James Henderson

8/31/2022

IT FDN 110 B

Assignment08

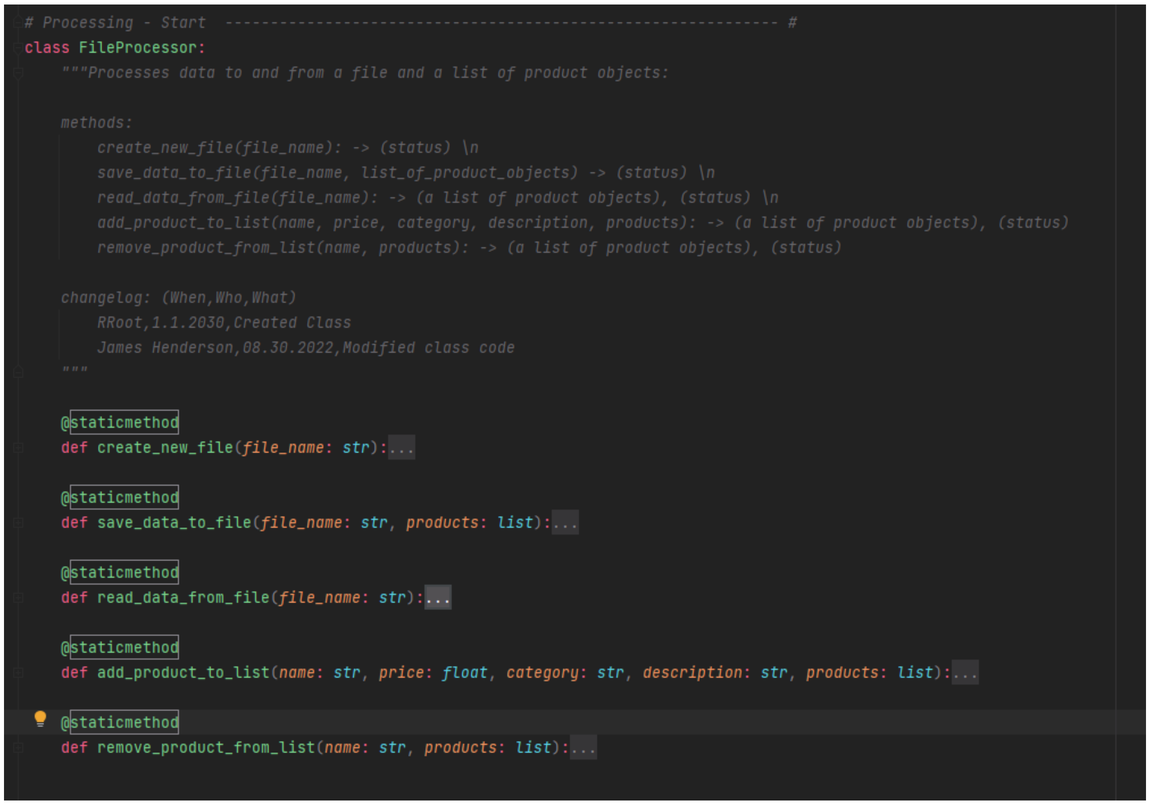
Assignment 8 – Product List Script

Introduction

For this assignment, I worked to modify an existing script similar to the one that we completed for processing a list of tasks and their priorities. I used the base of that code to represent an interactive product list that can be edited and saved to a file in a binary format using the pickle module. The script is separated into 3 distinct layers – processing, I/O, and a main code section. The script also has several areas that illustrate the use of error handling in Python.

Product Script: Processing Binary Data using Pickle

To accomplish the goals of this assignment, I modified a class called FileProcessor which performs the processing tasks. The functions that read / write data out to a flat *.txt file use the pickle module, since that data uses binary formatting; the other functions in this class modify data in the program's memory.

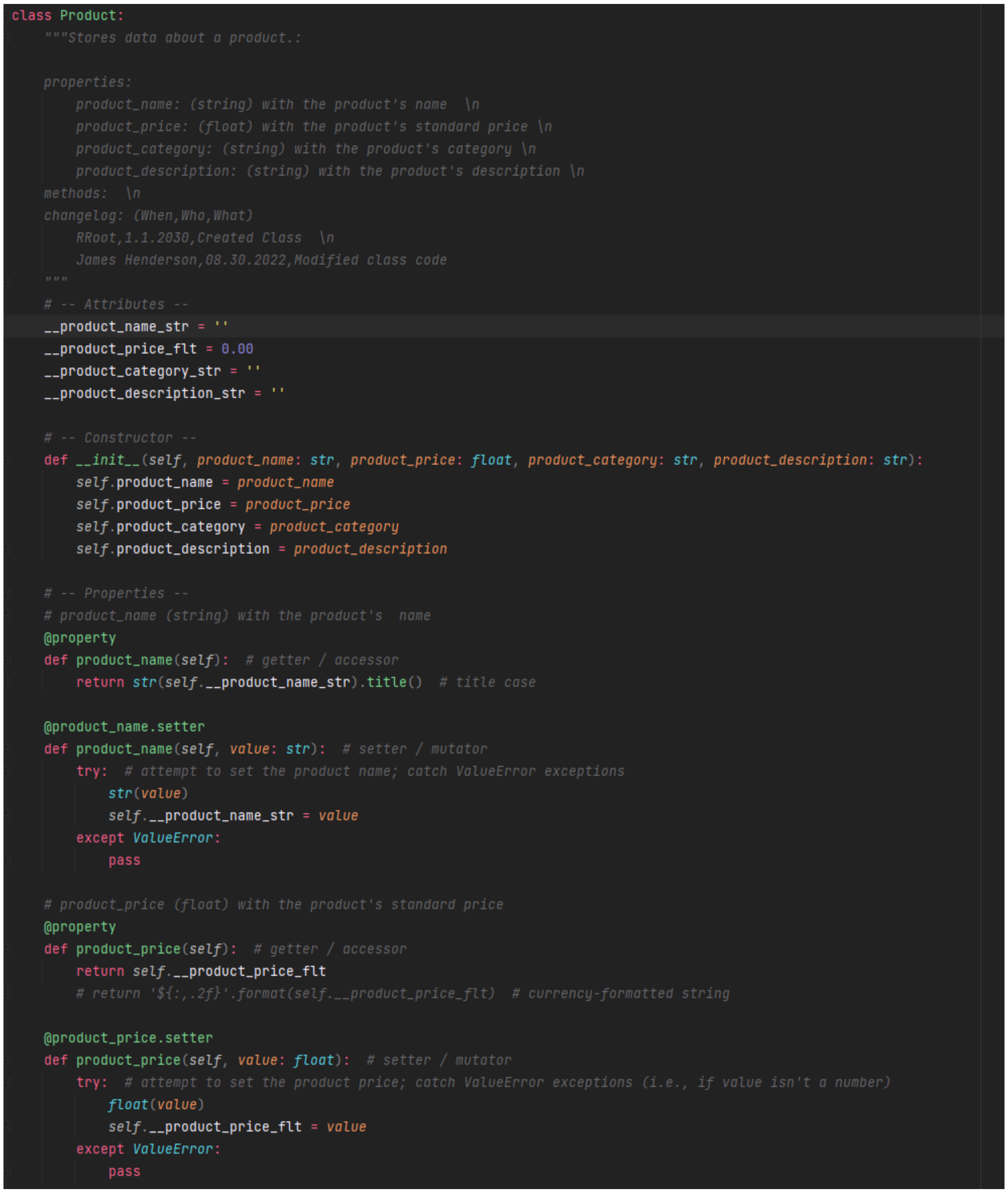


```
# Processing - Start  ------------------------------------------------------------ #
class FileProcessor:
    """Processes data to and from a file and a list of product objects:

    methods:
        create_new_file(file_name): -> (status) \n
        save_data_to_file(file_name, list_of_product_objects) -> (status) \n
        read_data_from_file(file_name): -> (a list of product objects), (status) \n
        add_product_to_list(name, price, category, description, products): -> (a list of product objects), (status)
        remove_product_from_list(name, products): -> (a list of product objects), (status)

    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        James Henderson,08.30.2022,Modified class code
    """

    @staticmethod
    def create_new_file(file_name: str):...

    @staticmethod
    def save_data_to_file(file_name: str, products: list):...

    @staticmethod
    def read_data_from_file(file_name: str):...

    @staticmethod
    def add_product_to_list(name: str, price: float, category: str, description: str, products: list):...

    @staticmethod
    def remove_product_from_list(name: str, products: list):...
```

Figure 1 – The processing layer of the script is organized into the FileProcessor class

Product Script: Storing Data Using a Class with Properties

In **Figure 2**, you can see a sample of the Product class I use to store product data in this script. The Product class I modified has 4 properties, each with a getter and setter method. These are: name, price, category, and description. When the data is written out, even though the pickle.load() function only loads a single line at a time from the text file, and the menu list may contain multiple Product objects, the entire list is treated as if it were a single line – so there is no need to iterate through during the load process. Saving the data is equally simple, using the pickle.dump() function.



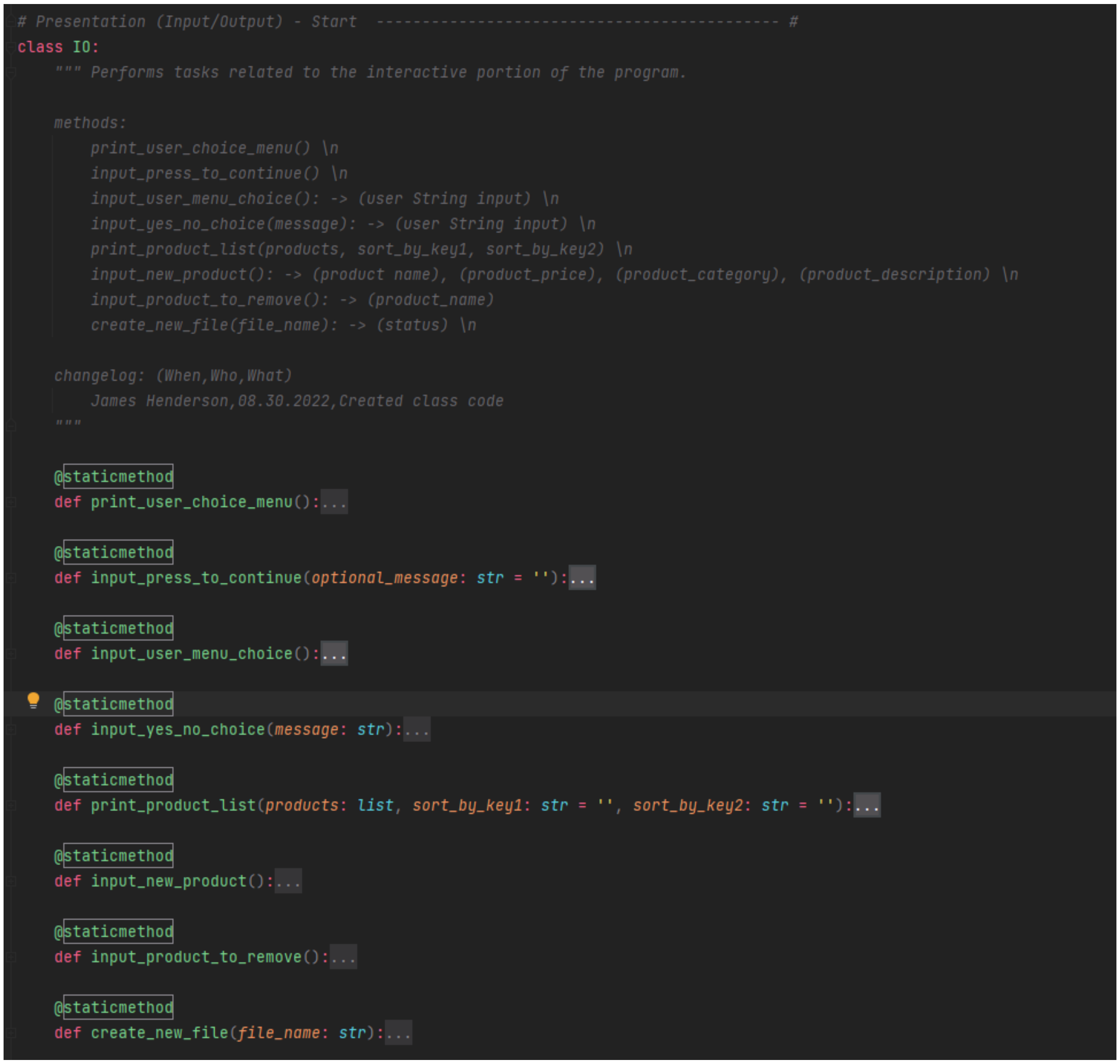
```python
class Product:
    """Stores data about a product.:

    properties:
        product_name: (string) with the product's name  \n
        product_price: (float) with the product's standard price \n
        product_category: (string) with the product's category \n
        product_description: (string) with the product's description \n
    methods:  \n
    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class  \n
        James Henderson,08.30.2022,Modified class code
    """
    # -- Attributes --
    __product_name_str = ''
    __product_price_flt = 0.00
    __product_category_str = ''
    __product_description_str = ''

    # -- Constructor --
    def __init__(self, product_name: str, product_price: float, product_category: str, product_description: str):
        self.product_name = product_name
        self.product_price = product_price
        self.product_category = product_category
        self.product_description = product_description

    # -- Properties --
    # product_name (string) with the product's  name
    @property
    def product_name(self):  # getter / accessor
        return str(self.__product_name_str).title()  # title case

    @product_name.setter
    def product_name(self, value: str):  # setter / mutator
        try:  # attempt to set the product name; catch ValueError exceptions
            str(value)
            self.__product_name_str = value
        except ValueError:
            pass

    # product_price (float) with the product's standard price
    @property
    def product_price(self):  # getter / accessor
        return self.__product_price_flt
        # return '${:,.2f}'.format(self.__product_price_flt)  # currency-formatted string

    @product_price.setter
    def product_price(self, value: float):  # setter / mutator
        try:  # attempt to set the product price; catch ValueError exceptions (i.e., if value isn't a number)
            float(value)
            self.__product_price_flt = value
        except ValueError:
            pass
```

Figure 2 – Using the Product class to store product details in memory

Product Script: Input / Output

As I mentioned in the introduction section, the script for this assignment has been separated into multiple distinct layers. The Input / Output functions that display data to the user, or take in input from the user, are organized into a Class named “IO”. There are eight functions in all – of these, 2 just display data or choices to the user, while 6 are interactive in that they require some type of input from the user.

The interactive functions (names beginning with “input_”) return the user input via one or more variables. Within the main code body, these returned values are then handed off to one of the Processor-class functions as needed. Each of the classes in this script is also documented with a docstring.

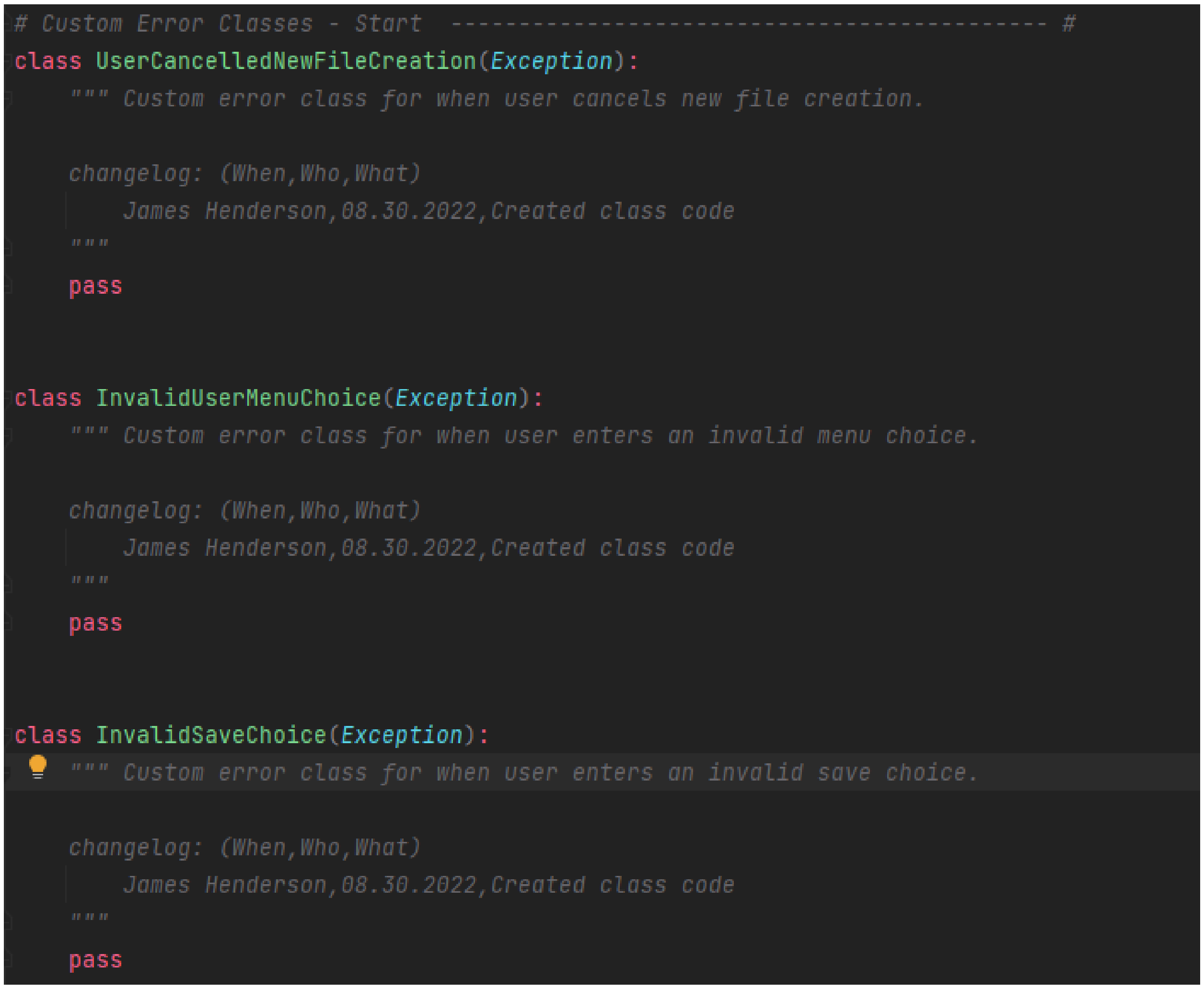


```python
# Presentation (Input/Output) - Start  -------------------------------------------- #
class IO:
    """ Performs tasks related to the interactive portion of the program.

    methods:
        print_user_choice_menu() \n
        input_press_to_continue() \n
        input_user_menu_choice(): -> (user String input) \n
        input_yes_no_choice(message): -> (user String input) \n
        print_product_list(products, sort_by_key1, sort_by_key2) \n
        input_new_product(): -> (product name), (product_price), (product_category), (product_description) \n
        input_product_to_remove(): -> (product_name)
        create_new_file(file_name): -> (status) \n

    changelog: (When,Who,What)
        James Henderson,08.30.2022,Created class code
    """

    @staticmethod
    def print_user_choice_menu():...

    @staticmethod
    def input_press_to_continue(optional_message: str = ''):...

    @staticmethod
    def input_user_menu_choice():...

    @staticmethod
    def input_yes_no_choice(message: str):...

    @staticmethod
    def print_product_list(products: list, sort_by_key1: str = '', sort_by_key2: str = ''):...

    @staticmethod
    def input_new_product():...

    @staticmethod
    def input_product_to_remove():...

    @staticmethod
    def create_new_file(file_name: str):...
```

Figure 3 – There are 8 different static methods for handling interaction with the user – all use type hints

Product Script: Custom Error Classes

The script also contains a few custom error classes for errors that are explicitly raised in the main code body section **(Figure 4):**



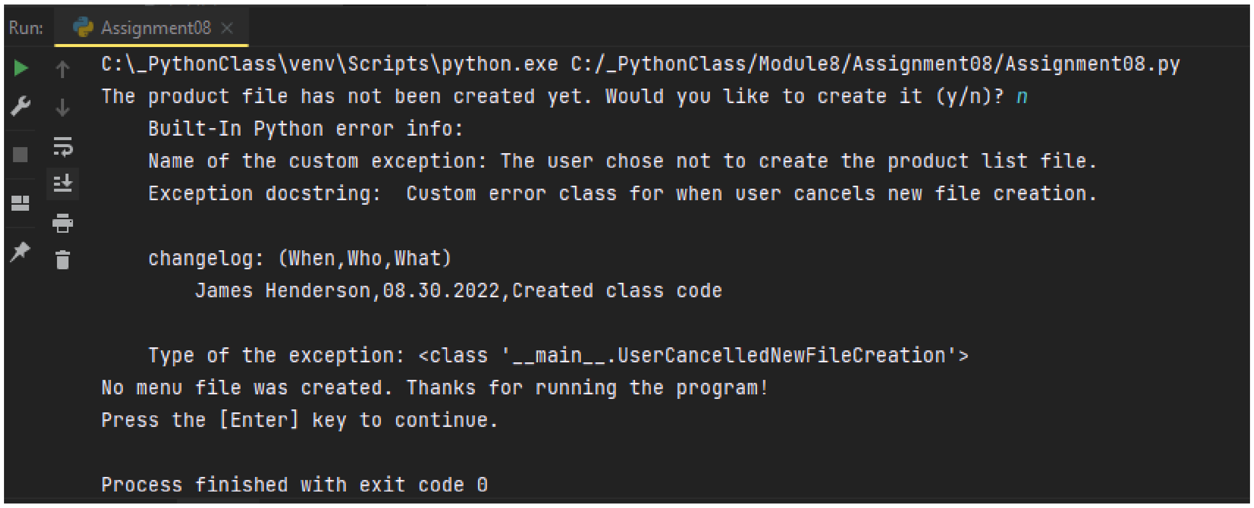
```python
# Custom Error Classes - Start  ------------------------------------------ #
class UserCancelledNewFileCreation(Exception):
    """ Custom error class for when user cancels new file creation.

    changelog: (When,Who,What)
        James Henderson,08.30.2022,Created class code
    """
    pass


class InvalidUserMenuChoice(Exception):
    """ Custom error class for when user enters an invalid menu choice.

    changelog: (When,Who,What)
        James Henderson,08.30.2022,Created class code
    """
    pass


class InvalidSaveChoice(Exception):
    """ Custom error class for when user enters an invalid save choice.

    changelog: (When,Who,What)
        James Henderson,08.30.2022,Created class code
    """
    pass
```

Figure 4 – Custom error classes for better identification of custom errors

Product Script: Running the Script

To test the script, I first ran the Assignment08.py file within PyCharm, using the latest available Python interpreter on my machine (Python 3.10).

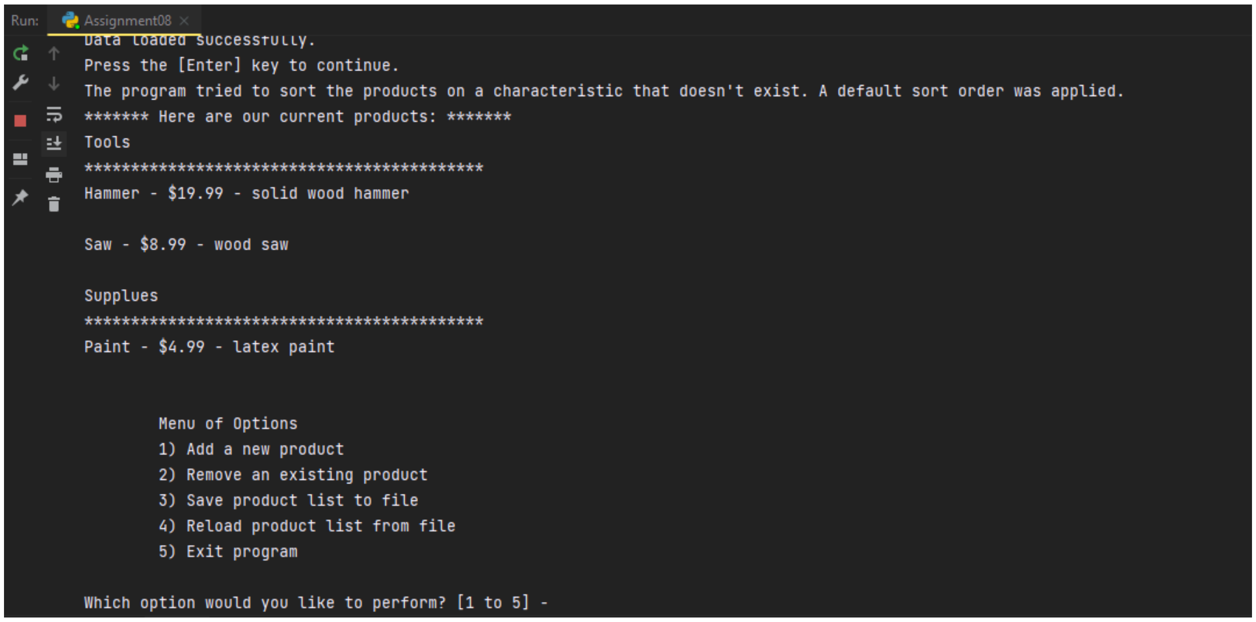
You can see part of the script's input / output in process in ***Figure 5*** below **–** I cancelled the creation of a new file at the start of the script (the text file used for storage hadn't been created yet)



```
C:\_PythonClass\venv\Scripts\python.exe C:/_PythonClass/Module8/Assignment08/Assignment08.py
The product file has not been created yet. Would you like to create it (y/n)? n
    Built-In Python error info:
    Name of the custom exception: The user chose not to create the product list file.
    Exception docstring:  Custom error class for when user cancels new file creation.

    changelog: (When,Who,What)
        James Henderson,08.30.2022,Created class code

    Type of the exception: <class '__main__.UserCancelledNewFileCreation'>
No menu file was created. Thanks for running the program!
Press the [Enter] key to continue.

Process finished with exit code 0
```

Figure 5 – Testing the error handling when the user chose to abort the new file creation process

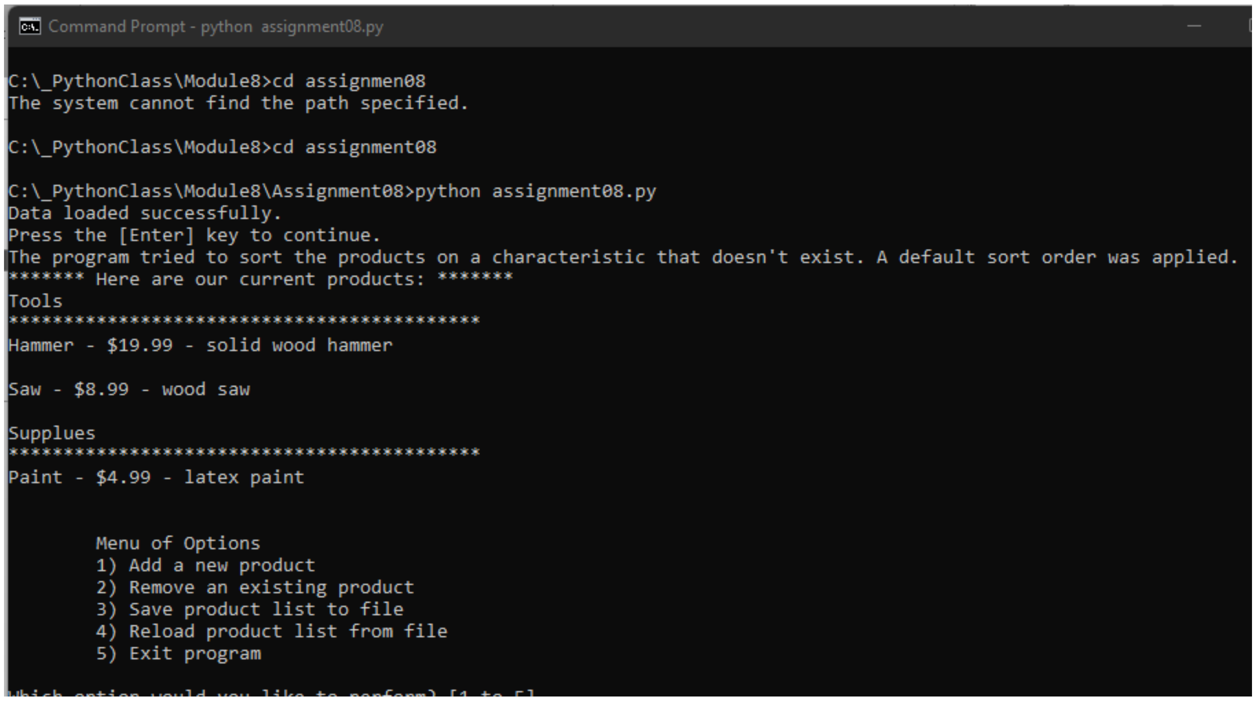
After running the script for a few iterations and adding a few products, you can see the script running and correctly pulling the data out of the txt file ---



```
Data loaded successfully.
Press the [Enter] key to continue.
The program tried to sort the products on a characteristic that doesn't exist. A default sort order was applied.
******* Here are our current products: *******
Tools
*******************************************
Hammer - $19.99 - solid wood hammer

Saw - $8.99 - wood saw

Supplues
*******************************************
Paint - $4.99 - latex paint


        Menu of Options
        1) Add a new product
        2) Remove an existing product
        3) Save product list to file
        4) Reload product list from file
        5) Exit program

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

Figure 6 – The main code running in PyCharm

I also ran the script from the Windows Command Prompt window (***Figure 7).*** I verified that data from a previous run of the script was correctly picked up at the start of the program.



```
C:\_PythonClass\Module8>cd assignmen08
The system cannot find the path specified.

C:\_PythonClass\Module8>cd assignment08

C:\_PythonClass\Module8\Assignment08>python assignment08.py
Data loaded successfully.
Press the [Enter] key to continue.
The program tried to sort the products on a characteristic that doesn't exist. A default sort order was applied.
******* Here are our current products: *******
Tools
*******************************************
Hammer - $19.99 - solid wood hammer

Saw - $8.99 - wood saw

Supplues
*******************************************
Paint - $4.99 - latex paint


        Menu of Options
        1) Add a new product
        2) Remove an existing product
        3) Save product list to file
        4) Reload product list from file
        5) Exit program
```

Figure 7 – running the script from the Windows command prompt

Summary

In this assignment, I edited an existing script to make it work with multiple layers – a processing layer (one class), interactive layer (one class), and a main code body. I also made use of a custom class, Product, for data storage, rather than a Dictionary object. I made sure that the functions AND classes in the script were all appropriately documented with docstrings, and that the code I added was compartmentalized correctly. I also demonstrated the use of pickle to read/write data in a binary format, as well as basic error handling throughout the script.