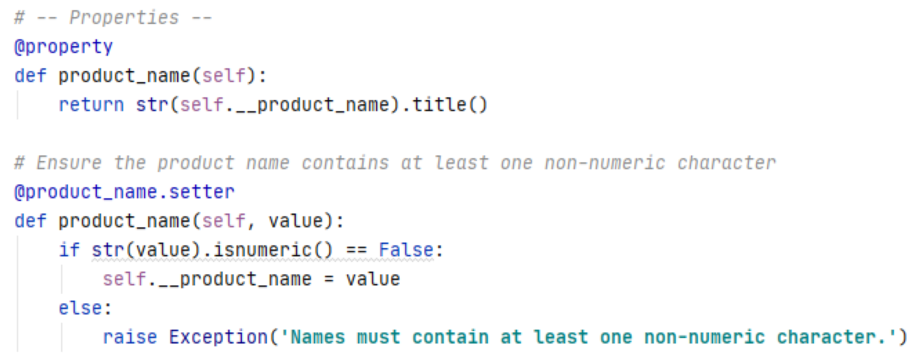
```
# -- Constructor --
def __init__(self, name, price):
    self.product_name = name
    self.product_price = price
```

Figure 1: Product class constructor

Each time the class used to create an object, a new memory location is assigned the given values. Allowing objects with identical structure, but unique attributes, to be generated without modifying the template.

If we could be certain the users will always input the data the way we want, nothing else should be needed. However, to be safe, it would be better to provide a means to ensure the data is in the correct format by either using input masking (which may work for a program with either a web or graphic interface) or by modifying the data after it's passed into the class (which should work with a web, graphic, or consoler interface).

By creating Properties within in the class, we can perform actions on the data to ensure the data is either in or converted to the correct formats before the class is created (Ramos, Leodanis Pozo (n.d.). *Python's property(): Add Managed Attributes to Your Classes*, https://realpython.com/python-property/) (External link).

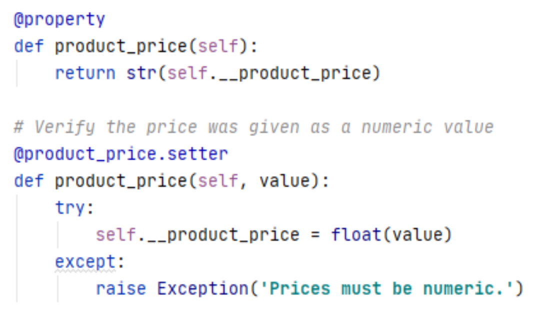


```python
# -- Properties --
@property
def product_name(self):
    return str(self.__product_name).title()

# Ensure the product name contains at least one non-numeric character
@product_name.setter
def product_name(self, value):
    if str(value).isnumeric() == False:
        self.__product_name = value
    else:
        raise Exception('Names must contain at least one non-numeric character.')
```

Figure 2: Product name property

In the code above the first property (`product_name(self)`) allows the program to access the `product_name` value stored in the Object, while the `@product_name.setter` allows the program to modify this value. Before the Object is created the attribute has no inherent value, but this second property (the setter) checks the input from the user to verify it's not all numbers, and if so assigns the value to the `product_name` attribute that was created by the constructor.

For the `product_price` we want to convert the value to a float before assigning it to the Object. By wrapping the conversion and assignment in a `try…except` statement, and throw a custom exception if the conversion fails.



```python
@property
def product_price(self):
    return str(self.__product_price)

# Verify the price was given as a numeric value
@product_price.setter
def product_price(self, value):
    try:
        self.__product_price = float(value)
    except:
        raise Exception('Prices must be numeric.')
```

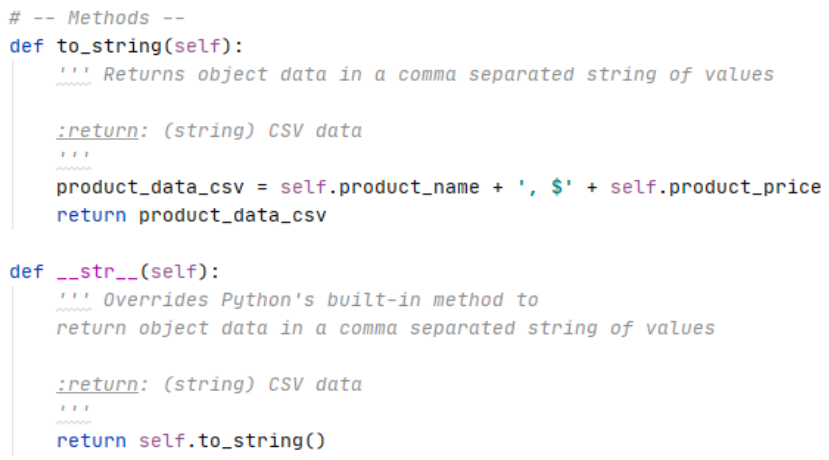
Figure 3: Product price property

This will cause an exception if the user includes any currency symbol in the input, but we will address that when adding new data, before calling the Product class.

The final piece to be added to the Product class is the methods (functions within the class) to provide action to perform on the internal data we've defined. We will add a new method to format the data and override a built-in method to display the data as we've defined.

The `to_string` method uses the `self` reference to extract name and price from the current object's memory location – then combines them into a variable so the value can be passed to other methods. We can add a currency indicator to be included in the output.

By default, the class includes a built-in `__str__` method that when called will display the memory location for the given instance when invoked by the print statement (`print(<object>)` or `print(<object>.__str__)` (Introduction to the Python __str__ method (n.d.). Retrieved December 7, 2022 from https://www.pythontutorial.net/python-oop/python-__str__/) (External link). We can override this behavior by reusing the name and providing our own code.



```
# -- Methods --
def to_string(self):
    ''' Returns object data in a comma separated string of values

    :return: (string) CSV data
    '''
    product_data_csv = self.product_name + ', $' + self.product_price
    return product_data_csv

def __str__(self):
    ''' Overrides Python's built-in method to
    return object data in a comma separated string of values

    :return: (string) CSV data
    '''
    return self.to_string()
```

Figure 4: Product class methods

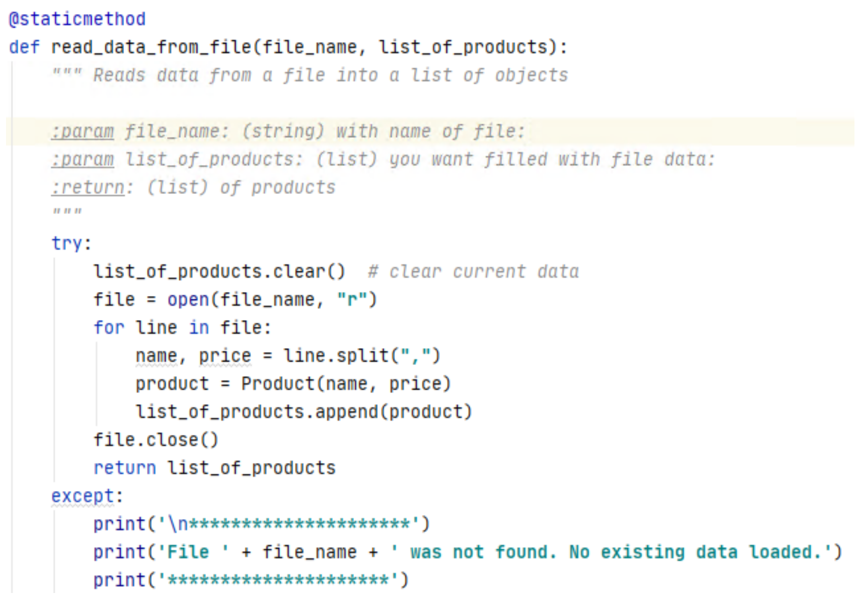
Now when the `__str__` method is invoked, it in turn calls the `to_string` method to build the string from the attributes and returns that value to the screen. If there were additional attributes, only those addressed in the `to_string` method would be returned – so this method can be used to make it simpler for the user to get the values you want them to see without knowing the internal structure of the class/object.

The FileProcessor class

The logic for the read function is very similar to that used for other projects. The most significant change is instead of assigning the values from the file into a dictionary – it will create an instance of the Product class, populating its attributes with the values from the file.

- Process data from a file
    - Receive file name and product list from main program
    - If file exists
        - Open file in read mode
        - Read line from file
        - Call Product class to create object
        - Add object to product list
        - If not end of file
            - Go to Read line from file

```
        • Else
            • Close file
            • Return product list to main program
```



```python
@staticmethod
def read_data_from_file(file_name, list_of_products):
    """ Reads data from a file into a list of objects

    :param file_name: (string) with name of file:
    :param list_of_products: (list) you want filled with file data:
    :return: (list) of products
    """
    try:
        list_of_products.clear()  # clear current data
        file = open(file_name, "r")
        for line in file:
            name, price = line.split(",")
            product = Product(name, price)
            list_of_products.append(product)
        file.close()
        return list_of_products
    except:
        print('\n**********************')
        print('File ' + file_name + ' was not found. No existing data loaded.')
        print('**********************')
```

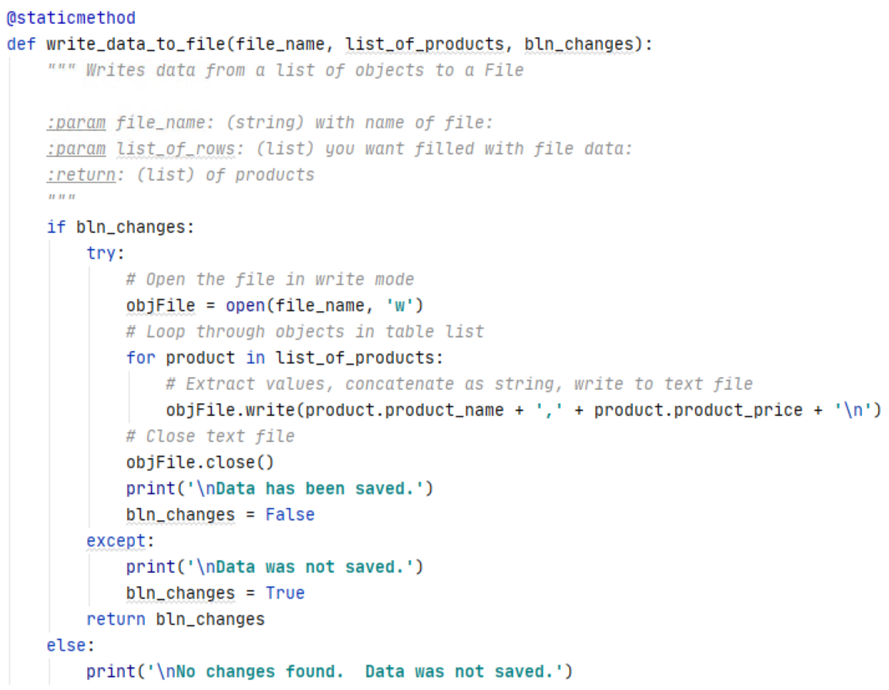
Figure 5: Process data from a text file

During each iteration of the for loop, the line `product = Product(name, price)` passes the new values to the class, where the constructor and properties create a new instance that is assigned to the object named `product`. Since the objects are then added to a list, they can be addressed individually using the list index, so for this application it's not an issue they are given the same name. If they were not managed in some type of collection like this, it would be necessary to assign a unique identifier for each object.

Writing the data to a file also closely follows the flow used in other projects.

```
• Process data to a file
    • Receive file name and product list from main program
    • Open file in write mode
    • Get object from list
        • Write product name and price to file
        • If not end of list
            • Go to Get object from list
        • Else
            • Close file
```

The notable difference is how the code access the values of the object's properties. Similar to the dictionary, the values are addressed by the name – but in this case the format is `<object>.<property>` rather than `<dictionary>['key value']`.



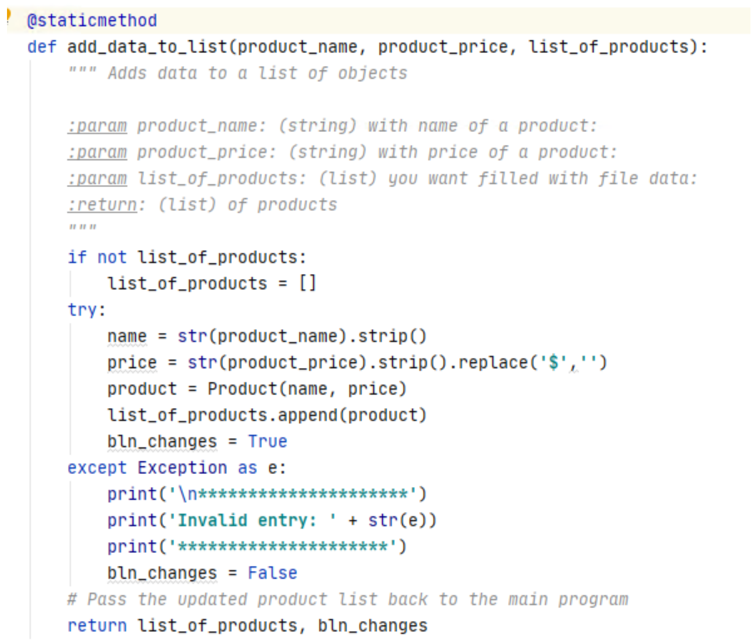
```
@staticmethod
def write_data_to_file(file_name, list_of_products, bln_changes):
    """ Writes data from a list of objects to a File

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of products
    """
    if bln_changes:
        try:
            # Open the file in write mode
            objFile = open(file_name, 'w')
            # Loop through objects in table list
            for product in list_of_products:
                # Extract values, concatenate as string, write to text file
                objFile.write(product.product_name + ',' + product.product_price + '\n')
            # Close text file
            objFile.close()
            print('\nData has been saved.')
            bln_changes = False
        except:
            print('\nData was not saved.')
            bln_changes = True
        return bln_changes
    else:
        print('\nNo changes found.  Data was not saved.')
```

Figure 6: Write data to a text file

The DataProcessor class

This class was not in the original file, but was added because the functionality doesn't quite fit with the methods in the FileProcesser class.

- Add data to list
    - Receive part name, price, and product list from main program
    - Remove dollar sign from price if present
    - Call Product class to create object
    - Add object to product list
    - Set change flag to True
    - Return product list and change flag to main program



```
@staticmethod
def add_data_to_list(product_name, product_price, list_of_products):
    """ Adds data to a list of objects

    :param product_name: (string) with name of a product:
    :param product_price: (string) with price of a product:
    :param list_of_products: (list) you want filled with file data:
    :return: (list) of products
    """
    if not list_of_products:
        list_of_products = []
    try:
        name = str(product_name).strip()
        price = str(product_price).strip().replace('$','')
        product = Product(name, price)
        list_of_products.append(product)
        bln_changes = True
    except Exception as e:
        print('\n*********************')
        print('Invalid entry: ' + str(e))
        print('*********************')
        bln_changes = False
    # Pass the updated product list back to the main program
    return list_of_products, bln_changes
```

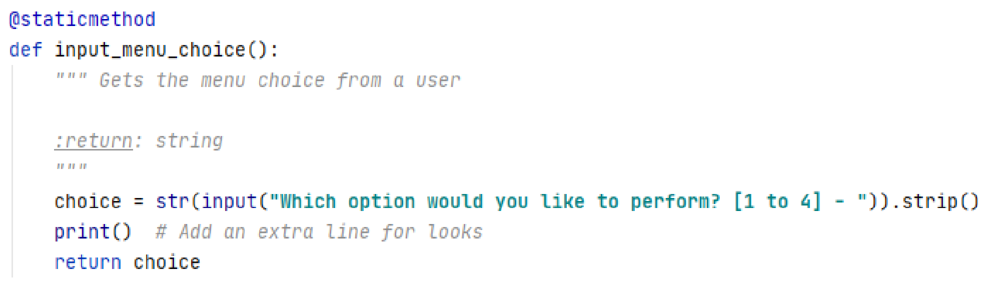
Figure 7: Add new product to list in memory

As mentioned previously, you can see this method uses a `replace` statement to remove the dollar sign from the value in `product_price` before calling the Product class. If there is a chance the user may enter prices in other currencies, additional symbols could be added – but additional code would be required to capture, record, and report with the appropriate symbol.

The IO class

Getting the user menu selection can be done with a simple input statement – we will account for invalid selections in the main program, so the value can be passed back without further processing.

- Get user's choice
    - choice = str(input("Which option would you like to perform? [1 to 4] - ")).strip()
    - Return choice to main program



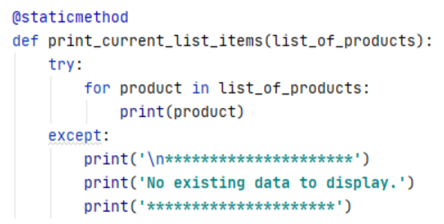
```
@staticmethod
def input_menu_choice():
    """ Gets the menu choice from a user

    :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
```

Figure 8: Get user selection from menu choices

Because the methods in the class are controlling how the properties are displayed, we don't need to address them individually – though we can, as shown previously in the write function. To display the data in the default format defined the class we can print it directly using `print(product)`.

- Show the current data from the file to user
    - Receive product list
        - Get object from list

        - print(product)
        - if not end of file
            - Go to Get object from list

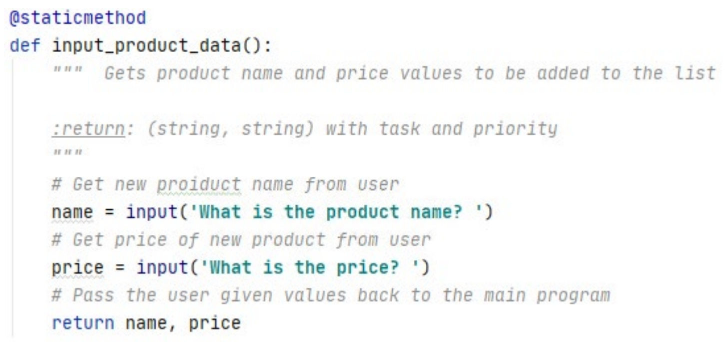


```python
@staticmethod
def print_current_list_items(list_of_products):
    try:
        for product in list_of_products:
            print(product)
    except:
        print('\n**********************')
        print('No existing data to display.')
        print('**********************')
```

Figure 9: Display the data in memory

Between the `add_data_to_list` method and the methods in the Product class we have already provided code manage incorrect data formats – so again, getting the product name and price from the user can be done with `input` and `return` statements similar to getting the menu selection.

- Get product data from the user
    - Request product name from user
    - Request product price from user
    - Return name and price to main program



```python
@staticmethod
def input_product_data():
    """  Gets product name and price values to be added to the list

    :return: (string, string) with task and priority
    """
    # Get new proiduct name from user
    name = input('What is the product name? ')
    # Get price of new product from user
    price = input('What is the price? ')
    # Pass the user given values back to the main program
    return name, price
```

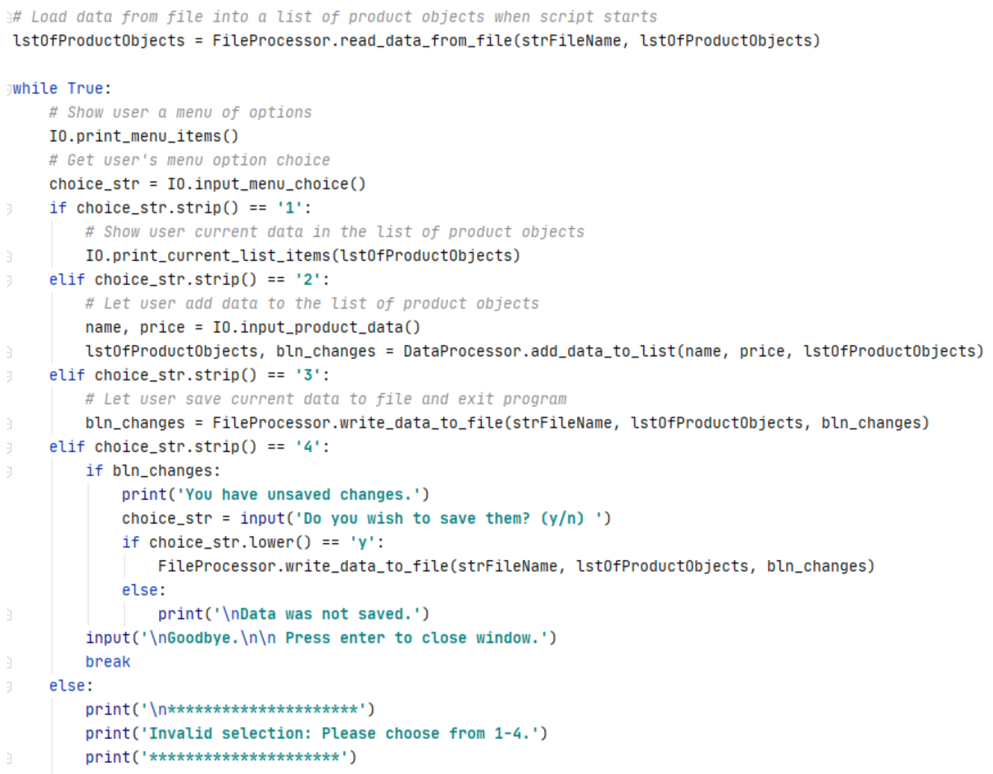
Figure 10: Get new product data

The main body of script

The classes and methods will be created but not run when the program is launched, so the main body will need to call the appropriate methods in turn, and as mentioned earlier also handle invalid inputs for the menu options.

- Load data from file into a list of product objects when script starts
    - Call FileProcessor.read_data_from_file and assign value to lstOfProductObjects
- Show user a menu of options
    - Call IO.print_menu_items
- Get user's menu option choice
    - Call IO.input_menu_choice
    - If show current data in the list of product objects
        - Call IO.print_current_list_items
    - If add data to the list of product objects
        - Call IO.input_product_data

        - Call DataProcessor.add_data_to_list
    - If save current data to file
        - FileProcessor.write_data_to_file
    - If exit program
        - If change flag is True
            - Notify user of unsaved changes and give option to save
            - If Yes
                - Call FileProcessor.write_data_to_file
            - Else
                - Display message that data was not saved
            - Close program
    - Else
        - Display invalid selection message
        - Go to Get user’s menu option choice



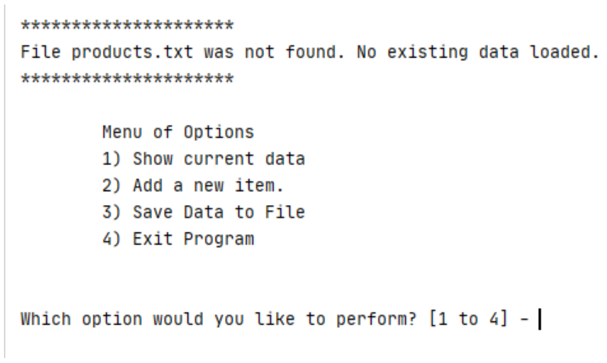
```python
# Load data from file into a list of product objects when script starts
lstOfProductObjects = FileProcessor.read_data_from_file(strFileName, lstOfProductObjects)

while True:
    # Show user a menu of options
    IO.print_menu_items()
    # Get user's menu option choice
    choice_str = IO.input_menu_choice()
    if choice_str.strip() == '1':
        # Show user current data in the list of product objects
        IO.print_current_list_items(lstOfProductObjects)
    elif choice_str.strip() == '2':
        # Let user add data to the list of product objects
        name, price = IO.input_product_data()
        lstOfProductObjects, bln_changes = DataProcessor.add_data_to_list(name, price, lstOfProductObjects)
    elif choice_str.strip() == '3':
        # Let user save current data to file and exit program
        bln_changes = FileProcessor.write_data_to_file(strFileName, lstOfProductObjects, bln_changes)
    elif choice_str.strip() == '4':
        if bln_changes:
            print('You have unsaved changes.')
            choice_str = input('Do you wish to save them? (y/n) ')
            if choice_str.lower() == 'y':
                FileProcessor.write_data_to_file(strFileName, lstOfProductObjects, bln_changes)
            else:
                print('\nData was not saved.')
        input('\nGoodbye.\n\n Press enter to close window.')
        break
    else:
        print('\n*********************')
        print('Invalid selection: Please choose from 1-4.')
        print('*********************')
```

Figure 11: Main program script

Running the program

Executing the program in Pycharm:

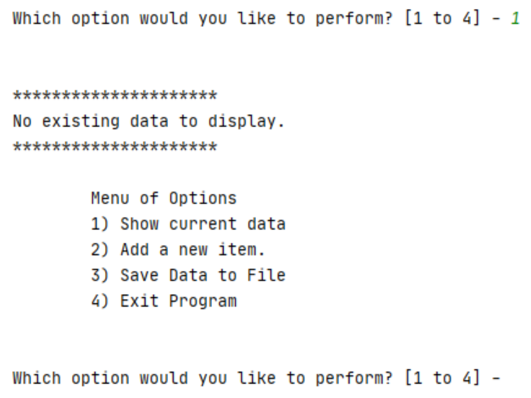
For the initial run the text file has been removed so that we can test the exception handler in the method to read the data from a file.



```
*********************
File products.txt was not found. No existing data loaded.
*********************

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


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

Figure 12: No data file available

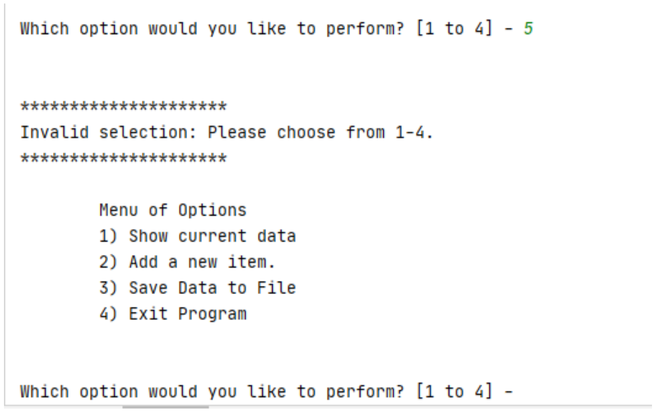
With no data to load, if the user tries to display the current data the program would crash. So another exception handler was included in the display method.



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


*********************
No existing data to display.
*********************

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


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

Figure 13: Attempt to display data with no data loaded

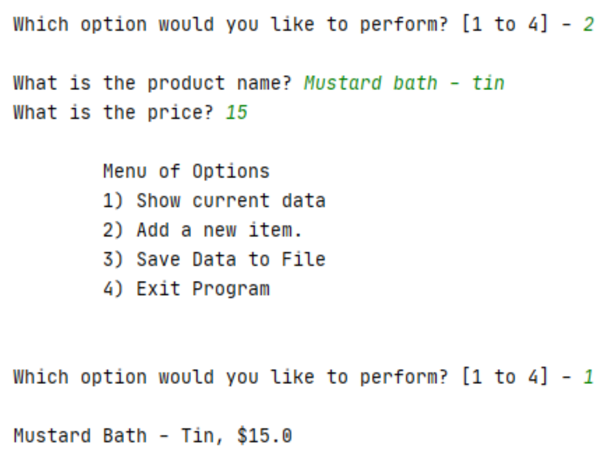
Though not using an exception handler this time - the program also notifies the user if they select an invalid option from the menu.



```
Which option would you like to perform? [1 to 4] - 5


*********************
Invalid selection: Please choose from 1-4.
*********************

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


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

Figure 14: Invalid menu selection

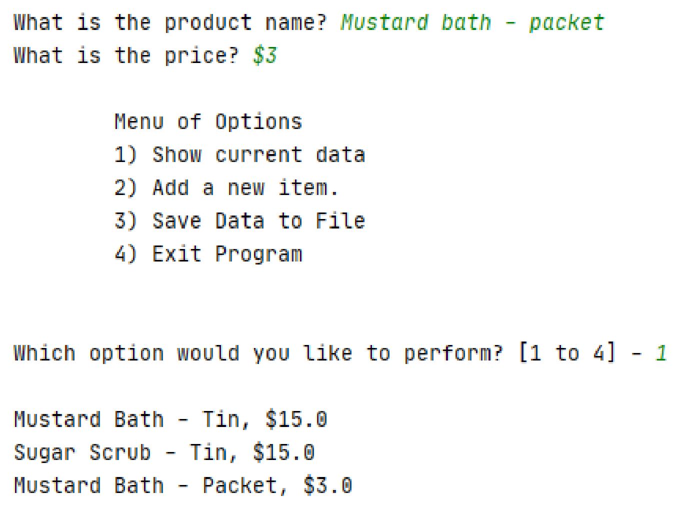
Selecting option 2, we can see the program works for a single entry in the following example.



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

What is the product name? Mustard bath - tin
What is the price? 15

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


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

Mustard Bath - Tin, $15.0
```

Figure 15: Add a new product, and display data

Adding a few more entries, to verify it can handle multiple products:



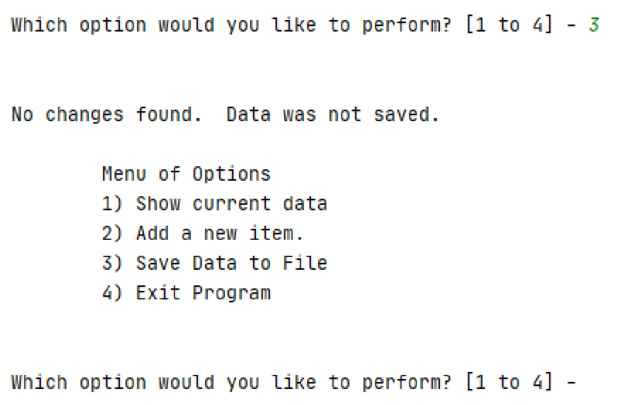
```
What is the product name? Mustard bath - packet
What is the price? $3

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


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

Mustard Bath - Tin, $15.0
Sugar Scrub - Tin, $15.0
Mustard Bath - Packet, $3.0
```

Figure 16: Add more products

You will notice in this example the dollar sign was included in the price input, but in the previous example it was not included. However, in both cases the dollar sign does appear in the output.

This is because the add_data_to_list method contains a replace statement to remove the dollar sign (if present) before handing the values off to the Product class. Within the Product class the price value is converted from a string to a float, which causes extra leading and trailing spaces to be dropped – so the program will accept a price that is all digits or contains a dollar sign, even if there are spaces between the symbol and the numbers.

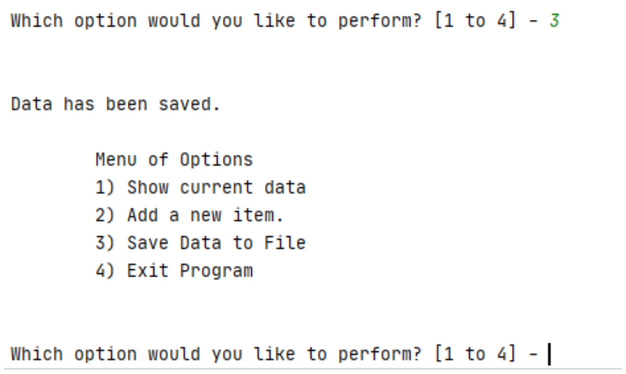
By including the variable bln_changes, we have a flag that can be used to identify if there are unsaved change in the dataset. If the user does not enter any data it remains at the default value of None, and when the user tries to save I notifies them there were no changes and no data was saved.



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


No changes found.  Data was not saved.

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


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

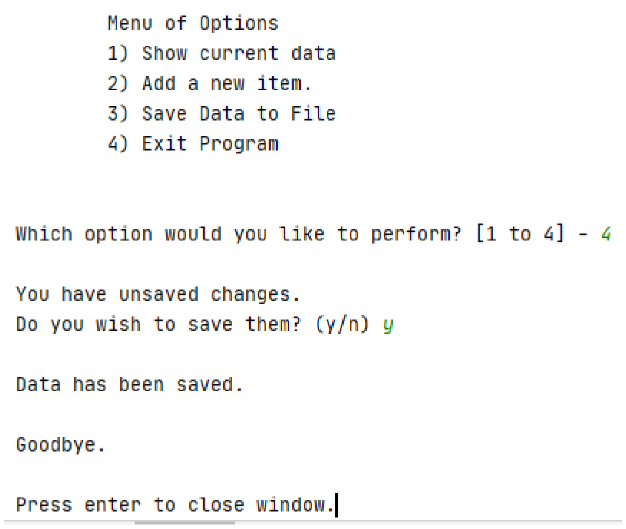
Figure 17: Attempt to save without making changes

When the user adds a new item, the add_data_to_list method resets the value of bln_changes to True so the user can save the data. If they attempt to exit before saving, they get notification of the unsaved changes and an option to save before the program closes. Once the data has been saved, bln_changes is set to False to indicate the data in memory matches the data in the file.



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


Data has been saved.

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


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

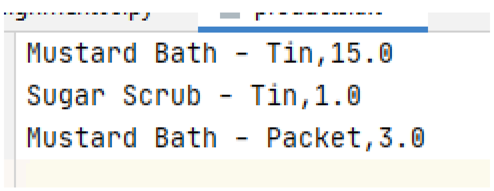
Figure 18: Attempt to save after making changes



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


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

You have unsaved changes.
Do you wish to save them? (y/n) y

Data has been saved.

Goodbye.

Press enter to close window.
```

Figure 19: Exit before saving

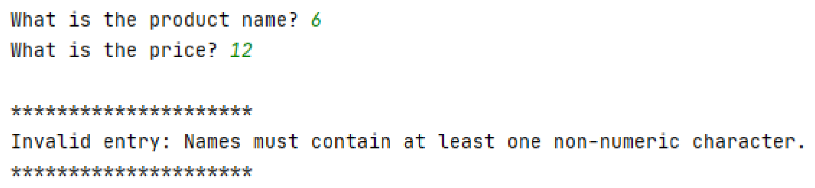
Checking the text file to ensure the data was saved, we see the following.



```
Mustard Bath - Tin,15.0
Sugar Scrub - Tin,1.0
Mustard Bath - Packet,3.0
```

Figure 20: Verify contents of text file

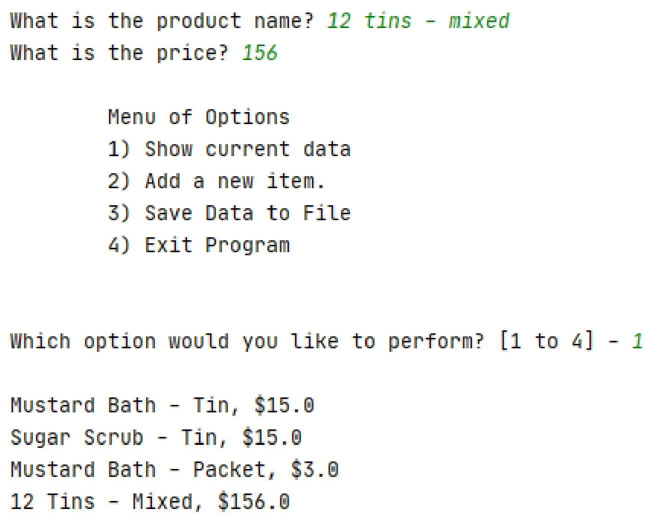
So far everything appears to be working – but what if the user gives invalid values? Within the Product class we required that the product name contain at least one non-numeric character. If we try to enter a number for the name we see the following.



```
What is the product name? 6
What is the price? 12

*********************
Invalid entry: Names must contain at least one non-numeric character.
*********************
```

Figure 21: Invalid product name

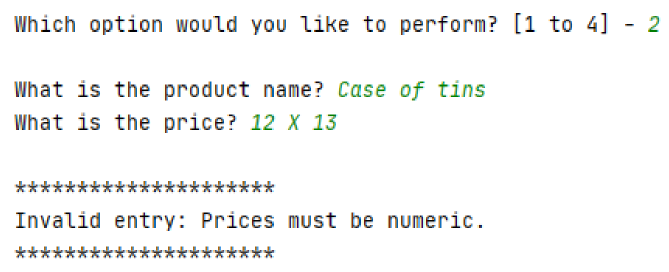
However, we can still include numbers in the name, as long as it contains some non-numeric characters.



```
What is the product name? 12 tins - mixed
What is the price? 156

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


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

Mustard Bath - Tin, $15.0
Sugar Scrub - Tin, $15.0
Mustard Bath - Packet, $3.0
12 Tins - Mixed, $156.0
```

Figure 22: Mixed alpha numeric product name

For the price, we specified the value must contain only numbers (with the exception of the dollar sign as previously mentioned) – so if instead of explicitly giving the price, the user intends to calculate it later and enters something like this:



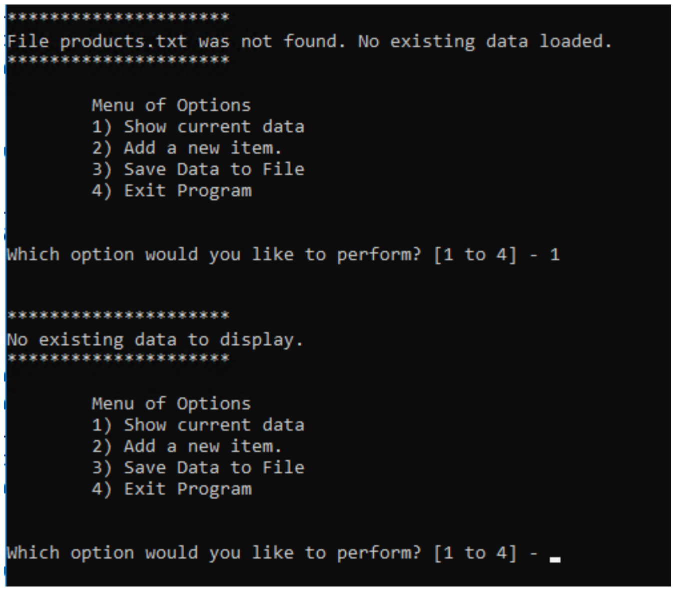
```
Which option would you like to perform? [1 to 4] - 2

What is the product name? Case of tins
What is the price? 12 X 13

*********************
Invalid entry: Prices must be numeric.
*********************
```

Figure 23: Invalid product price

We see the program rejects the entry as intended.

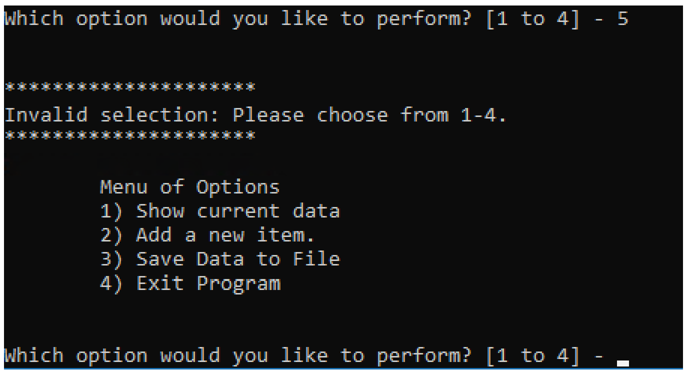
Executing in a Terminal window:

Deleting the products.txt file and running the code in a console window we see the following interactions.



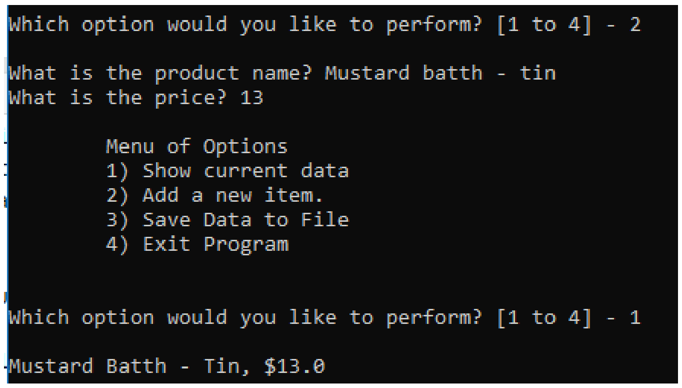
```
*********************
File products.txt was not found. No existing data loaded.
*********************

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

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

*********************
No existing data to display.
*********************

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

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

Figure 24: No file available, and display empty list



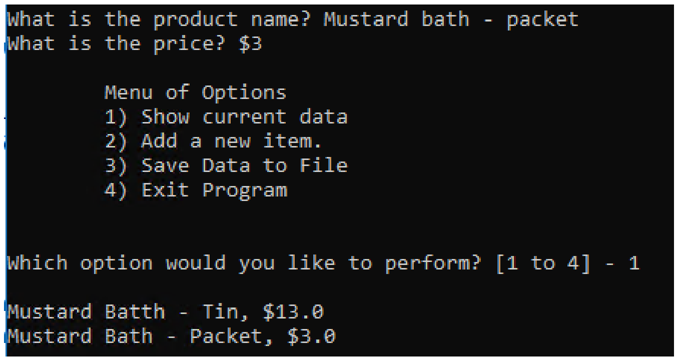
```
Which option would you like to perform? [1 to 4] - 5

*********************
Invalid selection: Please choose from 1-4.
*********************

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

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

Figure 25: Invalid menu selection



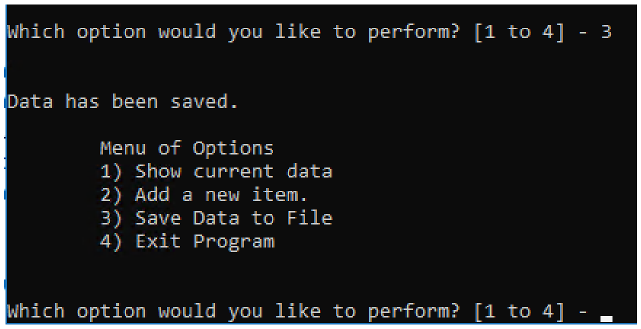
```
Which option would you like to perform? [1 to 4] - 2

What is the product name? Mustard batth - tin
What is the price? 13

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


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

Mustard Batth - Tin, $13.0
```

Figure 26: Add new product (no dollar sign) and display



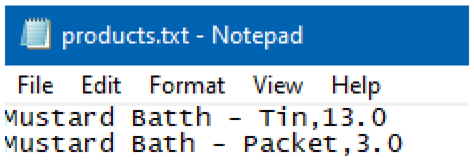
```
What is the product name? Mustard bath - packet
What is the price? $3

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


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

Mustard Batth - Tin, $13.0
Mustard Bath - Packet, $3.0
```

Figure 27: Add new product (with dollar sign) and display



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


Data has been saved.

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


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

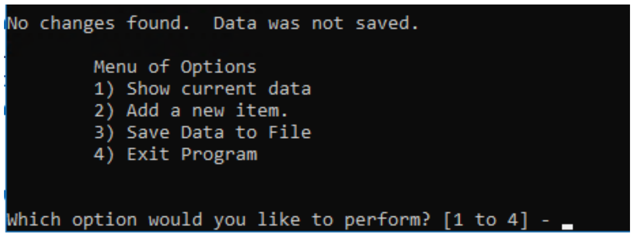
Figure 28: Save data



```
Mustard Batth - Tin,13.0
Mustard Bath - Packet,3.0
```

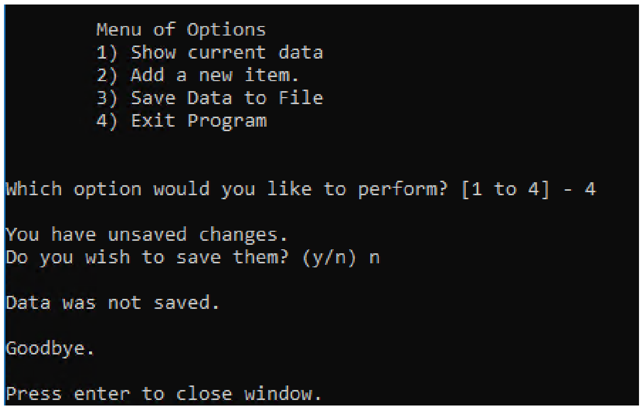
Figure 29: Verify contents of text file

While the misspelling was not intentional – it was left in to help show that we are not using the previous text file, and program is saving as it should.



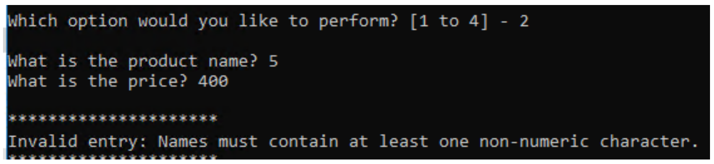
```
No changes found.  Data was not saved.

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


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

Figure 30: Save without making changes



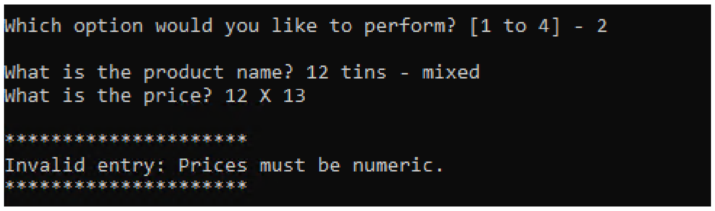
```
        Menu of Options
        1) Show current data
        2) Add a new item.
        3) Save Data to File
        4) Exit Program


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

You have unsaved changes.
Do you wish to save them? (y/n) n

Data was not saved.

Goodbye.

Press enter to close window.
```

Figure 31: Exit without saving



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

What is the product name? 5
What is the price? 400

*********************
Invalid entry: Names must contain at least one non-numeric character.
*********************
```

Figure 32: Invalid product name



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

What is the product name? 12 tins - mixed
What is the price? 12 X 13

*********************
Invalid entry: Prices must be numeric.
*********************
```

Figure 33: Invalid price

Everything appears to be working the same as it did in Pycharm.

Summary

In this project we created a simple class to collect, reformat and report data for a product.

Using a constructor and the `self` reference, the attributes given by the user or read from a file were used to create new instances using the class as a template to ensure they had a common structure and behavior, while still existing as separate objects within memory.

Within the class, properties were created to validate and modify the incoming data so that it matched the types intended for the individual elements without explicitly constraining the user inputs.

Finally, methods were created to define a custom output format, and override a built-in method and allow the custom output to be displayed when the object is called by the print statement.

While the example used was quite simple, it still demonstrated some of the benefits of using a class to define a common structure for creating data object. Though it doesn't demonstrate the behavior of inheritance (*Python Inheritance* (n.d.) Retrieved December 7, 2022 from https://www.w3schools.com/python/python_inheritance.asp) (External site) – it's not too difficult to understand how we could create new classes based on this one, then add attributes or override methods similar to the way it was done here. The common traits (properties and methods) would be defined once in the base class, while specialized attributes are introduced in derived classes so they are only applied to those products where they are applicable.