M.Bruce
Aug 29, 2022
IT FDN 110 (B)
Assignment 08
Link:(GitHub repository)- https://github.com/Jane2024/IntroToProg-Python-Mod08

Classes in Python

Introduction

This week covered the types of classes, class objects, the standard class pattern, abstraction, docstrings and Github desktop. The assignment for the week was to read and understand the provided pseudo-code in the Assignment08-Starter.py script, then add code to make the application work and adding some error handling as well. This knowledge document will cover the above topics how the assignment was accomplished.

Classes

In Python, classes are a way of grouping data and functions to perform particular actions. Like functions they are loaded into memory and run when they are called. A number of elements make up a class and normally a class contains a standard pattern of items. Classes help to organize functions like functions help to organize statements. Classes usually fall under two types: Data or Processing.

Data class vs Processing Class

Most classes are designed to either organize data or handle processing of data. A “product” class would gather and organize data about a product. A class called “product_inventory” would handle processing of data of all products and provide an inventory of them. The data in a class is stored in “fields” which are the same as variables and constants. Functions in a class are called “methods”.

Objects

An object is a runtime instance of the class. It contains attributes defined in the class. If a cookie recipe is like a class definition “cookie”, then a cookie would be an actual object of the class, an instance of it. Its attributes might make it a chocolate chip cookie or a peanut butter cookie. Classes are loading into memory when a program launches and can be called one of two ways-

Directly:

product.Name = car
product.ID = 1
This is the best method to use when processing data.

Indirectly via a copy or instance of the class (there can be numerous instances of a class):

objproduct1= Product()
objproduct1.ID = 1
objproduct1.Name= car
This is best method to use when storing and organizing data of a class object

Standard Class Pattern

Classes typically have **Fields, Constructors, Properties, and Methods**. Like scripts, class code follows a general design pattern in most of the languages.

Fields

Fields are the data that is defined inside a class via variables and constants. See ***Listing 8***.*1* below.

```
class Product:
   # --Fields--
   product_name_str = ""

   # -- Constructor --
   # -- Attributes --
   # -- Properties --
   # -- Methods –

# End of class
```

LISTING 8.1 Code to define a field in class “Product”

Lab 8-1 Class to hold data

For Lab 8-1, create a simple class to hold personal data. See **Figure 8.1** below for code example.

```
# ------------------------------------------------ #
# Title: Lab_8-1_MBruce
# Description: A class with a field
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# MBruce,8.26.2022, added last name
# ------------------------------------------------ #

#--- Make the class ---
class Person:
    # --Fields--
    strFirstName = ""
    strLastName = ""
    # -- Constructor --
        # -- Attributes --
    # -- Properties --
    # -- Methods --
# End of class

# --- Use the class ----
objP1 = Person()
objP1.strFirstName = "Bob"
objP1.strLastName = "Marley"

objP2 = Person()
objP2.strFirstName = "Sue"
objP2.strLastName = "Foley"

print(objP1.strFirstName, objP1.strLastName )
print("-------------")
print(objP2.strFirstName, objP2.strLastName)
```

FIGURE 8.1 Lab8-1 script code for class “Person”

The output of this code is shown is ***Figure 8.1.1*** below.

```
c:\_PythonClass\DemoCode8>python.exe C:\_PythonClass\DemoCode8\Lab_8-1_MBruce.py
Bob Marley
-------------
Sue Foley
```

FIGURE 8.1.1 Lab8-1 output of simple class “Person”

Constructors

Constructors are special methods (functions) that automatically runs when you create an object from the class. Constructors instantiate an object with some initial values. Python's constructors use a double underscore name of "__init__".
When creating an object of a class it is called like a function with arguments. The constructor uses the arguments to populate the object if its initial values or used default values if none are provided in the arguments.
objProduct1 = Product("car"). See ***Listing 8.2*** below

```
class Product:
   # --Fields--
   product_name_str = ""

   # -- Constructor --
   def __init__(self, product_name = “”):  # The default is an empty string
      #-- Attributes --
      self.product_name_str = product_name

   # -- Properties --
   # -- Methods --
# --End of class--
```

LISTING 8.2 Listing of class “Product” with a constructor

Self Keyword

This keyword “self” is used to **refer to data or functions found in an object instance,** but **not directly in the class**. Since there can be multiple object instances of a class, “self” helps identify them.

Lab 8-2 Adding a constructor

See **Figure 8-2 below** for Lab8-2 code example. This will define a constructor with two parameters.

```
# ------------------------------------------------ #
# Title: Lab_8-2_MBruce.py
# Description: A class with two fields +constructor
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# MBruce,8.26.2022, added last name and constructor
# ------------------------------------------------ #

#--- Make the class ---
class Person:
    # --Fields--
    strFirstName = ""
    strLastName = ""

    # -- Constructor --
    def __init__(self, first_name = "", last_name = ""):
        self.strFirstName = first_name
        self.strLastName = last_name

            # -- Attributes --
    # -- Properties --
    # -- Methods --
# End of class

# --- Use the class ----
objP1 = Person("Bob", "Martley")
objP2 = Person("Sue", "Foley")


print(objP1.strFirstName, objP1.strLastName)
print("-------------")
print(objP2.strFirstName, objP2.strLastName)
```

FIGURE 8.2 Lab8-2 code example

In Figure 8.2.1, the output of the code from lab 8.2 is shown.

```
c:\_PythonClass\DemoCode8>python.exe C:\_PythonClass\DemoCode8\Lab_8-2_MBruce.py
Bob Martley
-------------
Sue Foley
```

FIGURE 8.2.1 Lab8-2 output in cmd shell

Attributes

Attributes are virtual fields.

Lab 8-3 Class to hold data

For Lab 8-3, used the same script code from Lab8-2 to create a constructor that implicitly sets the values of the two attributes for first name and last name. See **Figure 8.3** below for code example of lab.

```
# ------------------------------------------------ #
# Title: Lab_8-3_MBruce.py
# Description: A class constructor + attributes
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# MBruce,8.26.2022, added last name,constructor, attributes
# ------------------------------------------------ #

#--- Make the class ---
class Person:
    # --Fields--
    #strFirstName = ""
    #strLastName = ""

    # -- Constructor --
    def __init__(self, first_name = "", last_name = ""):
         # -- Attributes --
        self.strFirstName = first_name
        self.strLastName = last_name


    # -- Properties --
    # -- Methods --
# End of class

# --- Use the class ----
objP1 = Person("Bob", "Martley")
objP2 = Person("Sue", "Foley")


print(objP1.strFirstName, objP1.strLastName)
print("-------------")
print(objP2.strFirstName, objP2.strLastName)
```

FIGURE 8.3 Lab8-3 script code constructor with parameters

See **Figure 8.3.1** below is the output of the script above for Lab 8-3. The output is the same as for Lab 8-2, but now it uses attributes instead of fields to set value of the “person” objects.

```
c:\_PythonClass\DemoCode8>python.exe C:\_PythonClass\DemoCode8\Lab_8-3_MBruce.py
Bob Martley
-------------
Sue Foley
```

FIGURE 8.3.1 Lab8-3 output in cmd shell

Properties and Abstraction

Properties are special functions used to manage attribute data in a class. Normally there are two properties for each field/attribute, one for "getting" data and one for "setting data. They are called "Getters" and "Setters". It is considered a best practice to only work with the data in a class through a Method or Property. This practice creates a layer of "Abstraction" and protects software using your class from internal changes to the Fields or Attributes. Creating private attributes adds to the level of abstraction and protects data from unintended changes from calls outside the class.

Lab 8-4 Hidden attributes

For Lab 8-4, I modified the Person class from Lab 8-3 to use private attributes, with a getter and setter property for each. See **Figure 8.4** for Lab8-4 code example.

```
# ------------------------------------------------ #
# Title: Lab_8-4_MBruce.py
# Description: A class constructor + Getter + Setter
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# MBruce,8.26.2022, added last name,constructor, getter, setter
# ------------------------------------------------ #

#--- Make the class ---
class Person:
    # --Fields--
    # -- Constructor --
    def __init__(self, first_name, last_name):
        # -- Attributes --
        #self.__strFirstName = first_name
        #self.__strLastName = last_name
        self.first_name = first_name          # Now the constructor calls on the setter prop function
        self.last_name = last_name            # Now the constructor calls on the setter prop function


    # -- Properties --
    # Property First Name
    @property
    def first_name(self):
        return str(self.__strFirstName).title() # private property getter for strFirstName attribute
                                                 # with convert to titlecase
    @first_name.setter                          # setter property for first name
    def first_name(self, value):
        if str(value).isnumeric() == False:     # check if numeric value
            self.__strFirstName = value         # set first name to passed value
        else:
            raise Exception ("Names cannot be numbers")

    #Property Last Name
    @property
    def last_name(self):
        return str(self.__strLastName).title() # private property getter for strLastName attribute
                                                # with convert to titlecase
    @last_name.setter                           # setter property for last name
    def last_name(self, value):
        if str(value).isnumeric() == False:     # check if numeric value
            self.__strLastName = value          # set last name to passed value
        else:
            raise Exception ("Names cannot be numbers")


    # -- Methods --
# End of class

# --- Use the class ----

objP1= Person("Bob", "Marley")              #uses property name in constructor
print(objP1.first_name, objP1.last_name)    # Print it
print("-------------")
objP2= Person ("Sue", "Foley")             #uses property name in constructor
print(objP2.first_name, objP2.last_name)    # Print it
```

FIGURE 8.4 Lab8-4 script code for functions to save

Below in **Figure 8.4.1** is the cmd shell output of lab8-4. for Lab8-4 output. Note that even though the first and last name fields are getting set by different elements of the class, the output remains the same.

```
c:\_PythonClass\DemoCode8>python.exe C:\_PythonClass\DemoCode8\Lab_8-4_MBruce.py
Bob Marley
-------------
Sue Foley
```

FIGURE 8.4.1 Lab8-4 output to cmd shell

Methods

Functions that organize actions and statements into groups inside the class are called “methods”. They behave much like functions is scripts. The __str_() method is a built-in invisible method that is used to return class data as a string

Lab 8-5 Class to hold data

For Lab 8-5, I modified the Person class from Lab 8-4 and did an override of the "__str__()" method to it would print first and last name fields separated by a comma. See **Figure 8.5** below.

```
# ------------------------------------------------ #
# Title: Lab_8-5_MBruce.py
# Description: A class constructor + Getter + Setter
# ChangeLog: (Who, When, What)
# RRoot,1.1.2030,Created Script
# MBruce,8.31.2022, added str method override
# ------------------------------------------------ #

#--- Make the class ---
class Person:
    # --Fields--
    # -- Constructor --
    def __init__(self, first_name, last_name):
        # -- Attributes --
        #self.__strFirstName = first_name
        #self.__strLastName = last_name
        self.first_name = first_name          # Now the constructor calls on the setter prop function
        self.last_name = last_name            # Now the constructor calls on the setter prop function


    # -- Properties --
    # Property First Name
    @property
    def first_name(self):
        return str(self.__strFirstName).title() # private property getter for strFirstName attribute
                                                 # with convert to titlecase
    @first_name.setter                          # setter property for first name
    def first_name(self, value):
        if str(value).isnumeric() == False:     # check if numeric value
            self.__strFirstName = value         # set first name to passed value
        else:
            raise Exception ("Names cannot be numbers")

    #Property Last Name
    @property
    def last_name(self):
        return str(self.__strLastName).title() # private property getter for strLastName attribute
                                               # with convert to titlecase
    @last_name.setter                          # setter property for last name
    def last_name(self, value):
        if str(value).isnumeric() == False:    # check if numeric value
            self.__strLastName = value         # set last name to passed value
        else:
            raise Exception ("Names cannot be numbers")


    # -- Methods --

    def __str__(self):                         # new str method to override default str method
        return self.first_name + "," + self.last_name

# End of class

# --- Use the class ----

objP1= Person("Bob", "Marley")              #uses property name in constructor
print(objP1)                                # Print it
print("-------------")
objP2= Person ("Sue", "Foley")             #uses property name in constructor
print(objP2)                                # Print it
```

FIGURE 8.5 Lab8-5 script code override of str method.

Below in **Figure 8.5.1** is the cmd shell output which utilizes the new str override method. If there was no override, then the default str() method would have printed out a sting containing the calling module "main", and the name of the object, and its memory address. Not to useful if you want to see the object's data printed out.

```
c:\_PythonClass\DemoCode8>python.exe C:\_PythonClass\DemoCode8\Lab8-5_MBruce.py
Bob,Marley
-------------
Sue,Foley
```

FIGURE 8.5.1 Lab8-5 output to cmd shell

Static Methods

An "@staticmethod" decorator is placed before a class's method definition if that method is going to be called directly. There is NO "self" keyword used in the method. There is no need to create an instance (object) of the class when calling methods directly. These are called static methods. A direct method call for a method "first_name" in "Person" class from main might look like this:

Person.first_name("bob")

The other type of method is an instance method. This does NOT have a static decorator and uses the keyword "self", since it is used with instances(objects) of the class. The same call above to an instance method version would look like this:

```
per1= Person()
print( per1.first_name("bob"))
```

Normally a class consists of one type of methods or the other based on its purpose, but generally not both.

DocStrings

Doc strings for a class are used just as they are for a function. They can be utilized in IDE tooltips or shown using the built-in "__doc__"property called in association with the class. Example:

print (Person.__doc__)

Git and GitHub desktop

Git is a distributed code version control system that manages code changes, provides backups and allows dev teams to work in "branches" in a trunk-based development environment. GitHub is a web-based hosted code repository where code is stored in projects. Code is clones (snapshots of current state) and downloaded to local dev's machine for work.

Assignment 08

The assignment this week is to modify the starter script to add three classes with various methods to handle the work in the application and include error handling.

Create Script

1) Opened PyCharm and created folder named "Assignment08"
2) Created a new project **Assignment08.py** at:
   C:/_PythonClass/Assignment08/ **Assignment08.py**
3) Used **"Assigment08-Starter.py"** as a starter

Class Product

This is the first of the three classes to be modified and it will store data about products. Since it is storing data, this will be a data class and have instance methods using the "self" keyword.

Figure 8.6 shows the code for ***class Product()*** constructor and properties to get and set the class attributes "product_name" and "product_price". Plus, there are two str() methods to override the default str().

```
    # -- Constructor --
    ##############################################
    def __init__(self, product, price):
        self.product_name = product             # uses the property to create attribute for product_name
        self.product_price = price              # uses the property to create attribute for product_price
    # -- Properties --
    ##############################################
    # product_name
    @property
    def product_name(self):                                 # Getter property for product_name
        return str(self.__product_name).title()             # converts to title case and returns product_name

    @product_name.setter
    def product_name(self, productX):                       # Setter property for product_name
        if str(productX).isnumeric() == False:              # Check if entry is numeric
            self.__product_name = productX                  # If string, assign name to attribute
        else:
            raise Exception("Names cannot be numbers")      # Else inform user entry in invalid

    # product_price
    @property
    def product_price(self):                                # Getter property for product_price
        return str(self.__product_price)                    # returns product_price

    @product_price.setter
    def product_price(self, priceX):                        # Setter property for product_price
        try:
            if float(priceX):                               # Check if entry is float
                self.__product_price = float(priceX)        # If float, assign price to attribute
        except ValueError:
            raise Exception("Price needs to be a number")   # Else inform user entry in invalid

    # -- Methods --
    ##############################################
    def to_string(self):
        return self.__str__()                                   # Overrides defualt str() method and converts class data to string

    def __str__(self):
        return self.product_name + ", " + self.product_price    # Returns product_name and product price as a string
```

FIGURE 8.6 Assignment8 code for class Product() constructor, properties and str method overrides.

Class FileProcessor

This class is a "processing" data class and is therefore going to be called directly and will contain static methods with "@staticmethod" decorators before each method definition. Three methods were defined in this class to handle reading and writing list data from/to a file, in addition to adding data to the list of products.

In *Figure 8.7* below, the code for the static method that handles reading data from file is shown.

```
# TODO: Add Code to process data from a file
@staticmethod
def read_data_from_file(file_name):                    # reads a list of product objects from file
    """ Reads a list of product objects from a file

        :param file_name: (string) name of file to open and read
        :return: list_of_products: (list) returns list of products read from file
    """

    list_of_products=[]                                # Initialize list of product to empty set
    try:
        file = open(file_name, "r")                    # Try to open file to read list of products
        for line in file:                              # loop through each line in file
            item = line.split(",")                     # separate the product and price items in each line
            row = Product(item[0],item[1])             # assign each pair of product and price to a row
            list_of_products.append(row)               # append it to the list
        file.close()
    except IOError:                                    # If file was not found, then except clause executes and informs the user.
        print("ERROR...cannot locate", file_name, "file.")
    return list_of_products                            # returns list of product objects
```

FIGURE 8.7 Assignment8 code for class FileProcessor and method “read_data_from_file()”

In *Figure 8.8* below, the code for the static method that handles adding more product object data to the is listed.

```
#########################################################
# Added additional method #
@staticmethod
def add_data_to_list(product, price, list_of_products):
    """ Adds new data to a list of product objects

        :param product: (string) name of product:
        :param price: (float) price of product:
        :param list_of_products: (list) the list of products:
        :return: (list) appended list of products
    """
    p= Product(str(product).strip(), float(price))       # create a new product object with new product name and price
    list_of_products.append(p)                           # append the new product instance to the list


    return list_of_products                              # return the appended list of products
#########################################################
```

FIGURE 8.8 Assignment8 code for class FileProcessor and method “add_data_to_list()”

In ***Figure 8.9*** the code for the static method that handles writing to the file.

```
# TODO: Add Code to process data to a file
@staticmethod
def save_data_to_file(file_name, list_of_product_objects):
    """ Saves a list of product objects to file

        :param file_name: (string) name of file
        :param list_of_product_objects: (list) a list containing product objs
        :return: write_status: (bool) return true or false based if it successfully wrote to file
    """
    write_status = False                        # initialize write status to False
    try:
        objFile = open(file_name, "w")          # Open file for write
        for row in list_of_product_objects:     # Loop through rows in list
            objFile.write(str(row) + "\n")      # Write each row to file
        objFile.close()
        write_status =True                      # set the status to true- It Worked!!!
    except Exception as e:                      # Inform user if could not open file
        print("Could not write to file")
    return write_status                         # Need to return if the write to file was successful or not


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

FIGURE 8.9 Assignment8 code for class FileProcessor and method "save_data_to_file()"

Class IO

This class is "presenting" data and is therefore going to be called directly and will contain static methods with "@staticmethod" decorators before each method definition. This class handles the methods that shows the menu to the user, takes in the user's menu choice, outputs the current list and takes in new product and price to be added to list of products. See ***Figure 8.10*** for the static method IO.output_menu.

```
# TODO: Add code to show menu to user
@staticmethod
def output_menu():
    """  Display a menu of choices to the user

        :return: nothing
    """
    print('''
        Menu of Options
        ---------------------------
        1) Show current data in list
        2) Add a new product to list
        3) Save list to File
        4) Exit Program
        ''')
    print()  # Add a line to beautify
```

FIGURE 8.10 Assignment8 code for Class IO output_menu()

The code section to taking in the user's menu choice is handled by static method ***IO.input_menu_choice()*** and is shown in ***Figure 8.11*** below.

```
# TODO: Add code to get user's choice
@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 a line to beautify
    return choice
```

FIGURE 8.11 Assignment8 code - static method ***IO.input_menu_choice()***

The code section to output the current list is handled by static method ***IO.output_current_list()*** and is shown in ***Figure 8.12*** below. It has one parameter which is the list of product objects.

```
# TODO: Add code to show the current data from the file to user
@staticmethod
def output_current_list(list_of_rows):
    """ Shows the current products and their prices in the list of Products

        :param list_of_rows: (list) of rows you want to display
        :return: nothing
    """
    print("******* The current product list is: *******")
    for row in list_of_rows:
        print(row.product_name + " -  $" + row.product_price)
    print("*******************************************")
    print()  # Add a line to beautify
```

FIGURE 8.12 Assignment8 code- static method IO.output_current_list()

To add a new product to the list, static method IO.input_new_product_and_price() is called and returns a string and a float for product and price. This is passed to FileProcessing.add_new_data() (listed in ***Figure 8.8***) to complete adding the new product object to the product list. See ***Figure 8.13 below*** for the IO class method.

```python
# TODO: Add code to get product data from user
@staticmethod
def input_new_product_and_price():
    """  Gets product and price data to be added to the list

        :return: (string, string) with product and price
    """
    str_product = input("Please input a new product name: ")  # Prompt user for new product- assign to string "str_product"
    flt_price = input("Please input its price: ")             # Prompt user for price- assign to string "flt_price"

    return (str_product, flt_price)  # Return user entered product and price values
```

FIGURE 8.13 Assignment8 code- static method “input_new_product_and_price().

Main Method

With the three classes defined and associated methods, the main body code was constructed to use a while loop to show the user the menu of options and allow them to choose one. A try-except block was built around it to handle any script our file errors *Figure 8.14* shows the calls in Main.

```python
# Main Body of Script  ------------------------------------------------------ #
# TODO: Add Data Code to the Main body
# Load data from file into a list of product objects when script starts
try:
    lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)

    while (True):

        # Show user a menu of options
        IO.output_menu()

        # 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 in list
            IO.output_current_list(lstOfProductObjects)
            continue

        # Let user add data to the list of product objects
        elif choice_str.strip() == '2':  # add product to list
            str_product, flt_price = IO.input_new_product_and_price()
            lstOfProductObjects = FileProcessor.add_data_to_list(product=str_product, price=flt_price, list_of_products=lstOfProductObjects)
            continue  # to show the menu

        # let user save current data to file
        elif choice_str == '3':  # Save Data to File
            table_lst = FileProcessor.save_data_to_file(file_name=strFileName, list_of_product_objects=lstOfProductObjects)
            print("Data Saved!")
            continue  # to show the menu

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

except Exception as e:
    print("Dude, you got an file error!")
    print(e, e.__doc__, sep="\n")
# End of Main Body of Script  ----------------------------------------------- #
```

FIGURE 8.14 Assignment8 code- Main method

Here is the sequence of outputs as seen when the program is run in the cmd shell following each menu option selection in sequential order.

Figure 8.14 shows the initial display when program launches and user makes option choice #1 to display current list.

```
:\_PythonClass\Assignment08>python.exe C:\_PythonClass\Assignment08\Assigment08.py

           Menu of Options
           --------------------------
           1) Show current data in list
           2) Add a new product to list
           3) Save list to File
           4) Exit Program


Which option would you like to perform? [1 to 4] - 1

******* The current product list is: *******
Tea -  $1.99
Coffee -  $2.99
Milk -  $5.99
Sugar -  $3.99
*********************************************
```

FIGURE 8.14 Assignment8 code- Option #1 output

Figure 8.15 shows the output when option#2 to add a new item is chosen and then the current list is reprinted.

```
           Menu of Options
           --------------------------
           1) Show current data in list
           2) Add a new product to list
           3) Save list to File
           4) Exit Program


Which option would you like to perform? [1 to 4] - 2

Please input a new product name: chocolate
Please input its price: 2.55

           Menu of Options
           --------------------------
           1) Show current data in list
           2) Add a new product to list
           3) Save list to File
           4) Exit Program


Which option would you like to perform? [1 to 4] - 1

******* The current product list is: *******
Tea -  $1.99
Coffee -  $2.99
Milk -  $5.99
Sugar -  $3.99
Chocolate -  $2.55
*********************************************
```

FIGURE 8.15 Assignment8 code- menu option #2

Figure 8.16 shows the output when menu option # 3 is chosen to save the list to file.

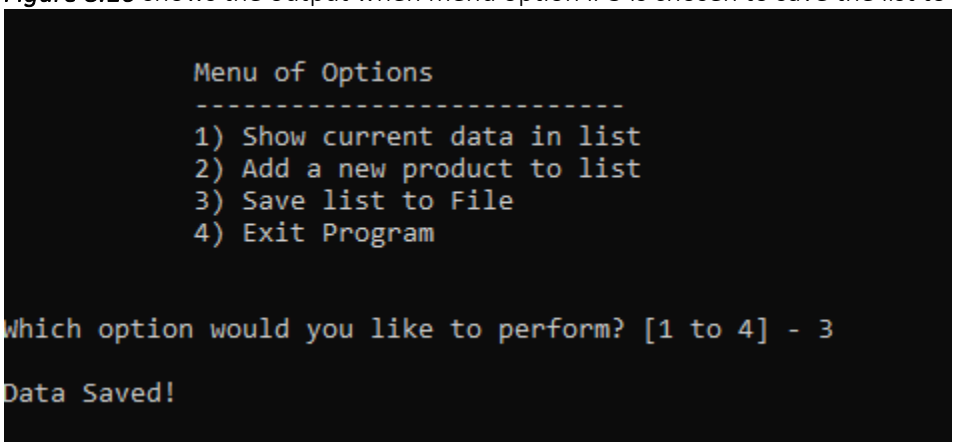

FIGURE 8.16 Assignment8 code- menu option #3

Figure 8.17 shows output when menu option #4 is chosen.

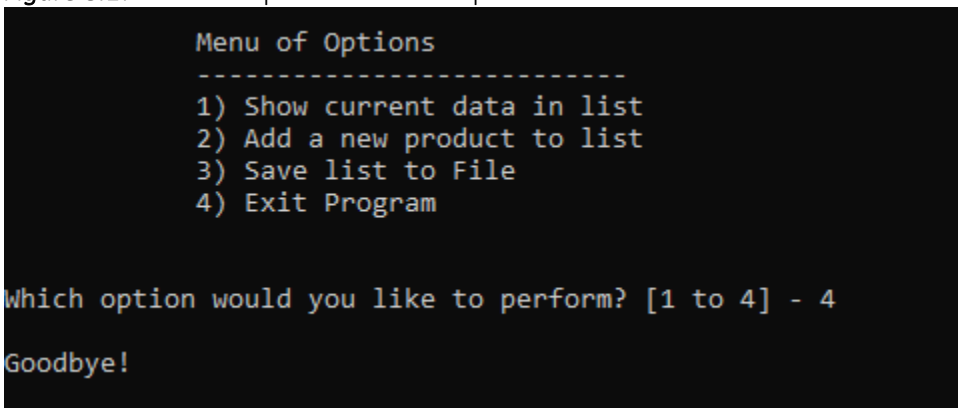

FIGURE 8.17 Assignment8 code- menu option #4

Summary

In the module08, it bought together everything we have learned on functions/methods, reading from and saving to files, error handling, variables and encapsulation and how they are incorporated within classes. This was a challenging assignment due to the efforts needed to plan and co-ordinate which classes and methods would handle the user actions and grouping them accordingly.

The TRY/BLOCK error handling used throughout the program was critical to help catch the many errors while troubleshooting. I still need to develop my skills more in debug mode in PyCharm to help work through issues more efficiently.