Rex Sarabia

28 August 2022

IT FDN 110 B

Assignment08

GitHub repo: <https://github.com/rsar-uw/IntroToProg-Python-Mod08> (External)

Python Script: Products

Table of Contents

[2 Introduction 2](#_Toc112880793)

[3 My system information 2](#_Toc112880794)

[3.1 Operating system (OS) 2](#_Toc112880795)

[3.2 Console 2](#_Toc112880796)

[3.3 Shell 3](#_Toc112880797)

[3.4 Python 3](#_Toc112880798)

[3.5 Integrated Development Environment (IDE) 3](#_Toc112880799)

[3.6 Directory / File path 3](#_Toc112880800)

[4 Module assignment 5](#_Toc112880801)

[4.1 Requirements 5](#_Toc112880802)

[4.1.1 Out of scope 5](#_Toc112880803)

[4.1.2 Lessons learned 5](#_Toc112880804)

[4.2 Design 6](#_Toc112880805)

[4.2.1 Standard elements 6](#_Toc112880806)

[4.2.1.1 Code style 6](#_Toc112880807)

[4.2.1.2 Script header 6](#_Toc112880808)

[4.2.1.3 Saving files 6](#_Toc112880809)

[4.2.2 Program description 7](#_Toc112880810)

[4.2.3 Data structure 7](#_Toc112880811)

[4.2.4 Program architecture 8](#_Toc112880812)

[4.2.4.1 Product class 9](#_Toc112880837)

[4.2.4.2 FileProcessor class 11](#_Toc112880838)

[4.2.4.3 IO class 12](#_Toc112880839)

[4.2.4.4 Main body 14](#_Toc112880840)

[4.2.5 Proposed solution 16](#_Toc112880841)

[4.3 Test 21](#_Toc112880842)

[4.3.1 Procedure 21](#_Toc112880843)

[4.3.2 Results 24](#_Toc112880844)

[4.4 Execution 24](#_Toc112880845)

[4.4.1 Terminal 25](#_Toc112880846)

[4.4.2 Results 28](#_Toc112880847)

[5 Summary 28](#_Toc112880848)

[6 References 28](#_Toc112880849)

[6.1 Schema 28](#_Toc112880850)

[6.1.1 Books 28](#_Toc112880851)

[6.1.2 Websites 29](#_Toc112880852)

[6.2 Sources 29](#_Toc112880853)

# Introduction

Although most of the code for this program has either already been started for us or can be directly leveraged from prior assignments. My confidence going into this assignment is not high. The concepts of object classes, new vocabulary, and general “abstract” nature of concepts I think will cause me to spend equal amounts of time between documentation and programming. I am glad to see that overall structure of these knowledge documents has been able to adapt to each week’s assignments. There is a benefit to getting “documentation” patterns set and revised as I think it helps how I approach and mentally frame the structure of the assignment.

[Table of Contents]

# My system information

I do not expect to include this information in every assignment. The purpose of including this information is to serve as a reference documentation for future needs (e.g., troubleshooting).

Where applicable, the information below will be updated after significant system component version updates, and the addition of new or replacement of existing system components (i.e., deltas from this information will be recorded).

[Table of Contents]

## Operating system (OS)

**macOS Monterey version 12.5.1**

*For more information, refer to A07-RSar.docx – Section 3.1.*

[Table of Contents]

## Console

**Terminal version 2.12.7 (445)**

*For more information, refer to A01-RSar.docx – Section 3.2.*

[Table of Contents]

## Shell

**zsh version 5.9**

*For more information, refer to A01-RSar.docx – Section 3.3.*

[Table of Contents]

## Python

**Python version 3.10.5**

*For more information, refer to A01-RSar.docx – Section 3.4.*

[Table of Contents]

## Integrated Development Environment (IDE)

Per instruction of Randal Root, using PyCharm Community Edition (CE) as default IDE for Module03. “Step 5.2 Create a new Project n PyCharm” (Randall, R. \_ Assignment03\_instructions.docx, Self-published, 2019).

**PyCharm version 2022.1.4 (Community Edition)**

*For more information, refer to A06-RSar.docx – Section 3.5.*

[Table of Contents]

## Directory / File path

Open Terminal

Navigate to Assignment08 working folder.

Enter the following command (see Figure 1):

|  |
| --- |
| cd documents/\_pythonclass/module08/a08rsarabia/introtoprog-python-mod08 |

Figure 1. Command in Terminal for navigating to assignment directory

This folder and its contents will be compressed into .zip file and submitted for Assignment06.

Enter the following command in Terminal to return the directory path (Figure 2):

|  |
| --- |
| pwd |

Figure 2. Command in Terminal to return directory path

Enter the following command in Terminal to return the directory content (see Figure 3):

|  |
| --- |
| ls -la |

Figure 3. Command in Terminal to return directory contents (visible and invisible files)

**/Users/rex/documents/\_PythonClass/module08/A08-RSarabia/IntroToProg-Python-Mod08** (Figure 4)

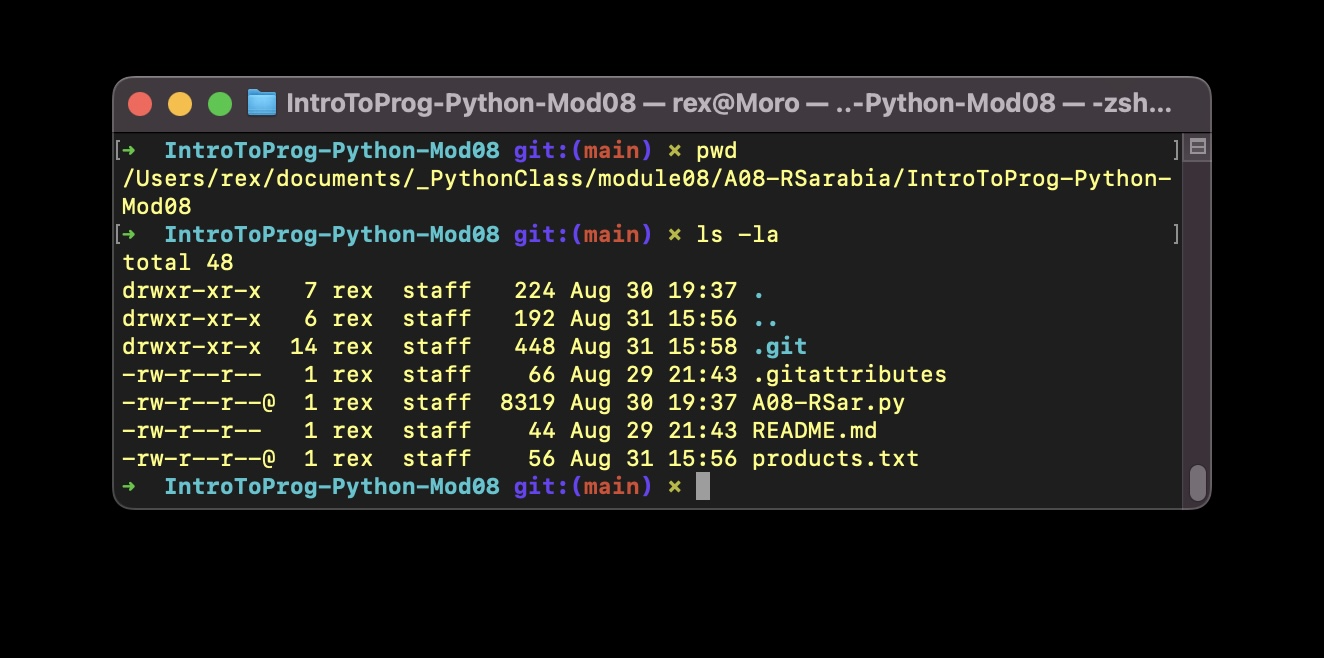


Figure 4. Screen capture of assignment directory and contents

Note: Since this assignment had specifically instructed us to use GitHub Desktop, the directory / folder above is the offline sync folder for this assignment’s GitHub repository. Hence, why the git:(main) appearing. Running the following command in Terminal informs us of the status of the repository (Figure 5).

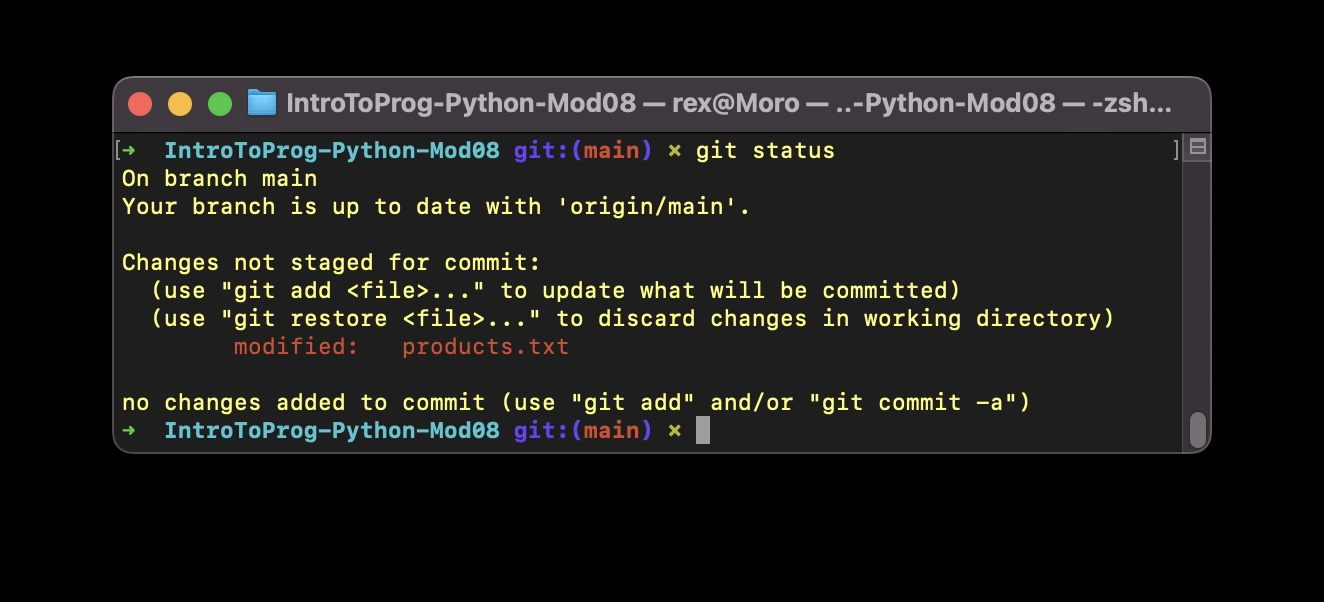


Figure 5. Screen capture of git status

[Table of Contents]

# Module assignment

## Requirements

“I have **provided starter** code in the file called, **"Assigment08-Starter.py"**. Currently, the code does nothing, but it does include pseudo-code.

**Read and understand the pseudo-code, then add code to make the application work**.

Make sure to **include error handling!**

Run the script in **BOTH** PyCharm and an OS command/shell window and capture images of it working on your computer.

" (Randall R., Assignment08\_instructions.docx, Self-published, 2022).

[Table of Contents]

### Out of scope

For the purposes of this document, documentation is limited to “Apply your knowledge” (Step 4) and “Document your knowledge” (Step 5) of the assignment. Additional assignment tasks related to “Watch some videos on GitHub” (Step 6) through “Create a GitHub webpage (optional)” (Step 8) are not included in this write-up.

[Table of Contents]

### Lessons learned

As weeks progress and assignments are graded, I thought it would be helpful to keep a log of comments received from prior assignments (Figure 6). The simplest benefit is to serve as a log of lessons learned along the way and more practically, as immediate reminders to not repeat the same mistakes. *Steel sharpens steel.*

| **Assignment filename** | **Comment** | **Penalties** |
| --- | --- | --- |
| A01-RSar.docx | end the file with  print(‘(Press Enter to End Program)’) #Conclusion line  input() | -2 |
| A02-RSar.docx | any figure# in a caption below a images/code snippet is referenced by figure # in the text written up above | -1 |
| A03-RSar.docx | Filename should be: HomeInventory.py | -2 |
| A04-RSar.docx | the save/exit option didn't ask me if i wanted to save to the file  Note: see Figure 5 - Assignment04\_instructions.docx | -2 |
| A05-RSar.docx | Needed to post a link to your GitHub site on the assignment textbox | -1 |
| A06-RSar.docx | for the zip file, only include the document file and the python file | -1 |
| A07-RSar.docx | need to insert comments near your pickling/exception program code | -1 |

Figure 6. Table of review comments from prior assignments

[Table of Contents]

## Design

This section includes the relevant components in the Python code that compose my proposed solution to the assignment.

[Table of Contents]

### Standard elements

#### Code style

By default, PyCharm will automatically perform checks of code style for PEP 8 (JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022) (External site). I made a conscious decision to follow the Problems/Solutions PyCharm identified, which in some cases conflict with the code style taught in class / prior assignments – honestly, it makes reading for errors in PyCharm a lot easier without having to mess with config settings of the feature.

For more information about PEP 8 code style guide: <https://peps.python.org/pep-0008/>

[Table of Contents]

#### Script header

In the script header (top of the script file), the following information must be recorded in the following format (Figure 7):

|  |
| --- |
| # ------------------------------- #  # Title: Assignment##  # Dev: RSar  # Desc: short description of script  # ChangeLog: (date,name,change)  # yyyy/mm/dd, Dev, Created File  # ------------------------------- # |

Figure 7. Source code for header

[Table of Contents]

#### Saving files

“PyCharm automatically saves changes that you make in your files. Saving is triggered by various events, such as compiling, running, debugging, performing version control operations, closing a file or a project, or quitting the IDE.” (JetBrains, <https://www>.jetbrains.com/help/pycharm/saving-and-reverting-changes.html, 2022) (External site)

Per assignment requirements (Section 4.1), script filename and data output filename as follows:

* Script filename: **A08-RSar.py**
* Data output filename: **products.txt**
* Directory / file path: Figure 4

[Table of Contents]

### Program description

Since this program is heavily dependent on the starter code file: **Assigment08-Starter.py**. My intention will be to build out the program as it appears to be intended (versus my own approach, i.e., adding \*not explicitly required\* features). Additionally, my focus be towards the “new” code covered in this module. Most, if not all, of the program’s functionality can be reused from prior assignments. Therefore, the focus of this write up will be on the scope of topics covered in the current module: classes with relevant coverage of try-except statements and other data validation mechanisms, as needed.

### Data structure

The data structure for this assignment is relatively simple in comparison to the prior assignment. However, since the focus of this assignment is setting up classes, it is helpful to identify relevant details from the assignment’s starter code file. From this pseudocode we can expect the following:

* Data will be read from and written to a text file named products
* Data read from the file and data added through the program will need to be combined together (lstOfProductObjects)
* Data will be of “products” with properties:
  + Name – string value data type
  + Price – float value data type

|  |
| --- |
| strFileName = ‘products.txt'  lstOfProductObjects = []  class Product:  """Stores data about a product:  properties:  product\_name: (string) with the products's name  product\_price: (float) with the products's standard price |

Figure 8. Data structure

Since I’ll be working within the structure of the starter code provided for this assignment, I’ll include the standard class pattern taught in this module’s lesson when programming the classes for this assignment (Figure 9).

|  |
| --- |
| # -- Fields --  # -- Constructor --  # -- Attributes --  # -- Properties --  # -- Methods -- |

Figure 9. Module template

[Table of Contents]

### Program architecture

Following lessons from Module06 (Randall R., \_Mod6PythonProgrammingNotes.docx, Self-published, 2019), efforts will be made to follow principles of abstraction and separation of concerns.

Based on the starter code provided for this assignment, the following can be expected for this program’s high-level design (Figure 10). The majority of the program’s code will reside within these three classes where the main body of the program will be minimal since its primary purpose will be to craft the narrative of the program by calling the necessary functions from these classes in a logical order.

Three classes:

1. Product
2. FileProcessor
3. IO

|  |
| --- |
| class Product:  class FileProcessor:  class IO: |

Figure 10. starter code for program's classes

With the exception of the Product class, the FileProcessor and IO classes are both “stripped” down version of functions that have already been programmed from prior assignments (Figure 11). Similar to prior assignments, the FileProcessor class will have two methods:

1. save\_d ata\_to\_file
2. read\_data\_from\_file where “list of product objects” is the lstOfProductObjects array

|  |
| --- |
| class FileProcessor:  methods:  save\_data\_to\_file(file\_name, list\_of\_product\_objects):  read\_data\_from\_file(file\_name): -> (a list of product objects) |

Figure 11. starter code for program's classes

Also similar to prior assignments, the IO class will have four methods (Figure 12):

1. output\_menu\_tasks
2. input\_menu\_choice
3. show\_current\_data
4. add\_new\_product

|  |
| --- |
| class IO:  # TODO: Add code to show menu to user  def output\_menu\_tasks()  pass  # TODO: Add code to get user's choice  def input\_menu\_choice()  pass  # TODO: Add code to show the current data from the file to user  def show\_current\_data ()  pass  # TODO: Add code to get product data from user  def add\_new\_product ()  pass |

Figure 12. starter code for program's classes

The main body of the script will be minimal to provide to serve sole purpose of the program “skeleton” acts as a “program interface” or “intermediary / broker” between the class functions and the user (Figure 13).

|  |
| --- |
| # Main Body of Script ---------------------------------------------------- #  # Load data from file into a list of product objects when script starts  # Show user a menu of options  # Get user's menu option choice  # Show user current data in the list of product objects  # Let user add data to the list of product objects  # let user save current data to file and exit program  # Main Body of Script ---------------------------------------------------- # |

Figure 13. starter code for program's classes

#### Product class

*Requirement 1: Create Product class with properties of name and price.*

Leveraged reference code provided in module lesson:

1. Lab8-4.py
2. Listing04.py

##### Constructors

The constructor function of the Product class purpose is to initialize an object with the \_\_init\_\_ function by default in Python. This function determines what attributes that object should be required to be created with when the object of the class is initialized.

In this program, we are creating Product objects with product\_name and product\_price arguments that will be passed to “virtual” attributes: self.\_\_product\_name and self.\_\_product\_price (Figure 14).

Note: No decorator (e.g., @staticmethod) before the \_\_init\_\_ function because it is a private method, i.e., the attribute is hidden.

|  |
| --- |
| class Product:# -- Fields --  # -- Constructor --  def \_\_init\_\_(self**,** product\_name**,** product\_price):  # -- Attributes --  self.\_\_product\_name = product\_name  self.\_\_product\_price = product\_price  # -- Properties --  # -- Methods -- |

Figure 14. Source code for constructors of the Product class

##### Properties

The Properties functions manage the attribute data of objects in the Product class. For this program, I created two properties (getter and setter) for each attribute (Figure 15). The setter determines properties for the attribute.

For example, when the product\_name attribute is specified then the product\_name() function is called. In this program, validation logic is included to reject the value if all characters in the string are numbers. If the value does not contain any letters, then custom Exception can be called: raise Exception("Product name must contain a letter.").

The getter statement determines what is returned when the attribute of the object is used. In this program, the product\_name attribute is formatted as a string data type with Title Casing: return str(self.\_\_product\_name).title().

|  |
| --- |
| class Product:# -- Fields --  # -- Constructor -  # -- Properties –  # product\_name  @property  def product\_name(self): # getter  return str(self.\_\_product\_name).title() # Title case   @product\_name.setter  def product\_name(self**,** value): # setter  if not str(value).isnumeric():  self.\_\_product\_name = value  else:  raise Exception("Product name must contain a letter.")   # product\_price  @property  def product\_price(self): # getter  return float(self.\_\_product\_price)   @product\_price.setter  def product\_price(self**,** value): # setter  if float(value) and value >= **0**:  self.\_\_product\_price = value  elif not str(value).isnumeric() or float(value) < **0**:  raise Exception("Price must be a number and cannot be negative.")  # -- Methods - |

Figure 15. Source code for Properties of Product class

Note: The validation logic applied to product\_price attribute is that value must be able to convert to float data type and that the value must be greater than or equal to zero, i.e., price must be a number and price cannot be a negative number: if float(value) and value >= 0. Afterall, that would be absurd! The business pays the customer?!

This function appears in in the program in the following functions are called:

1. add\_new\_product()

##### Methods

Although not specifically required for the program to work, the to\_string and \_\_str\_\_ functions allow additional customization in setting a “default formatting” of the object (Figure 16). In this program, the defeault output when an object of the Product class is used generates a string value: return self.product\_name + "\t $" + '{0:.2f}'.format(self.product\_price).

For this program, I decided to format the string value of item object:

For example: “Item500 $500.50”

This function appears in in the program in the following functions are called:

1. show\_current\_data()
2. add\_new\_product()

|  |
| --- |
| class Product:  # *TODO: Add Code to the Product class* # -- Fields --  # -- Constructor --  # -- Properties -  # -- Methods --def to\_string(self):  return self.\_\_str\_\_()  def \_\_str\_\_(self):  return self.product\_name + "\t $" +'{0:.2f}'.format(self.product\_price) |

Figure 16. Source code for the methods of the Product class

[Table of Contents]

#### FileProcessor class

*Requirement 2: Create FileProcessor class with methods to read the data from the file, add data input by the user to a list, and save data to file.*

The majority of code base of the main body was adapted or directly reused from prior assignments (Figure 17).

|  |
| --- |
| # -- Methods -- # *TODO: Add Code to process data from a file*  @staticmethod def read\_data\_from\_file(file\_name**,** list\_of\_rows):  with open(file\_name**,** "r") as file:  for line in file:  data = line.split(",")  row = Product(product\_name=data[**0**].strip()**,** product\_price=data[**1**].strip())  list\_of\_rows.append(row)  file.close()  return list\_of\_rows  @staticmethod def add\_data\_to\_list(name**,** price**,** list\_of\_rows):  row = (str(name).strip()**,** str(price).strip()**,** "\n")  list\_of\_rows.append(row)  return list\_of\_rows  *# TODO: Add Code to process data to a file*  @staticmethod def save\_data\_to\_file(file\_name**,** list\_of\_rows):  with open(file\_name**,** "w") as file:  for item in list\_of\_rows:  file.write(str(item.product\_name) + "," +  str(item.product\_price) + "\n")  file.close()  print("\tData saved to file: '" + file\_name + "'") |

Figure 17. Source code for methods of the FileProcessor class

[Table of Contents]

#### IO class

*Requirement 4: Create IO class with methods to display the menu options to the user, allow the user to select an option from the menu, display the current list of product data (data from file + new data added), and allow the user to add a new item to the product list.*

Encountered an unexpected behavior in program code – details submitted to course module discussion board: <https://canvas.uw.edu/courses/1595783/discussion_topics/7243548>.

A quick note on the unexpected behavior in program code. During the module09 lecture, Randal Root addressed the topic I had raised in the module08 Discussion Board.

My understanding of this behavior: since the “validation” logic for the product\_name and product\_price is not located in the initialization portion of the class and in combination with standard class structure convention, it is necessary to include code (perhaps not the most elegant solution, but it does work) to initialize an item object with empty values for product\_name and product\_price properties (Figure 18). Then, when the user inputs the name and price values, the validation in the product\_name setter and product\_price setter functions will run the user input for each (one at a time, in series) through the respective setter functions.

|  |
| --- |
| class IO:  # *TODO: Add docstring*  # -- Fields --  # -- Constructor --  # -- Attributes --  # -- Properties --  # -- Methods –  # *TODO: Add code to show menu to user* @staticmethod   # *TODO: Add code to get user's choice* @staticmethod   # *TODO: Add code to show the current data from the file to user* @staticmethod   # *TODO: Add code to get product data from user* @staticmethod  def add\_new\_product():  item = Product(''**, 0**) # instantiates 'null' item object  while True:  try:  break  except:  continue |

Figure 18. Source code for function add\_new\_product()

The remainder of the code base for the methods of the IO class was adapted or directly reused from prior assignments (Figure 19).

|  |
| --- |
| class IO:  # *TODO: Add docstring* # -- Fields --  # -- Constructor --  # -- Attributes --  # -- Properties --  # -- Methods –  # *TODO: Add code to show menu to user*  @staticmethod  def output\_menu\_tasks():  *""" Display a menu of choice to the user* ***:return****: nothing  """* print("\n\t" + "="\***9** + " Menu of options " + "="\***9** +  "\n\t1 - Show current list of products"  "\n\t2 - Add product"  "\n\t3 - Save and exit" +  "\n\t" + "="\***35**)   # *TODO: Add code to get user's choice*  @staticmethod  def input\_menu\_choice():  choice = str(input("\nSelect an option from the menu "  "(1-3): ")).strip()  return choice   # *TODO: Add code to show the current data from the file to user*  @staticmethod  def show\_current\_data(list\_of\_products):  print("\t-------- List of Products --------")  for item in list\_of\_products:  print("\t" + str(item))  print("\t" + "-"\***35**)   # *TODO: Add code to get product data from user*  @staticmethod  def add\_new\_product():  item = Product(''**, 0**) # instantiates 'null' item object  while True:  name = input("Enter product NAME: \t")  try:  item.product\_name = name  break  except Exception as e:  print("\n\t" + errorCode + str(e) + "\n")  continue   while True:  price = input("\nEnter product PRICE: \t")  try:  item.product\_price = float(price)  break  except Exception as e:  print("\n\t" + errorCode + str(e))  print()  print("\tNew product added: \t" + str(item))  return item |

Figure 19. Source code for methods of the IO class

[Table of Contents]

#### Main body

The majority of code base of the main body was adapted or directly reused from prior assignments (Figure 20).

|  |
| --- |
| # Data -------------------------------------------------------------------- # strFileName = "products.txt" lstOfProductObjects = [] errorCode = "Error. Error. : "  # Main Body of Script ---------------------------------------------------- # # 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.output\_menu\_tasks()   # Get user's menu option choice  choice\_str = IO.input\_menu\_choice()   # Show user current data in the list of product objects  if choice\_str.strip() == '1': # show current data  print()  print("\tDisplaying current list of products...")  print()  IO.show\_current\_data(lstOfProductObjects)  continue   # Let user add data to the list of product objects  elif choice\_str.strip() == '2': # add product  print()  print("\tAdd product information...")  print()  lstOfProductObjects.append(IO.add\_new\_product())  continue   # let user save current data to file and exit program  elif choice\_str.strip() == '3':  # ask user to save and exit  while True:  choice = input("\nSave changes to file? [Y/N]: \t")  if choice.lower() == "y":  print()  print("\tSaving data to file...")  print()  FileProcessor.save\_data\_to\_file(strFileName**,** lstOfProductObjects)  break  elif choice.lower() == "n":  print()  print("\tData not saved to file...")  break  else:  print("\n\t" + errorCode + ": Invalid choice.")  # continue  input("\nPress ENTER key to quit the program.")  break |

Figure 20. Source code for main body of script

A quick note on a new addition: try-except exception statement (Figure 21). This statement was added to for exception handling if the file specified when variable strFileName was defined. If the file is not found (e.g., filename is incorrect, file is in wrong directory), then the program will throw the built-in exception string value of the FileNotFoundError. Under the else statement if file is found, then program effectively starts by reading the file and populating object lstOfProductObjects with the data from the file.

|  |
| --- |
| # Data -------------------------------------------------------------------- # strFileName = "products.txt" lstOfProductObjects = [] errorCode = "Error. Error. : "  # Main Body of Script ---------------------------------------------------- # # Load data from file into a list of product objects when script starts try:  file = open(strFileName) except FileNotFoundError as e:  print("\n\t" + errorCode + e.\_\_str\_\_()) except Exception as e:  print(e) else:  lstOfProductObjects = FileProcessor.\  read\_data\_from\_file(strFileName**,** lstOfProductObjects) |

Figure 21. Source code for try-except exception statement

[Table of Contents]

### Proposed solution

The following source code is my program for Assignment08 (Figure 22).

|  |
| --- |
| # ------------------------------- # # Title: Assignment08 # Dev: RSar # Desc: Working with classes # ChangeLog: (date,name,change) # 2022/01/01, RRoot, Created started script # 2022/01/02, RRoot, Added pseudocode to start assignment 8 # 2022/08/28, RSar, Modified code to complete assignment 8 # ------------------------------- #   # Data -------------------------------------------------------------------- # strFileName = "products.txt" lstOfProductObjects = [] errorCode = "Error. Error. : "  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:  changelog: (date,name,change)  2022/01/01, RRoot, Created Class  2022/08/28, RSar, Modified code to complete assignment 8  """* # -- Fields --  # -- Constructor --  def \_\_init\_\_(self**,** product\_name**,** product\_price):  # -- Attributes --  self.\_\_product\_name = product\_name  self.\_\_product\_price = product\_price   # -- Properties --  # product\_name  @property  def product\_name(self): # getter  return str(self.\_\_product\_name).title() # Title case   @product\_name.setter  def product\_name(self**,** value): # setter  if not str(value).isnumeric():  self.\_\_product\_name = value  else:  raise Exception("Product name must contain a letter.")   # product\_price  @property  def product\_price(self): # getter  return float(self.\_\_product\_price)   @product\_price.setter  def product\_price(self**,** value): # setter  if float(value) and value >= **0**:  self.\_\_product\_price = value  elif not str(value).isnumeric() or float(value) < **0**:  raise Exception("Price must be a number and cannot be negative.")   # -- Methods --  # *TODO: Add Code to the Product class* def to\_string(self):  return self.\_\_str\_\_()   def \_\_str\_\_(self):  # return self.product\_name + "," + str(self.product\_price)  return self.product\_name + "\t $" + '{0:.2f}'.format(self.product\_price)  # Data -------------------------------------------------------------------- #  # Processing ------------------------------------------------------------- # 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):   add\_data\_to\_list(name, price, list\_of\_rows):   read\_data\_from\_file(file\_name): -> (a list of product objects)   changelog: (date,name,change)  2022/01/01, RRoot, Created Class  2022/08/28, RSar, Modified code to complete assignment 8  """* # -- Fields --  # -- Constructor --  # -- Attributes --  # -- Properties --  # -- Methods --  # *TODO: Add Code to process data from a file* @staticmethod  def read\_data\_from\_file(file\_name**,** list\_of\_rows):  with open(file\_name**,** "r") as file:  for line in file:  data = line.split(",")  row = Product(product\_name=data[**0**].strip()**,** product\_price=data[**1**].strip())  list\_of\_rows.append(row)  file.close()  return list\_of\_rows   @staticmethod  def add\_data\_to\_list(name**,** price**,** list\_of\_rows):  row = (str(name).strip()**,** str(price).strip()**,** "\n")  list\_of\_rows.append(row)  return list\_of\_rows   # *TODO: Add Code to process data to a file* @staticmethod  def save\_data\_to\_file(file\_name**,** list\_of\_rows):  with open(file\_name**,** "w") as file:  for item in list\_of\_rows:  file.write(str(item.product\_name) + "," +  str(item.product\_price) + "\n")  file.close()  print("\tData saved to file: '" + file\_name + "'") # Processing ------------------------------------------------------------- #  # Presentation (Input/Output) -------------------------------------------- # class IO:  # *TODO: Add docstring  """Performs Input and Output tasks   methods:  output\_menu\_tasks():   changelog: (date,name,change)  2022/08/28, RSar, Created Class  """* # -- Fields --  # -- Constructor --  # -- Attributes --  # -- Properties --  # -- Methods --  # *TODO: Add code to show menu to user* @staticmethod  def output\_menu\_tasks():  *""" Display a menu of choice to the user* ***:return****: nothing  """* print("\n\t" + "="\***9** + " Menu of options " + "="\***9** +  "\n\t1 - Show current list of products"  "\n\t2 - Add product"  "\n\t3 - Save and exit" +  "\n\t" + "="\***35**)   # *TODO: Add code to get user's choice* @staticmethod  def input\_menu\_choice():  choice = str(input("\nSelect an option from the menu "  "(1-3): ")).strip()  return choice   # *TODO: Add code to show the current data from the file to user* @staticmethod  def show\_current\_data(list\_of\_products):  print("\t-------- List of Products --------")  for item in list\_of\_products:  print("\t" + str(item))  print("\t" + "-"\***35**)   # *TODO: Add code to get product data from user* @staticmethod  def add\_new\_product():  item = Product(''**, 0**) # instantiates 'null' item object  while True:  name = input("Enter product NAME: \t")  try:  item.product\_name = name  break  except Exception as e:  print("\n\t" + errorCode + str(e) + "\n")  continue   while True:  price = input("\nEnter product PRICE: \t")  try:  item.product\_price = float(price)  break  except Exception as e:  print("\n\t" + errorCode + str(e))  print()  print("\tNew product added: \t" + str(item))  return item  # Presentation (Input/Output) -------------------------------------------- #   # Main Body of Script ---------------------------------------------------- # # Load data from file into a list of product objects when script starts try:  file = open(strFileName) except FileNotFoundError as e:  print("\n\t" + errorCode + e.\_\_str\_\_()) except Exception as e:  print(e) else:  lstOfProductObjects = FileProcessor.\  read\_data\_from\_file(strFileName**,** lstOfProductObjects)  while True:  # Show user a menu of options  IO.output\_menu\_tasks()   # Get user's menu option choice  choice\_str = IO.input\_menu\_choice()   # Show user current data in the list of product objects  if choice\_str.strip() == '1': # show current data  print()  print("\tDisplaying current list of products...")  print()  IO.show\_current\_data(lstOfProductObjects)  continue   # Let user add data to the list of product objects  elif choice\_str.strip() == '2': # add product  print()  print("\tAdd product information...")  print()  lstOfProductObjects.append(IO.add\_new\_product())  continue   # let user save current data to file and exit program  elif choice\_str.strip() == '3':  # ask user to save and exit  while True:  choice = input("\nSave changes to file? [Y/N]: \t")  if choice.lower() == "y":  print()  print("\tSaving data to file...")  print()  FileProcessor.save\_data\_to\_file(strFileName**,** lstOfProductObjects)  break  elif choice.lower() == "n":  print()  print("\tData not saved to file...")  break  else:  print("\n\t" + errorCode + ": Invalid choice.")  # continue  input("\nPress ENTER key to quit the program.")  break |

Figure 22. Source code for my proposed solution to Assignment08

[Table of Contents]

## Test

### Procedure

For the purpose of this assignment, testing is performed in PyCharm IDE.

Open PyCharm

PyCharm > File > Open > directory path (Section 3.6) > **A08-RSar.py**

PyCharm > Run > **A08-RSar**

For this assignment and based on the limitations placed on the inputs of the user, I intend to limit my test cases to key deltas from prior assignment and a single happy user journey. Namely exception handling (Figure 16).

| **Test ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
| 1 | Exception handling – File not found | *Incorrect file name*  strFileName = “\_products.txt” | Pass |
| 2 | Exception handling – Invalid product name | *Add data*  Name = 123 | Pass |
| 3 | Exception handling – Invalid product price | *Add data*  Price = -5 | Pass |
| 4 | Happy journey flow | *Open file*    *Show current list of products*    *Add valid item to products list*  Name = NewItem1  Price = 5    *Show current list of products*    *Save data to file* | Pass |

Figure 23. Summary of tests performed and results in PyCharm IDE

[Table of Contents]

### Results

Results from test cases all passed as the actual result matched what I was expecting for each of the value combinations input.

[Table of Contents]

## Execution

For the purpose of this assignment, execution is done via the Terminal which simulates executing in Production environment (PROD) as running the program in PyCharm IDE simulates testing environment (TEST).

[Table of Contents]

### Terminal

Open Terminal

Enter the following command (Figure 17):

|  |
| --- |
| python3 [file path][file name] |

Figure 23. Command for executing script in Terminal

Where:

File path: Figure 5

File name: Section 4.2.1.3

Re-perform test procedures (see Figure 18). For this assignment, the data inputted into default data file has been erased and so the file is null.

| **Test ID** | **Test description** | **Actual output** | **Result** |
| --- | --- | --- | --- |
| 1 | Exception handling – File not found | *Incorrect file name*  strFileName = “\_products.txt” | Pass |
| 2 | Exception handling – Invalid product name | *Add data*  Name = 123 | Pass |
| 3 | Exception handling – Invalid product price | *Add data*  Price = -5 | Pass |
| 4 | Happy journey flow | *Open file*    *Show current list of products*    File: products.txt    *Add valid item to products list*  Name = NewItem1  Price = 5    *Show current list of products*    *Save data to file*    File contents: products.txt | Pass |

Figure 24. Summary of tests performed and results in Terminal

[Table of Contents]

### Results

Using the same input values per Section 4.3.1, produced the same results as testing for both error messages of invalid input as well as correctly formatted and stored outputs of valid user inputs.

[Table of Contents]

# Summary

Beginning this course eight weeks ago already seems like an eternity. This assignment was a good culmination of prior assignments completed to date. Personally, I do not think I was able to keep up with the intended pacing, but glad to see that my decisions on how to design, execute, and document each modules’ assignments seems to have paid off. Admittedly, I spent more time than I thought I would each week learning about the current and maintaining consistent documentation week-to-week.

So, in that regard, I am relieved that this is the final assignment, as I do not think I could continue this level of energy and motivation on subsequent assignments as I had in earlier weeks. This is especially true as my work is picking up after a seasonal lull during August. Overall, I enjoyed this course and I appreciate the effort and time that David Blodgett and Randal Root put into making programming and Python approachable. Although I may not have mastered the coursework, I do understand it better, and have learned how to find and distill information on the internet better.

[Table of Contents]

# References

## Schema

### Books

“Quoted text“ (Author Last name Author First name initial., Title of book, Publisher, Year published)

### Websites

“Quoted text“ (Website name, URL, yearAccessed) (External site)

[Table of Contents]

## Sources

*Note: It is assumed that knowledge builds, therefore, duplicate sources already included in prior assignments have been removed – unless it has been directly referenced within this assignment.*

JetBrains, <https://www.jetbrains.com/help/pycharm/saving-and-reverting-changes.html>, 2022 (External site)

JetBrains, <https://www.jetbrains.com/help/pycharm/tutorial-code-quality-assistance-tips-and-tricks.html#df2e3bcf>, 2022 (External site)

Randall R., \_Mod8PythonProgrammingNotes.docx, Self-published, 2019

Randall, R. \_ Assignment08\_instructions.docx, Self-published, 2019

Python Guides, <https://pythonguides.com/python-print-2-decimal-places/>, 2022 (External site): Python print 2 decimal places

Stack Overflow, <https://stackoverflow.com/questions/59190175/how-to-create-an-object-from-a-list-of-lists>, 2022 (External site): How to create an object from a list of lists?

Flexiple, <https://flexiple.com/python/list-of-lists-python/>, 2022 (External site): How to create a list of lists in Python?

Career Karma, <https://careerkarma.com/blog/python-typeerror-method-object-is-not-subscriptable/?v=2>, 2022 (External site): Python TypeError: ‘method’ object is not suscriptable solution

Youtube – Corey Schafer, various (see below), 2022 (External site): Python OOP Tutorial #

* Python OOP 1 - Classes and Instances - <https://youtu.be/ZDa-Z5JzLYM>
* Python OOP 2 - Class Variables - <https://youtu.be/BJ-VvGyQxho>
* Python OOP 3 - Classmethods and Staticmethods - <https://youtu.be/rq8cL2XMM5M>
* Python OOP 4 - Inheritance - <https://youtu.be/RSl87lqOXDE>
* Python OOP 5 - Special (Magic/Dunder) Methods - <https://youtu.be/3ohzBxoFHAY>
* Python OOP 6 - Property Decorators - <https://youtu.be/jCzT9XFZ5bw>

[Table of Contents]