

## Assignment 08

# Custom Classes, Properties, and Methods with A Product Inventory Manager

### INTRODUCTION

For the eighth assignment for IT FDN 110, students were tasked with demonstrating use of custom classes, properties, and methods by creating a product inventory manager from a starter file. This document will describe the process I took and code I wrote to successfully accomplish this task. The code was written by both R. Root and N. Sandhu, with R. Root providing starting code segments, pseudo-code, and to-do list items to complete the assignment.

### CODE WALKTHROUGH & REASONING

Figure 1 shows the code written to perform the specified function. The first several lines encompass the header of the code, which is self-explanatory and will not be covered in detail in this document. The rest of the code will be explained by section: Data, Processing, and I/O.

#### DATA

The first section of code encompasses initialization of any variables that will be used later in the code. It also imports other modules of code used in the script. The *import pickle* statement imports the pickling module for use in pickling data. The *sys* module is imported due to use of the *sys.exit()* function, which is used for structured error handling of a severe scenario. Several variables are initialized, almost all blank, except for *strFileName*. This variable holds the default file name for the product data binary file, which is *products.dat*.

The custom *Product* class is also defined in this section. This class stores product name and pricing information in the properties *product\_name* and *product\_price*. The constructor for this sets these properties upon initialization. The *product\_name* and *product\_price* properties follow the same format for execution. When a *Product* object instance is created, it should be supplied with a desired name and price. Each property is then set via its designated setter code segment (either *product\_name.setter* or *product\_price.setter*). The properties are private and thus should not be accessed except by the class itself. There is error handling built in to each setter. If the *product\_name* setter comes across a numeric input, an exception is raised and passed to an error handling method, *error\_message*. This function takes the exception and handles output to the user based on the error information inputted into the function. If an error is raised during setting for either property, the property becomes “None,” and based on this, the main body of the program continues its loop without writing the information to the data list. Thus, that instance of the *Product* is not used.

If the properties are set without issue (if *product\_name* is a non-numeric string and *product\_price* can be converted into a float), then the program continues without fault or issue.

## PROCESSING

The second section of code defines the *FileProcessor* class and several functions that perform different tasks in the code. There are three functions in this section.

The first function is *save\_data\_to\_file*, which takes in arguments of *file\_name* (default: *products.dat*) and *list\_of\_product\_objects* (default: empty list) and pickles the list data to a binary file of name *file\_name* in the local code directory. This is programmatically very simple, as pickling data simply involves calling the *pickle.dump()* function on a valid, open binary file *file*. This is all these four lines of code do, and then the file *file* is closed and the function completes with no return values.

The second function is *read\_data\_from\_file*, which reads any existing pickled data from the file *file\_name* (*products.dat* by default). This function starts by trying to open the binary file in read mode. If the file does not exist, this will throw a *FileNotFoundError* in Python, which is handled via an *except* statement. The information for this error is passed to the function in the Input/Output portion of the code called *error\_message*, which then spits out a message to user based on the input argument *e*. In this instance, if the file *products.dat* does not exist, the *sys.exit()* function is called to immediately end the program and display a message to the user that the binary file is required, even if empty, prior to starting the program. The program cannot run without this file. In a future expansion, it would be simple to add a block of code that asks the user if they would like the system to create a new schedule file and then start, however this was not included in this script for the sake of simplicity. If the file exists, its contents are read into the *data* variable, which is then returned to the main body. Reading a pickled file line-by-line will create an *EOFError*, or “end of file error,” and this is handled by calling the *error\_message* function, which calls the *pass* statement to essentially ignore this error when raised in this scenario.

The third function is *remove\_product*. This function removes a product identified by the parameter *name* from the list parameter *data*. This function is very similar to previous weeks and will not be covered in detail in this document. More information on the guts of these functions can be found in the previous documentation.

## INPUT/OUTPUT

This section of code defines all of the I/O necessary to achieve the functionality of the code, mostly consisting of print and input statements, though the main error handling function is also housed in this class.

The first function is a simple *welcome\_message* that prints a message to the user when the code is first started up explaining what the code does and prompting the user to choose a menu option. There are no parameters or returns.

The second function is similar in its simplicity. The *display\_menu* function prints the menu to the user for selecting an option between 1 and 5 to perform the various tasks of the program. There are no parameters or returns.

The third function, *get\_menu\_input*, returns the user's choice for the menu in the parameter *choice*.

The fourth function is the main error handling function in the script, *error\_message*. At most common error-prone points (where the user makes an input), a *try-except* format is used. If an error is returned from these areas, the error information is stored in the variable *e* and is passed to this function. This function determines what type of error *e* is and returns a message based on that information or displays the custom exception information, if applicable. In most cases, the built-in Python error information is not displayed to the user for simplicity, as most mistakes are easily corrected by the user by trying their input again. Because this program is based on file I/O and user input of values (menu options, user input product name and price information, etc), the most common errors are file errors (*EOFError* and *FileNotFoundError*) or *ValueErrors*, where the user's input is not within the bounds of useability for the code. One of the custom *Exception* errors in this code is raised when the user inputs a menu option that is an integer, but not within the bounds of the menu (i.e., if the entered integer is not between 1 and 5). In this case, a custom error message is displayed, prompting the user to only enter an integer between 1 and 5 for the main menu. Other custom exceptions were given in the *Product* class description and code for setting class properties.

The fifth function in this class displays existing product data in the parameter *data* to the user on-screen. This is achieved by looping through the rows of *data* and printing them to the user. This is done by calling the *product\_name* and *product\_price* properties for each row item, which are strictly only *Product* class objects. There are no returned parameters in this function.

The sixth function, *get\_user\_product\_info*, is the main input function for the code. Two inputs, *user\_product\_name* and *user\_product\_price*, are taken from the user, which are then fed into the *Product* class to create an instance of a product object (aptly stored in the *product\_obj* variable). This *product\_obj* is returned as an output of the function. The function has no inputs.

The seventh and eighth functions are very simple in operation. The *get\_user\_product\_to\_remove* function returns a string input from the user for the name of a product they'd like to remove from the data list, while the *exit\_message* function displays a message to the user prior to the program ending.

## MAIN BODY

The main body of the script starts by calling the *read\_data\_from\_file* function from the *FileProcessor* class and storing the returned information in the variable *lstofProductObjects*. This variable serves as the main vessel for product data as it is manipulated through the code. The function is fed the *strFileName* variable, which is assigned to *products.dat* in the data portion of the code.

Next, the *welcome\_message* function is called to present a welcome message to the user. The process of displaying this message and initializing the *lstOfProductObjects* variable only occur once for every code run.

The remainder of the code is wrapped in an infinite loop, such that the user can continually make menu choices as long as they do not explicitly choose to exit the program. If the user enters a non-integer option for their menu input (retrieved via a call to *get\_menu\_input*), then a *ValueError* is raised and handled. If an integer is entered but it is out of range, then a custom error is raised and the user is told to restrict inputs to integers between 1 and 5. In both cases, the menu is displayed again and the user may make another input for menu option.

If a valid menu option is entered, a *match-case* system is used to handle the different choices. This formatting was covered in detail during the write up for Assignment 07 and will not be covered in detail here. Please refer to previous documentation for further information.

Case 1 is simple in that it calls the *view\_list* function with *lstOfProductObjects* as an input argument. This displays the current list of product and pricing data to the user.

Case 2 allows the user to add additional product and pricing information to the *lstOfProductObjects* via the *get\_user\_product\_info* function. This function returns a *Product* object, which is temporarily stored in the *userAdd* variable. Based on the conditions in the creation of a *Product* object (name must be non-numeric and price must convert to float), the properties *product\_name* and *product\_price* could feasibly be *None*. If this is true for either property, the loop continues and starts again, while displaying an error stating which of the properties failed to set. The logic allows for both errors to display to the user. If the properties are set appropriately without error, the *userAdd* object is then appended to the *lstOfProductObjects* variable.

Case 3, 4, and 5 are very similar to Assignment 7 in terms of operation and will not be covered in detail in this document. Case 3 retrieves a user input for an object to remove. The existing list of product data is looped through and if the user's input is found, the entire object is removed. Case 4 allows the user to save data via pickling to the *products.dat* file in the local directory. Case 5 exits the program.

Operation of the code in PyCharm and the Windows Command Console is given in Figs. 2 through 10.

## SUMMARY

In summary, this document describes the code behind a successful implementation of a custom product class in use for managing product and pricing information. The class includes custom properties and other custom classes include static methods used to do file processing, handle user inputs, and handle errors. Several common errors in the code are handled such that user-friendly error messages are displayed and the code continues to run rather than breaking and displaying high-level error information. Data is successfully pickled and unpickled from a binary file.

```

# ----- #
# Title: Assignment 08
# Description: Working with classes

# ChangeLog (Who,When,What):
# RRoot,1.1.2030,Created started script
# RRoot,1.1.2030,Added pseudo-code to start assignment 8
# NSandhu,06.05.2022,Modified code to complete assignment 8
# ----- #

import sys
import pickle

# Data ----- #
strFileName = 'products.dat'
lstOfProductObjects = []
userChoice = ""
userAdd = None
userRemove = ""

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: (When,Who,What)
            RRoot,1.1.2030,Created Class
            NSandhu,06.05.2022,Modified code to complete assignment 8
    """

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

    # -- Properties -- #
    # Product Name
    @property
    def product_name(self):
        if str(self.__product_name_str).isalpha():
            return str(self.__product_name_str).title()
        else:
            return self.__product_name_str

    @product_name.setter
    def product_name(self, data):
        try:
            if not str(data).isnumeric():
                self.__product_name_str = data
            else:
                raise Exception("Product names cannot contain numbers!")
        except Exception as e1:
            IO.error_message(e1)

```

```

        self.__product_name_str = None

# Product Price
@property
def product_price(self):
    if str(self.__product_price_val).isnumeric():
        return float(self.__product_price_val)
    else:
        return self.__product_price_val

@product_price.setter
def product_price(self, data):
    try:
        if not str(data).isalpha():
            self.__product_price_val = data
        else:
            raise Exception("Product prices cannot contain non-numeric
characters!")
    except Exception as e2:
        IO.error_message(e2)
        self.__product_price_val = None

# -- End of Class -- #

# End of 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): -> binary file
with product/price data

        read_data_from_file(file_name): -> (a list of product objects)

        remove_product(name,data): -> list of product objects with user input
removed if found

    changelog: (When,Who,What)
        RRoot,1.1.2030,Created Class
        NSandhu,06.05.2022,Modified code to complete assignment 8
    """

    @staticmethod
    def save_data_to_file(file_name="products.dat",
list_of_product_objects=[]):
        """ Writes data from a list of dictionary rows to a File

        :param file_name: (string) with name of file, default is
products.dat:
        :param list_of_product_objects: (list) you want filled with file
data, default is blank:
        :return: nothing

```

```

        """
        file = open(file_name, "ab")
        pickle.dump(list_of_product_objects, file)
        file.close()
        print("Data saved!")

    @staticmethod
    def read_data_from_file(file_name="products.dat"):
        """ Reads in existing pickled data, else creates new pickle file

        :param file_name: file where stored schedule data is, default is
products.dat. Note that this file needs to
exist prior to code execution:
        :return: data
        """
        data = []
        try:
            file = open(file_name, "wb")
            while True:
                try:
                    data = pickle.load(file)
                except EOFError as e:
                    IO.error_message(e)
                    break
        except FileNotFoundError as e:
            IO.error_message(e)
        file.close()
        return data

    @staticmethod
    def remove_product(name, data):
        """ Removes data from a list of product objects

        :param event: (string) with name of event to remove:
        :param data: (list) filled with schedule data:
        :return: (list) of dictionary rows of schedule data
        """
        i = 0
        flag = 0
        for row in data:
            if row.product_name.lower() == name.lower().strip():
                flag = 1
                i += 1
                data.remove(row)
                print("Product removed from list.")
            else:
                i += 1
                if i == len(data) and flag == 0:
                    print("Product not found.")
                    print() # print new line for looks
        return data

# End of Processing -----
----- #

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

```

```

class IO:
    """Performs Input/Output tasks, presenting and retrieving outputs and
    inputs for/from the user:

    methods:
        welcome_message(): -> welcome message to user

        display_menu(): -> menu display to user

        get_menu_input(): -> user input for menu option desired

        error_message(error): -> error information displayed to user

        view_list(data): -> current list of product information displayed to
user

        get_user_product_info(): -> Product object with (obj).product_name
and (obj).product_price info

        get_user_product_to_remove(): -> string from user input for product
to remove from list

        exit_message(): -> exit message displayed to user

    changelog: (When,Who,What)
        NSandhu,06.05.2022,Created class and added necessary methods for
Assignment08
    """
    @staticmethod
    def welcome_message():
        """ Display a Welcome message to the user

        :return: nothing
        """

        print("""
Welcome to the Product Pricer! I will help you
create and manage a list of product items and their prices.
Please choose from a menu option to get started:
""")

    @staticmethod
    def display_menu():
        """ Display a menu of choices to the user

        :return: nothing
        """
        print('\n'
Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program
')
        print()

```



```

@staticmethod
def get_menu_input():
    """ Get user input for any item

    :return: choice
    """
    choice = input("Please select a menu option from 1-5: ").strip()
    return choice

@staticmethod
def error_message(error):
    """ Displays one of several error messages based on exception raised

    :param error: an exception object:
    :return: nothing
    """
    print()
    if isinstance(error, ValueError):
        print("Please only enter integer values!\n")
    elif isinstance(error, EOFError):
        pass
    elif isinstance(error, FileNotFoundError):
        sys.exit("Product file does not exist, please create a blank
'products.dat' file in the local directory "
               "and try again")
    elif isinstance(error, Exception):
        print(error)

@staticmethod
def view_list(data):
    """ Shows the current list of products and prices

    :param data: list of data to be displayed:
    :return: nothing
    """
    print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
    print("The Product List is: ")
    for row in data:
        print(row.product_name, str(row.product_price), sep="\t||\t")
    print("%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%")
    print()

@staticmethod
def get_user_product_info():
    """
    Returns user product and pricing information

    :param: none
    :return product_obj: -> Product object instance with
(obj).product_name and (obj).product_price info
    :return e: -> error information
    """
    user_product_name = input("Please enter Product Name (non-numeric
only): ")
    user_product_price = input("Please enter Product Price (numeric
only): ")
    product_obj = Product(user_product_name, user_product_price)

```

```

        return product_obj

    @staticmethod
    def get_user_product_to_remove():
        """
        Get user input for product to remove from list

        :param: none
        :return remove_product: string from user for product to remove
        """
        remove_product = input("Please input product name to remove:
").strip()
        return remove_product

    @staticmethod
    def exit_message():
        """ Displays a message to the user prior to exiting program

        :return: nothing
        """
        print("Thank you! Program will end upon hitting 'Enter'...")
# End of Presentation (Input/Output) -----
----- #

# Main Body of Script ----- #

lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)

IO.welcome_message()

while True:
    IO.display_menu()
    try:
        userChoice = int(IO.get_menu_input())
        if userChoice not in range(1,6):
            raise Exception("Please only input a value from 1 to 5!")
    except ValueError as e:
        IO.error_message(e)
        continue
    except Exception as e:
        IO.error_message(e)
        continue

    match userChoice:
        case 1:
            IO.view_list(lstOfProductObjects)
        case 2:
            userAdd = IO.get_user_product_info()
            if userAdd.product_name == 'None':
                continue
            elif userAdd.product_price is None:
                continue
            else:
                lstOfProductObjects.append(userAdd)
        case 3:
            userRemove = IO.get_user_product_to_remove()
            lstOfProductObjects =

```

```
FileProcessor.remove_product(userRemove, lstOfProductObjects)
    case 4:
        FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)
    case 5:
        IO.exit_message()
        input()
        break

# End Main ----- #
```

**Fig. 1.** Code written by both R. Root and N. Sandhu to accomplish the task of creating a product inventory manager using a custom class, properties, and methods.

The screenshot shows a code editor with a Python script named 'Assignment08-Starter.py'. The script is a simple data processing application for a product pricer. It features a menu of options for the user to interact with, including viewing the current product list, adding new products, removing products, saving the list, and exiting the program. The terminal output shows the program running, displaying the menu and the current product list: Banana || 0.5.

```

Code > Assignment08-Starter.py
Project
  Code [Mod08Listings] C:\Python
    AppData.txt
    Assignment08-Starter.py
  Run: Assignment08-Starter
    "C:\_PythonClass\Module02 - Simple Data Processing\Code\venv\Scripts\python.exe" "C:/_Py

Welcome to the Product Pricer! I will help you
create and manage a list of product items and their prices.
Please choose from a menu option to get started:

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 1
%%%%%%%%%
The Product List is:
Banana || 0.5
%%%%%%%%%

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 2
Please enter Product Name (non-numeric only): apple
Please enter Product Price (numeric only): 0.5

```

```
Code / Assignment08-Starter.py
Project
  Code [Mod08Listings] C:\Python
    AppData.txt
    Assignment08-Starter.py
  Run: Assignment08-Starter
    2) Add a new Product and Price to the List
    3) Remove a Product from the List
    4) Save the List
    5) Exit Program
    Please select a menu option from 1-5: 2
    Please enter Product Name (non-numeric only): apple
    Please enter Product Price (numeric only): 0.5
    Menu of Options
    1) View current Product List
    2) Add a new Product and Price to the List
    3) Remove a Product from the List
    4) Save the List
    5) Exit Program
    Please select a menu option from 1-5: 1
    %%%%%%%%%%
    The Product List is:
    Banana || 0.5
    Apple  || 0.5
    %%%%%%%%%%
    Menu of Options
    1) View current Product List
    2) Add a new Product and Price to the List
    3) Remove a Product from the List
    4) Save the List
    5) Exit Program
    Please select a menu option from 1-5: |
  Version Control Run Debug Python Packages TODO Python Console Problems
  Breakpoint reached (yesterday 5:07 PM)
```

**Fig. 3.** The code operating in PyCharm. The ability to add product and pricing information are shown.

The image shows the PyCharm IDE interface. The top toolbar includes icons for Project, Run, Debug, Python Packages, TODO, Python Console, and Problems. The Project pane on the left shows a folder 'Code [Mod08Listings]' containing 'AppData.txt' and 'Assignment08-Starter.py'. The Run pane at the bottom shows the execution of 'Assignment08-Starter.py'. The main editor displays the script's output, which includes a menu of options and a demonstration of error handling for invalid input.

```
Code / Assignment08-Starter.py
Project
  Code [Mod08Listings] C:\Python
    AppData.txt
    Assignment08-Starter.py
Run: Assignment08-Starter x
  Please select a menu option from 1-5: 3
  Please input product name to remove: banana
  Product removed from list.

  Menu of Options
  1) View current Product List
  2) Add a new Product and Price to the List
  3) Remove a Product from the List
  4) Save the List
  5) Exit Program

  Please select a menu option from 1-5: 1
  %%%
  The Product List is:
  Apple || 0.5
  %%%

  Menu of Options
  1) View current Product List
  2) Add a new Product and Price to the List
  3) Remove a Product from the List
  4) Save the List
  5) Exit Program

  Please select a menu option from 1-5: f
  Please only enter integer values!

  Version Control Run Debug Python Packages TODO Python Console Problems
  Breakpoint reached (yesterday 5:07 PM)
```

**Fig. 4.** The code operating in PyCharm. A demonstration of error handling is shown with the user inputting an improper option for the main menu.

The image shows the PyCharm IDE interface. The top toolbar includes icons for Project, Code, Run, and other development tools. The Project pane on the left shows a folder named 'Code [Mod08Listings]' containing files like 'AppData.txt' and 'Assignment08-Starter.py'. The main editor window displays a Python script with line numbers 289 and 290. The script contains a menu of options and a loop for adding products. The Run pane at the bottom shows the execution output, including prompts for menu options, product names, and prices, as well as error messages for invalid input.

```
Code > Assignment08-Starter.py
Project
  Code [Mod08Listings] C:\Python
    AppData.txt
    Assignment08-Starter.py
    ...
Run: Assignment08-Starter
  Please select a menu option from 1-5: 2
  Please enter Product Name (non-numeric only): 1
  Please enter Product Price (numeric only): apple
  Product names cannot contain numbers!
  Product prices cannot contain non-numeric characters!

  Menu of Options
  1) View current Product List
  2) Add a new Product and Price to the List
  3) Remove a Product from the List
  4) Save the List
  5) Exit Program

  Please select a menu option from 1-5: 2
  Please enter Product Name (non-numeric only): apple
  Please enter Product Price (numeric only): apple

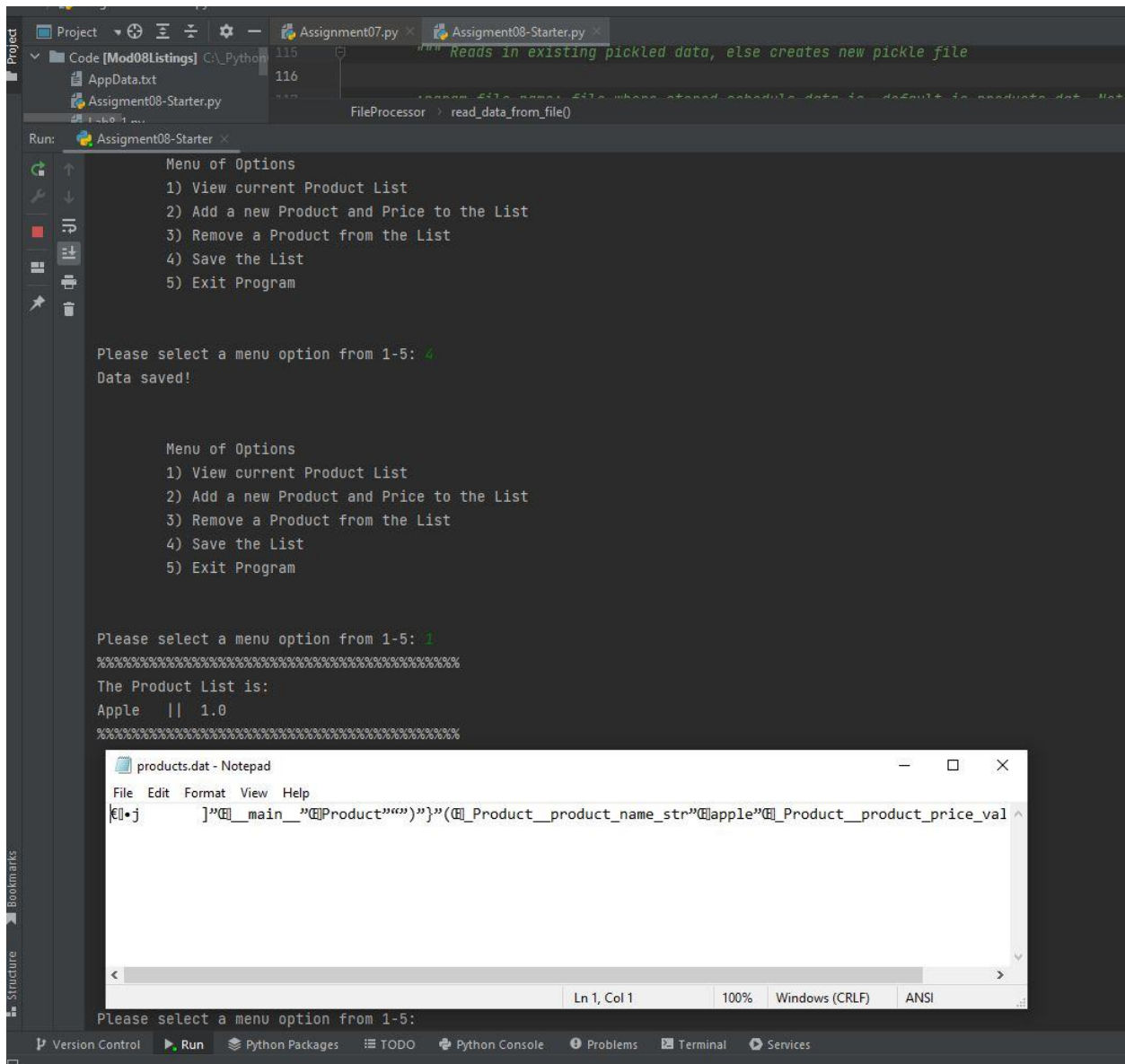
  Product prices cannot contain non-numeric characters!

  Menu of Options
  1) View current Product List
  2) Add a new Product and Price to the List
  3) Remove a Product from the List
  4) Save the List
  5) Exit Program

  Please select a menu option from 1-5: 2
  Please enter Product Name (non-numeric only): 1
  Please enter Product Price (numeric only): 1

  Product names cannot contain numbers!
```

**Fig. 5.** The code operating in PyCharm. Error handling is demonstrated with the user entering numeric characters for a product name and alpha characters for pricing information.



**Fig. 6.** The code operating in PyCharm. The ability to save data via pickling is shown.



```
Command Prompt - python.exe Assignment08-Starter.py

c:\_PythonClass\Module08 - Classes and Objects Updated\Code>python.exe Assignment08-Starter.py

Welcome to the Product Pricer! I will help you
create and manage a list of product items and their prices.
Please choose from a menu option to get started:

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The Product List is:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 2
Please enter Product Name (non-numeric only): apple
Please enter Product Price (numeric only): 3

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
The Product List is:
Apple || 3.0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program
```

**Fig. 7.** The code running in the Windows Command Console. Code start, an empty existing list, and the ability to add product information are shown.

```
Command Prompt - python.exe Assignment08-Starter.py

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 2
Please enter Product Name (non-numeric only): Banana
Please enter Product Price (numeric only): 0.5

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 1
The Product List is:
Apple || 3.0
Banana || 0.5

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 3
Please input product name to remove: apple
Product removed from list.

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 1
The Product List is:
Banana || 0.5
```

**Fig. 8.** The code running in the Windows Command Console. The ability to remove product information is shown.

```
Command Prompt - python.exe Assignment08-Starter.py

Please select a menu option from 1-5: 1
*****
The Product List is:
Banana || 0.5
*****

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: a
Please only enter integer values!

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 2
Please enter Product Name (non-numeric only): 123
Please enter Product Price (numeric only): apple

Product names cannot contain numbers!
Product prices cannot contain non-numeric characters!

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5:
```

**Fig. 9.** The code running in the Windows Command Console. Error handling is demonstrated with the user entering numeric characters for a product name and alpha characters for pricing information.

```
Command Prompt - python.exe Assignment08-Starter.py

5) Exit Program

Please select a menu option from 1-5: a
Please only enter integer values!

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 2
Please enter Product Name (non-numeric only): 123
Please enter Product Price (numeric only): apple

Product names cannot contain numbers!
Product prices cannot contain non-numeric characters!

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

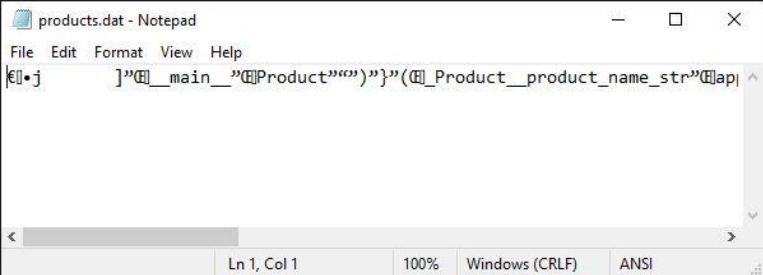
Please select a menu option from 1-5: 1
The Product List is:
Banana || 0.5

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5: 4
Data saved!

Menu of Options
1) View current Product List
2) Add a new Product and Price to the List
3) Remove a Product from the List
4) Save the List
5) Exit Program

Please select a menu option from 1-5:
```



**Fig. 10.** The code running in the Windows Command Console. The ability to save data via pickling is shown.