Craig Muth

12/5/2022

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

Assignment 08

<<https://github.com/cjmuth/IntroToProg-Python-Mod08>>

Classes and Objects

# Introduction

The goal of this project is …

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*

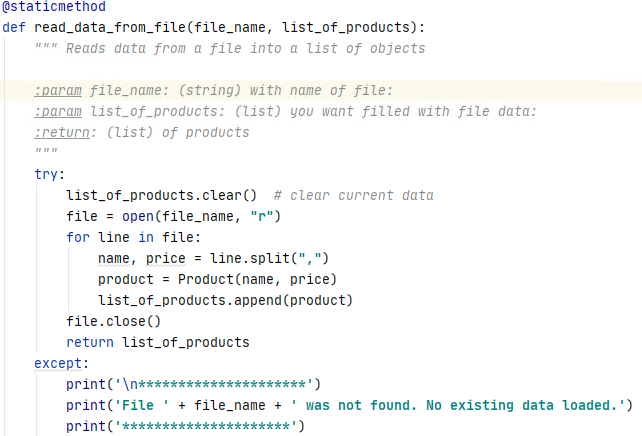
As the program is already divided into classes, we will address the updates for each class separately to make it easier to follow the changes as they are introduced.

### The Product class

* class Product
  + MISSING CODE

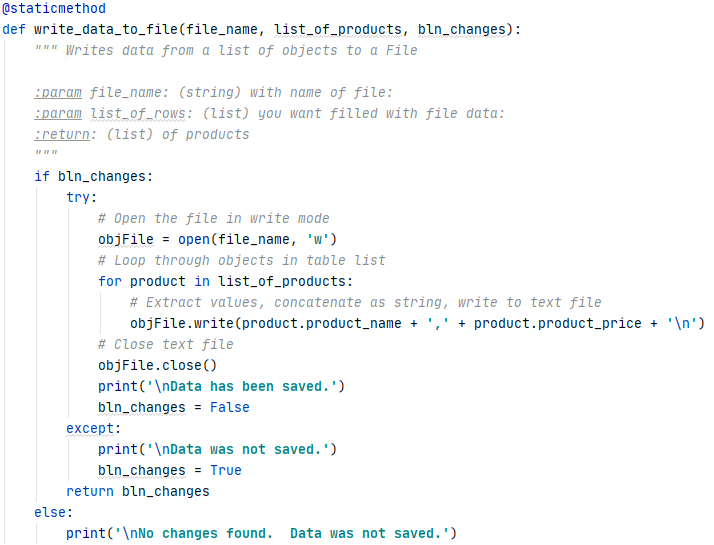
### The FileProcessor class

* + 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



***Figure 2:***

* + 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

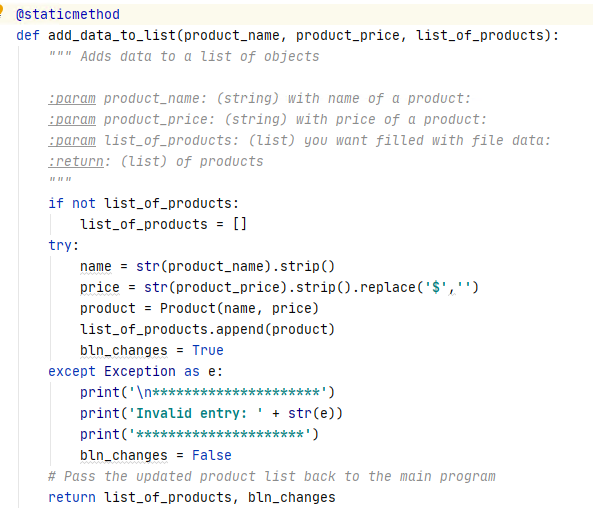


***Figure 3:***

### 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.

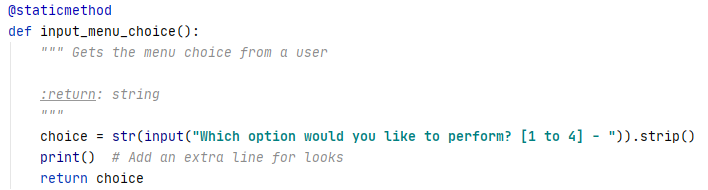
* Add data to list
  + Receive part name, price, and product list from main program
  + Remove dollar sign from price if present
  + Call Product class to create object
  + Add object to product list
  + Set change flag to True
  + Return product list and change flag to main program



***Figure 4:***

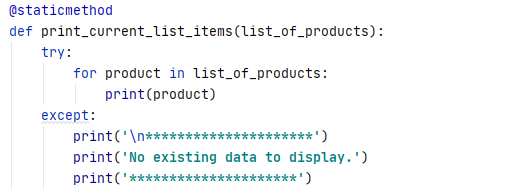
### The IO class

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



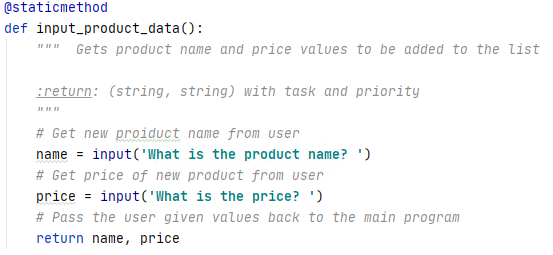
***Figure 5:***

* + - 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



***Figure 6:***

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



***Figure 7:***

### The main body of script

* 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

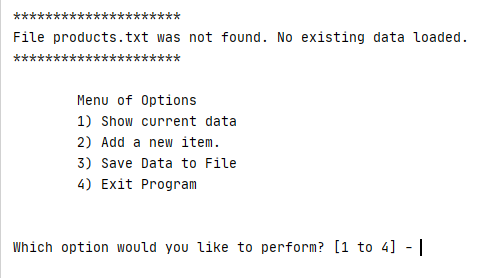


***Figure 8:***

# Running the program

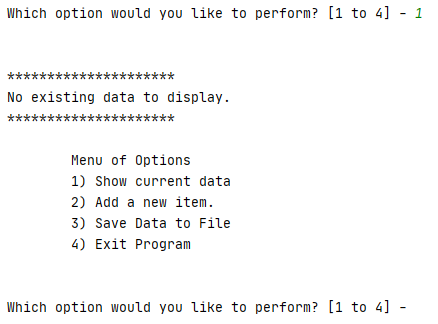
### Executing the program in Pycharm:

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.



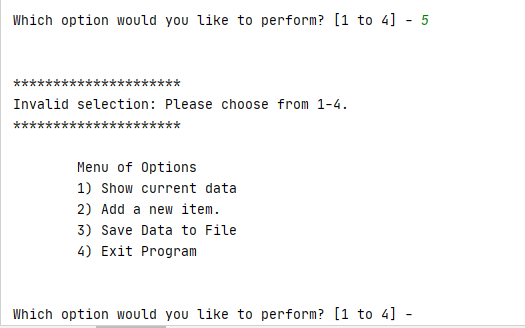
***Figure 9: No data file available***

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.



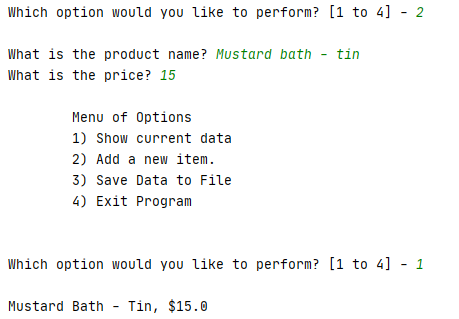
***Figure 10: Attempt to display data with no data loaded***

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



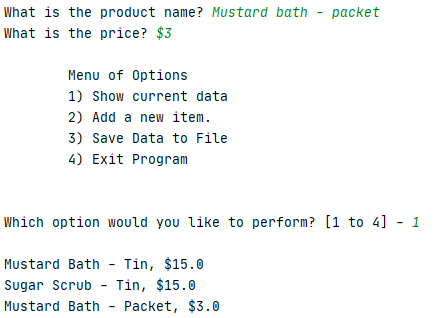
***Figure 11: Invalid menu selection***

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



***Figure 12: Add a new product, and display data***

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

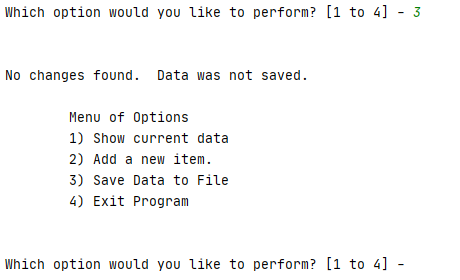


***Figure 13: Add more products***

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.

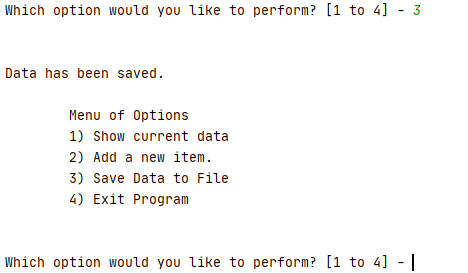
This is because the add\_data\_to\_list method contains a replace statement to remove the dollar sign (if present) before handing the values off to the Product class. Within the Product class the price value is converted from a string to a float, which causes extra leading and trailing spaces to be dropped – 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.

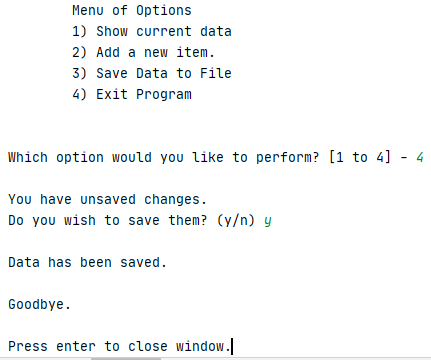


***Figure 14: Attempt to save without making changes***

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.

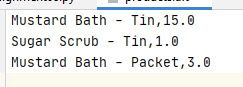


***Figure 15: Attempt to save after making changes***



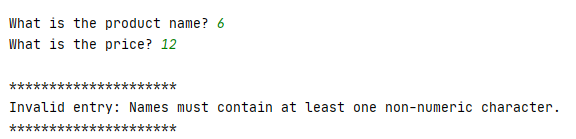
***Figure 16: Exit before saving***

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

******

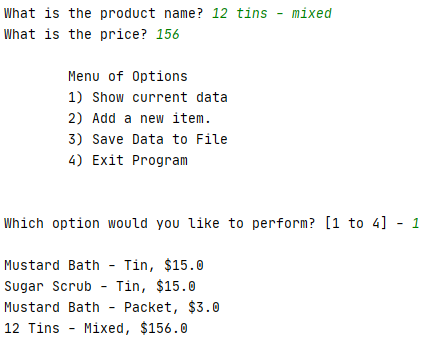
***Figure 17: Verify contents of text file***

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.



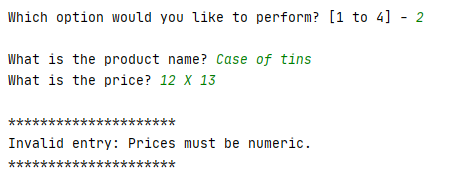
***Figure 18: Invalid product name***

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



***Figure 19: 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:

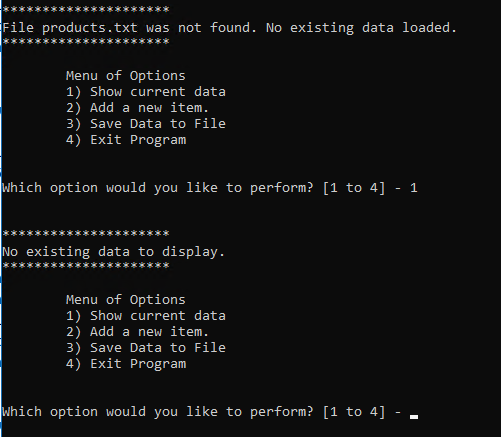


***Figure 20: Invalid product price***

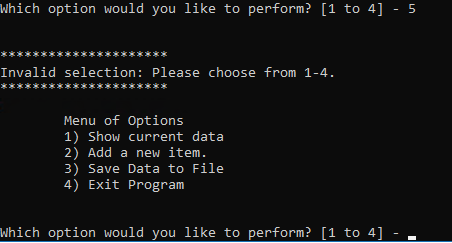
We see the program rejects the entry as intended.

### Executing in a Terminal window:

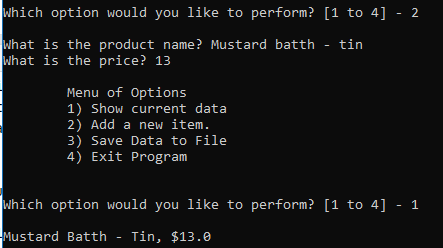
Deleting the products.txt file and running the code in a console window we see the following interactions.



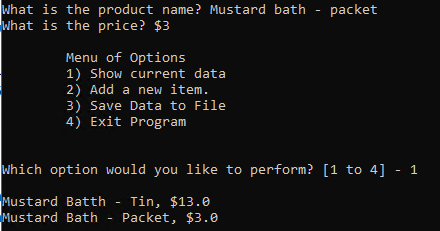
***Figure 21: No file available, and display empty list***



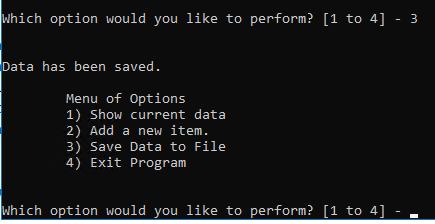
***Figure 22: Invalid menu selection***



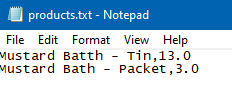
***Figure 23: Add new product (no dollar sign) and display***



***Figure 23: Add new product (with dollar sign) and display***

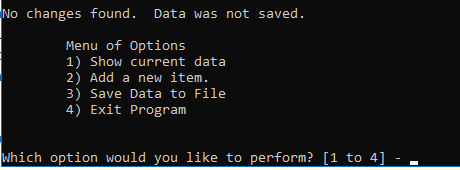


***Figure 24: Save data***

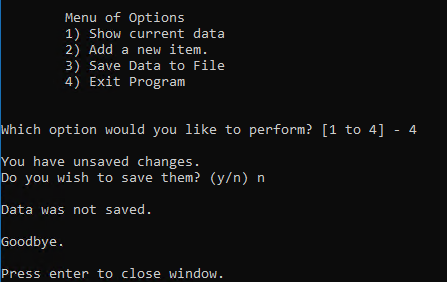


***Figure 25: Verify contents of text file***

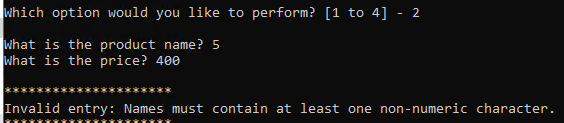
While the misspelling was not intentional – it was left in to help show that we are not using the previous text file, and program is saving as it should.



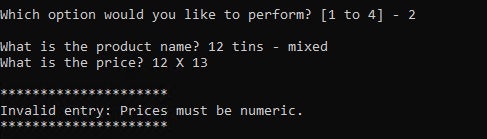
***Figure 26: Save without making changes***



***Figure 27: Exit without saving***

******

***Figure 28: Invalid product name***

******

***Figure 29: Invalid price***

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

# Summary

In this project