Name: Charles Lin
Date: March 5, 2023
Course: IT FDN 110 A
GitHubURL:

Assignment 08 Classes and Objects

Introduction

I learned about classes that organize functions and data. Functions in classes are called methods, and variables/constants in classes are called fields, attributes, or properties. Several on-line introductions like to use the car analogy. The class ‘Car’ contains ‘properties’ like brand, model, color, fuel, and so on. It also holds behavior, or ‘methods’ like start(), halt(), drift(), speedup(), and turn().

Class

We organize functions into classes when we want to use them repeatedly, similar to how we organize statements into functions. Classes are generally designed to be data, processing, or interaction (input/output). We can use the class’s code directly, or we can create an object instance and use the class indirectly.

A class often contains Constructor, Properties, and Methods. Constructors are special methods (functions) that automatically run when we create an object from a class. Constructors are sometimes called an initializer because it sets initial data values, named "__init__". Properties generally have a getter and a setter. A getter lets us access and format the data, and a setter lets us add validation code. Methods let us organize statements into named groups, and are part of a class's code.

Assignment on Working with Class

The starter script outlined Data, Processing, Presentation(input/output), and Main Body of Script portions. It’s a continuation of the past assignments.

Data

I first set the variables.

```
# Data ------------------------------------------------------------------- #
file_name_str = 'products.txt' # The name of the data file
file_obj = None  # An object that represents a file
table_lst = [] # A list that acts as a 'table' of rows
choice_str = "" # Captures the user option selection
```

Listing 1 Set the Variables

I created a “Product” class that contains a “constructor” section that runs when I create an object from a class.

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

Listing 2 Constructor that uses parameters to capture the initial values

The “Product” class also contains a “property” section for special functions that manage attribute data, and returns exceptions if data is not as expected, for example, not texts or numbers.

```
# --Properties--
@property
def product_name(self): # getter
    return str(self.__product_name).title() # Format attribute as Title case

@product_name.setter
def product_name(self, value): # setter
    if str(value).isnumeric() == False:
        self.__product_name = value
    else:
        raise Exception ("Product names cannot be numbers")

@property
def product_price(self): # getter
    return str(self.__product_price)

@product_price.setter
def product_price(self, value): # setter
    if str(value).isnumeric() == True:
        self.__product_price = value
    else:
        raise Exception ("Product price has to be numbers")
```

Listing 3 Property that gets and sets product name and price along with exceptions

The “method” section returns the class’s data as a string.

```
def to_string(self):
    """  Returns object data in a comma separated string of values

    ;return; (string) CSV data
    """
    object_data_csv = self.product_name + ',' + self.product_price
    return object_data_csv
```

Listing 4 Method that contains functions inside the class

To look at what the class “product” returns:

```
# Look at what the class does
objP1 = Product("Desk", "99")
print(objP1.to_string())
```

Listing 5 What the class does when passing product name “desk” and price “99”

```
/Users/charles.lin/PycharmProjects/
Desk,99
```

Figure 1 Output from running Listing 5

Processing

I created a “Processor” class that contains methods that read data, add to list, and save data to a file.

```
@staticmethod
def read_data_from_file(file_name, list_of_rows):
    """ Reads data from a file into a list of dictionary rows

    :param file_name: (string) with name of file:
    :param list_of_rows: (list) you want filled with file data:
    :return: (list) of dictionary rows
    """
    list_of_rows.clear()  # clear current data
    try:    # try/except to ask folks to place a starting product list file if it doesn't exist
        file = open(file_name, "r")
        for line in file:
            task, priority = line.split(",")
            row = {"product_name": task.strip(), "product_price": priority.strip()}
            list_of_rows.append(row)
        file.close()
        return list_of_rows
    except:
        print()
        print("Such file doesn't exist. Please place product.txt file in the working folder.")
        print()
```

Listing 6 Reads data and includes try/except if file doesn’t exist

```python
@staticmethod
def add_data_to_list(product_name, product_price, list_of_rows):
    """ Adds data to a list of dictionary rows

    :param product_name: (string) with name of product:
    :param product_price: (int) with price of product:
    :param list_of_rows: (list) you want to add more data to:
    :return: (list) of dictionary rows
    """
    row = {"product_name": str(product_name).strip(), "product_price": int(product_price)}
    list_of_rows.append({"product_name": product_name, "product_price": product_price})
    return list_of_rows
```

Listing 7 Add data to list

```python
@staticmethod
def write_data_to_file(file_name, list_of_rows):
    """ Writes data from a list of dictionary rows 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 dictionary rows
    """
    objFile = open(file_name_str, "w")
    for row in list_of_rows:
        objFile.write(str(row["product_name"]) + ',' + str(row["product_price"] + '\n'))
    objFile.close()
    return list_of_rows
```

Listing 8 Save data to file

Presentation(input/output)

I created a “Presentation” class that contains methods that display a menu of choices, get user’s choice, show the current list, and add a product to list. I use these methods directly from the class, so I use @staticmethod decorator.

```python
@staticmethod
def output_menu_products():
    """  Display a menu of choices to the user

    :return: nothing
    """
    print('''
    Menu of Options
    1) Show Current Data
    2) Add a New Item
    3) Save Data to File
    4) Exit Program
    ''')
    print()  # Add an extra line for looks in the terminal window
```

Listing 9 Display a menu of choices

```python
@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
```

Listing 10 Get user's choice

```python
@staticmethod
def output_current_products_in_list(list_of_rows):
    """ Shows the current Tasks in the list of dictionaries rows

    :param list_of_rows: (list) of rows you want to display
    :return: nothing
    """
    print("******* The current products are: *******")
    for row in list_of_rows:
        print(row["product_name"] + " (" + row["product_price"] + ")")
    print("*****************************************")
    print()  # Add an extra line for looks
```

Listing 11 Show current products

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

    :return: (string, int) with product and price
    """
    # TODO: Add Code Here!
    product_name = str(input("Product to add: ")).strip().title()
    product_price = str(input("Price in integer: ")).strip()
    print()
    return product_name, product_price
```

Listing 12 Input product to add

Main Body of Script

I call the corresponding class for the actions I want.

```python
# Load data from file into a list of product objects when script starts
Processor.read_data_from_file(file_name=file_name_str, list_of_rows=table_lst)  # read file data

while (True):
    # Show the menu
    IO.output_menu_products()  # Shows menu
    choice_str = IO.input_menu_choice()  # Get menu option

    # Show user current data in the list of product objects
    if choice_str.strip() == '1':  # Show current data
        IO.output_current_products_in_list(list_of_rows=table_lst)  # Show current data in the list/table
        continue  # to show the menu

    # Let user add data to the list of product objects
    elif choice_str.strip() == '2':  # Add a new Task
        product_name, product_price = IO.input_new_product_and_price()
        objP1 = Product(product_name,product_price)
        table_lst = Processor.add_data_to_list(product_name=product_name, product_price=product_price, list_of_rows=table_lst)
        continue  # to show the menu

    # let user save current data to file and exit program
    elif choice_str.strip() == '3':  # Save Data to File
        table_lst = Processor.write_data_to_file(file_name=file_name_str, list_of_rows=table_lst)
        print("Products written to file!")
        print()
        continue  # to show the menu

    elif choice_str == '4':  # Exit Program
        print("Goodbye!")
        print()
        break  # by exiting loop

    else:
        print("Please choose 1, 2, 3, or 4")
        print()
```

Listing 13 Based on the user's choice, perform the corresponding actions

My logic

My code assumes that there is a product.txt file in the working folder, that contains the current list of product name and product price. If the file isn't present, the code informs that the file doesn't exist, and asks the user to place the file. However, the code can still run. It just wouldn't show anything if the user selects "Show Current Data". The user can also select "Add a New Item". Following "Adding a New Item", if the user then selects "Save Data to File", a new product.txt file would be written. However, if the user selects "Exit Program", no file would be created or written.

During "Adding a New Item", My code also raises exceptions when numbers are entered for product name, or texts are entered for product price.

The output in Terminal when the file 'products.txt' cannot be found

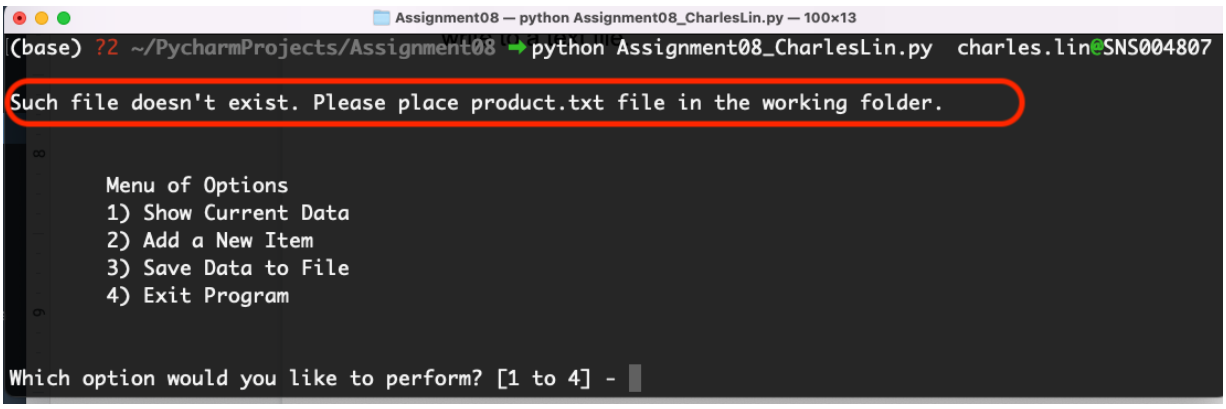

Figure 2 When file is not found, message shows built-in exception and communicates the file is not available

If the user enters anything other than 1, 2, 3, or 4, it would inform them to choose the specified value.

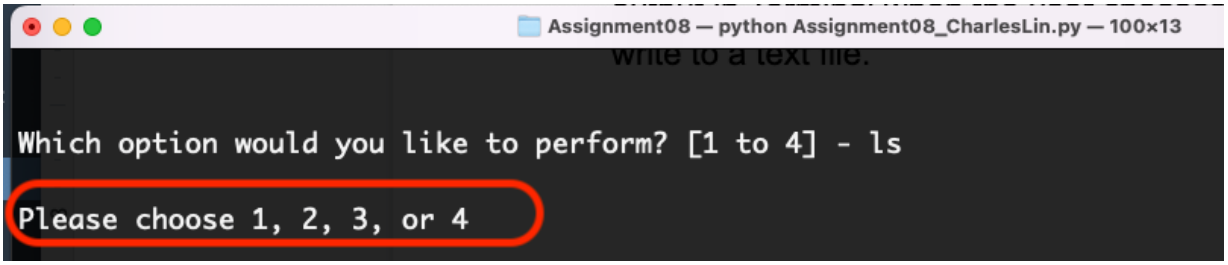

Figure 3 When user enters values other than 1, 2, 3, or 4

When the products.txt is present in the working folder, it would show the current products, and let the user perform other actions, provided that the user enters texts for product name, and numbers for product price.

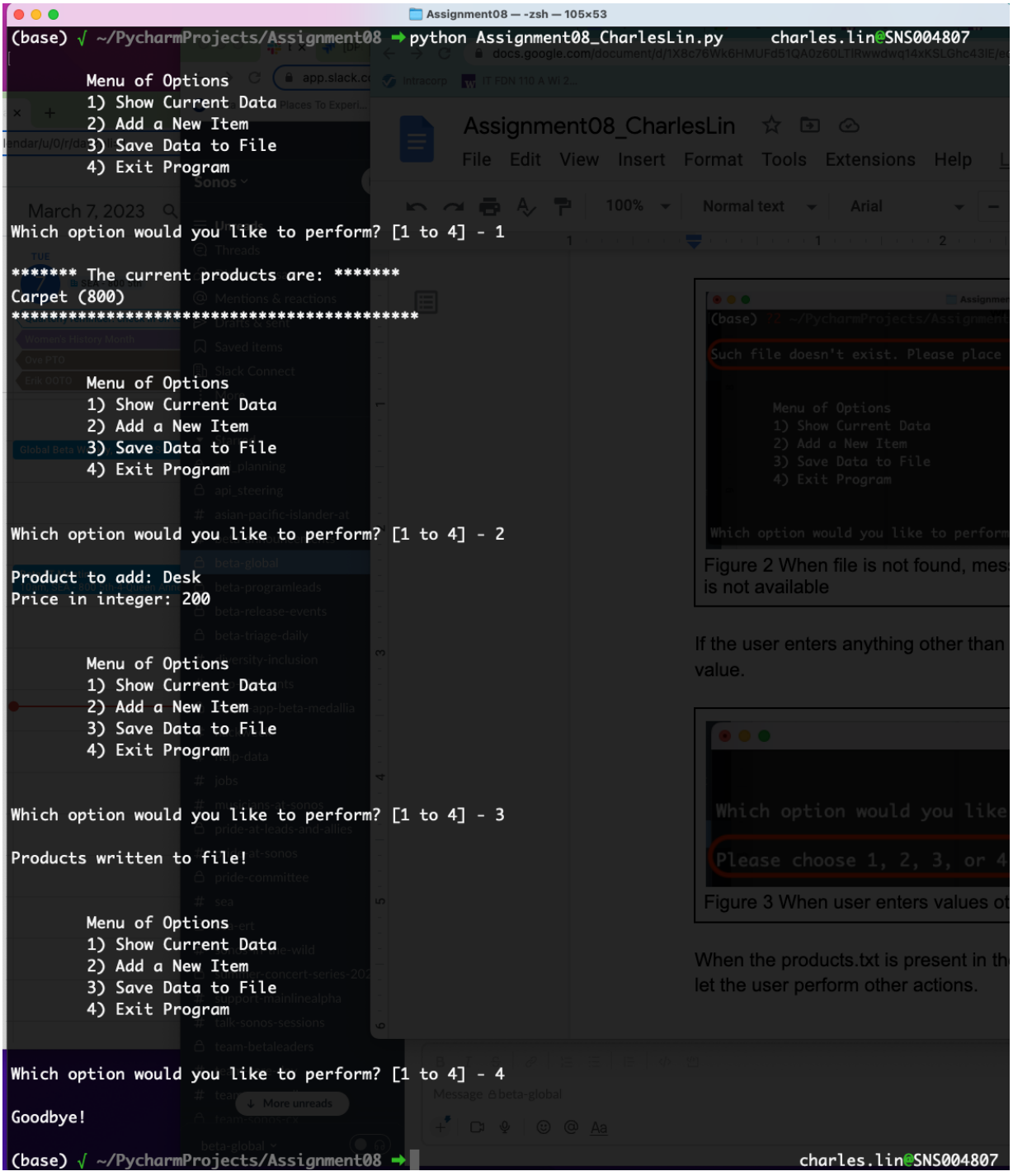

Figure 4 Going through the actions when products.txt is present

If the user enters a number when prompted for product name, the code would throw an exception and exit from the properties set in the Product class.

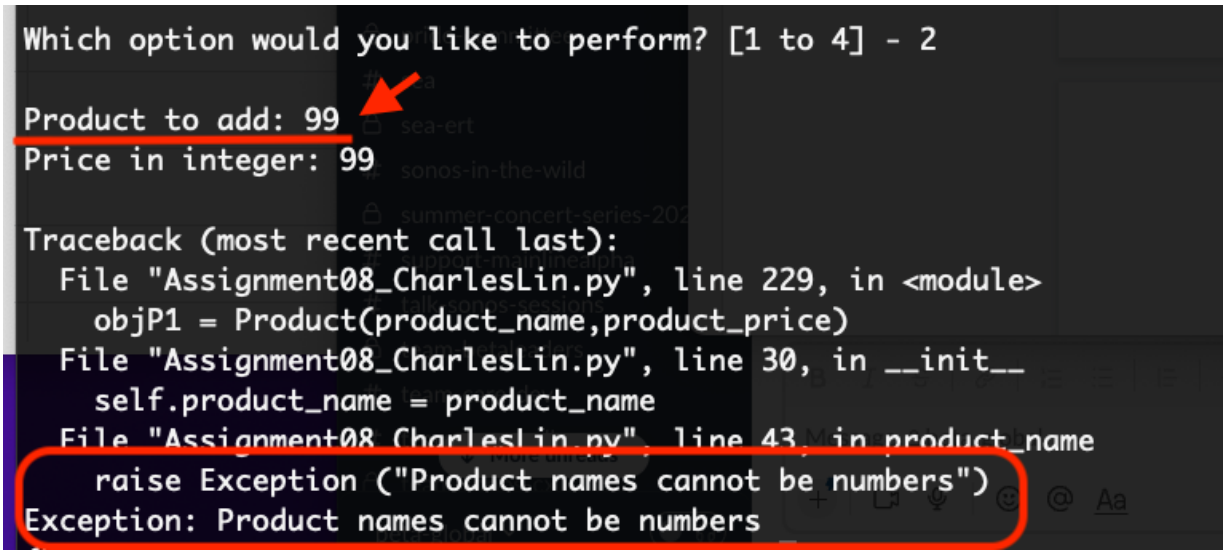

Figure 5 Exception raised from entering numbers for product name

Similarly, if the user enters text when prompted for product price, the code would throw an exception and exit from the properties set in the Product class.

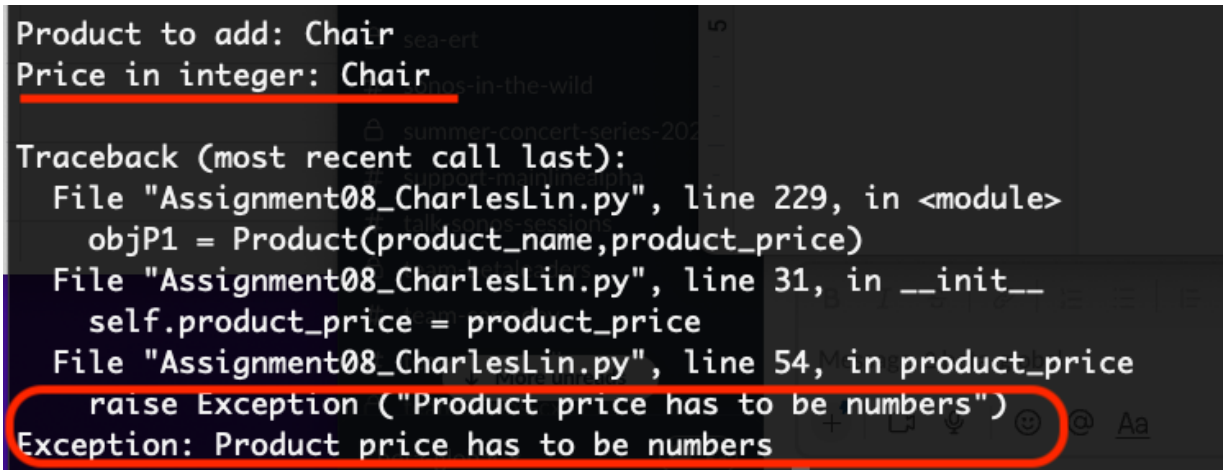

Figure 6 Exception raised from entering text for product price

Summary

I have a better sense of how to use class and function, and the role of a data class, processing class, and presentation (input/output) class. I also know how to organize the code to make it easier to read, by putting them into Data, Processing, Presentation(input/output), and Main Body of Script portions.