Using Objects and Classes

**Introduction and Script**

The main point of the script was to use a class with constructors and properties with a bit of error handling to create a product menu for the user. Most of that is near the top of the script so I’ll go over that more as the rest of the script is mostly taken from previous assignments.

**Code Without Header**

# Data -------------------------------------------------------------------- #

strFileName = 'products.txt'

lstOfProductObjects = []

class Product:

# -- Field --

strProductName = ""

# -- Constructor --

def \_\_init\_\_(self, product\_name: str, product\_price: float):

# -- Attributes --

self.\_\_product\_name = product\_name

self.\_\_product\_price = product\_price

# -- Properties --

# Product Name = !ONLY CHANGES THE CASE!

@property

def product\_name(self): # Getter

return str(self.\_\_product\_name).title() # Title Case

@property

def product\_price(self): # Getter with Error Handling

try:

return float(self.\_\_product\_price) # No Need for Title

except:

raise Exception("! - Error - ! Letter's aren't numerical prices.")

@product\_price.setter

def product\_price(self, value):

if float(value).isnumeric() == True:

self.\_\_product\_price = value

else:

raise Exception("Letter's aren't prices.")

# Data -------------------------------------------------------------------- #

# Processing ------------------------------------------------------------- #

class FileProcessor:

#Methods

# Saves current list to the text file

@staticmethod

def save\_data\_to\_file():

objFile = open("products.txt", "a")

for row in lstOfProductObjects:

objFile.write(row["Product"] + "," + row["Price"] + "\n")

objFile.close()

print("===== The current table has been saved =====\n")

# Creates a text file if none exists

@staticmethod

def create\_file\_for\_data (file\_name):

try:

objFile = open(file\_name,"x")

objFile.close()

except:

print("File Loaded Up - Only Shows if No File")

# Loads up data in text file

@staticmethod

def read\_data\_from\_file(file\_name, lstOfProductObjects):

objFile = open(file\_name,"r")

for row in objFile:

txtdata = row.split(",")

dicRow = {"Product": txtdata[0], "Price": txtdata[1].strip()}

lstOfProductObjects.append(dicRow)

objFile.close()

return lstOfProductObjects

# Displays current data in list

@staticmethod

def print\_current\_data\_in\_list(lstOfProductObjects):

print("--------------------------------------------\n"

"\tThis is the current data in the table.\n"

"--------------------------------------------\n"

"==============")

print("Row - Product - Price")

counter = 0

for row in lstOfProductObjects:

print(f'{counter} | {row["Product"]} | {row["Price"]} |')

counter += 1

print("==============")

return lstOfProductObjects

# Adds Data to Primary List

@staticmethod

def add\_data\_to\_list(product, price, lstOfProductObjects):

dicRow = {"Product": product, "Price": price}

lstOfProductObjects.append(dicRow)

print("\n=====", product, "has been added with priority", price, "=====")

# Presentation (Input/Output) -------------------------------------------- #

class IO:

# Greetings

@staticmethod

def greetings():

print("==============================================\n"

"\t Hello amazing user!\n"

"\t This script is made to allow you to\n"

"\t create a list showing a product and\n"

"\t the price that you put in. The table \n"

"\t Will always be displayed with the menu.\n\n"

"\t First it will open the file to keep\n"

"\t previously existing data then you the\n"

"\t user may interact with that data.\n\n"

"\t !IMPORTANT!\n"

"\t DATA IS ONLY SAVED WHEN THE OPTION\n"

"\t IS CHOSEN IN THE MENU!\n"

"==============================================")

# Menu

@staticmethod

def print\_menu\_tasks():

print("""

\*\*\* Menu of Options \*\*\*

1) Add a new Product

2) Save Data to File

3) Exit Program

""")

# Menu Choice

def input\_menu\_choice(self):

choice = str(input("Which option would you like to perform? [1 to 5] - "))

print()

return choice

# Continue Option

@staticmethod

def input\_press\_to\_continue(optional\_message=''):

print(optional\_message)

input('Press the [Enter] key to continue.')

# Yes or No Input

@staticmethod

def input\_yes\_no\_choice(message):

return str(input(message)).strip().lower()

# Main Body of Script

FileProcessor.create\_file\_for\_data("products.txt")

FileProcessor.read\_data\_from\_file("products.txt",lstOfProductObjects)

IO.greetings()

while True:

FileProcessor.print\_current\_data\_in\_list(lstOfProductObjects)

IO.print\_menu\_tasks()

strChoice = IO.input\_menu\_choice("self")

if strChoice.strip() == '1':

getinfo = Product(input("-------------------------------------------------\n"

"\tYou've chosen to add to the current table.\n"

"\tWhat product do you wish to add?\n"

"-------------------------------------------------\n"

"\tEnter New Product: "),

input("\n---------------------------------------------------------------\n"

"\tWhat price do you wish to designate to this task?\n"

"\tExamples: 1.99.\n"

"---------------------------------------------------------------\n"

"\tEnter Price for Product: "))

# User data goes through the constructor and properties

pdct1 = getinfo.product\_name

price1 = str(getinfo.product\_price)

FileProcessor.add\_data\_to\_list(pdct1,price1,lstOfProductObjects)

elif strChoice.strip() == '2':

strChoice = IO.input\_yes\_no\_choice("Save this data to file? (y/n) - ")

if strChoice.lower() == 'y':

FileProcessor.save\_data\_to\_file()

else:

IO.input\_press\_to\_continue("Save Cancelled!")

elif strChoice == '3': # Exit Program

print("Goodbye!")

break # and Exit

**Part 1**

First the script checks if a file named “products” already exists. If not it creates one then reads the created file, which will be black. It’s the first time I’ve used this tool in Python, I found it rather useful. After that all the classes and methods are loaded up ready to be used in the main script.

**Part 2**

The class “Product” takes the users input and edits it a bit for the primary list. It saves the product name as a string and the price as a float. The float is later changed to a string for ease in saving to the primary list. The product name will be capitalized, while the price is checked to make sure that it is only a number.

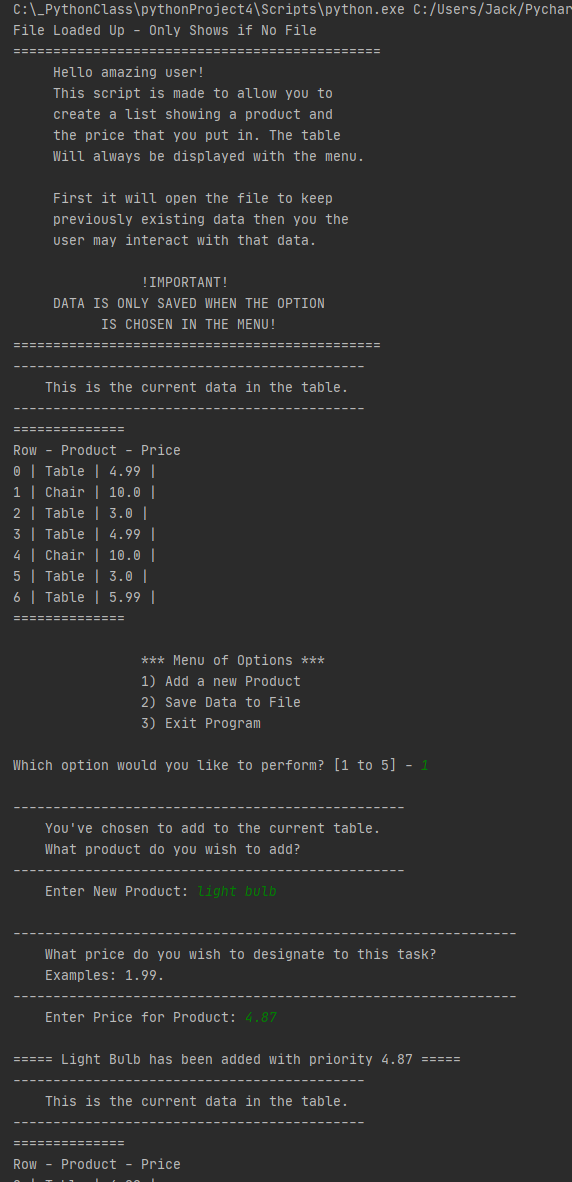
**Part 3**

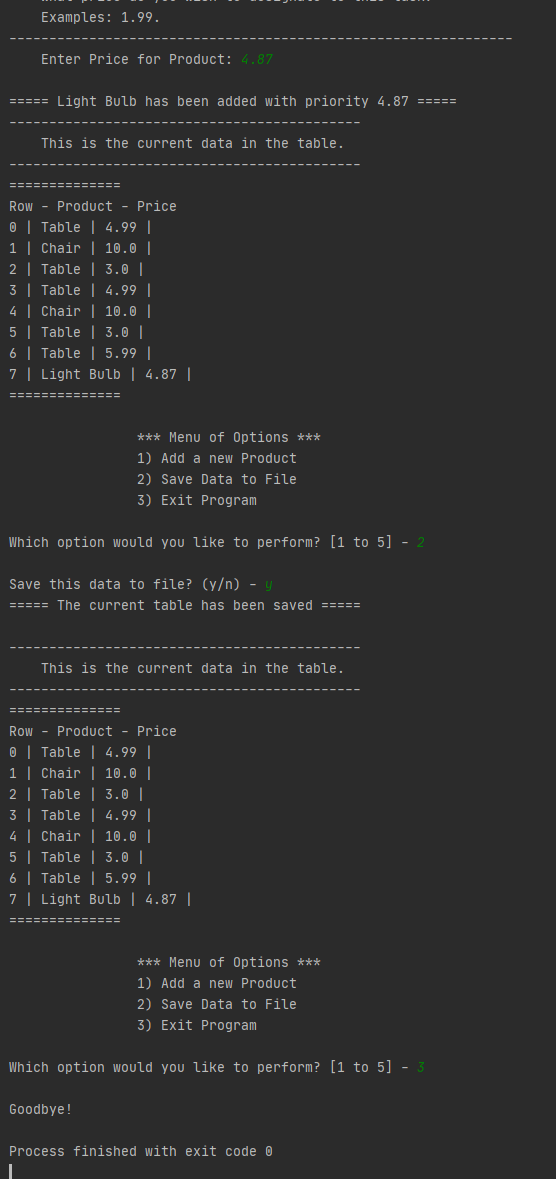
Having already made functions from previous projects I copied them in and changed the names to them. They each have commented out descriptions for their purpose.

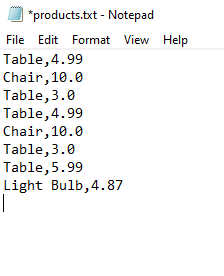
**Conclusion**

Using classes and objects is pretty handy in controlling how the data should be saved and displayed while capturing errors. It also allows for the abstract to work away from the frontend data.

**Proof**

****

****

****