Ryan Lupinski
03.03.2022
Foundations of Programming: Python
Assignment 08
Github URL: https://github.com/ryanlupinski/IntroToProg-Python-Mod08

Objects and Custom Classes

1 Introduction

This write up will discuss using custom classes and creating objects. Python is a general-purpose object-oriented programming (OOP) language which uses functions to create objects. These functions are grouped together in classes and when a class is invoked, an object instance is created. The functions inside the class determine attributes and properties of the instantiated object and allow for other functions to get or set these attributes using methods. This write up will put these aspects of Python to work by creating a program that allows the user to pull product and price data from a text file into a list of objects, add new product objects to the list, write/read to/from the file, and finally exit.

2 Objects

"An Object is an instance of a Class. A class is like a blueprint while an instance is a copy of the class with actual values." (Geeks For Geeks, https://www.geeksforgeeks.org/python-object/, 03,2022) (External Link). Objects can be created by calling a class and setting it equal to some variable. See listing 1. The object 'objectXYZ' is instantiated by calling ClassABC(). The class has one method called test(self) that prints the string "Testing". This special function is invoked using dot notation after the object name. Finally, the type of the object is printed, proving an object was created.

```
class ClassABC:
    def test(self):
        print("Testing")
        return


objectXYZ = ClassABC()

objectXYZ.test()
print(type(objectXYZ))

----------------------------------------

/usr/local/bin/python3.9 "/Users/ryanlupinski/Library/Application
Support/JetBrains/PyCharmCE2021.3/scratches/scratch_6.py"
Testing
<class '__main__.ClassABC'>

Process finished with exit code 0
```

Listing 1. Creating a simple object

Creating objects is very useful compared to creating many individual variables that represent different properties or attributes of an object. The custom class allows all the objects created with that class to function the same and allow the user to access or set these properties or attributes the same way every time.

3 Classes

3.1 General Structure

The general structure of a class is as follows in listing 2. MyClassName is the custom class name that follows the class designator. Classes can reference base classes such as strings, lists, tuples, etc. This write up will often omit putting the base class, as it is obvious (and done automatically by Python) that that base class will be object.

```
class MyClassName(MyBaseClassName):

    # -- Fields --
    # -- Constructor --
    #     -- Attributes --
    # -- Properties --
    # -- Methods --
```

Listing 2. Basic structure of a class

3.2 Components of the Class

Fields are the variables and constant data in a class. Fields can hold default values for objects if they are not given to the class as a parameter or constants associated with objects.

Constructors are special functions that automatically run when the object is created from the class. Constructors are run on the initial invocation and are often used to set the initial values in the field section.

Attributes are characteristics of the object. For instance, if a class called Person was created, each time an object was created, the attributes could set the first and last name of the person.

Properties are functions that manage attributes. For instance, if you created an object using the Person class, you may set their first and last name as an attribute. If the person wants to change their first name, a function in the property section could be called and the value could be changed. There are 'setter' and 'getter' properties for when an attribute is to be changed or retrieved, respectively.

Methods are functions that perform an action on the object. The product list uses a method to write data a file or read data from the file.

4 Product list program

The product list program uses five classes, Product, FileProcessor, IO, and two exception classes for handling errors in the naming or price input from the user. The product list program begins with creating a list of product objects using the read_data_from_file method within the FileProcessor class. See listing 3. This is invoked from the script using the following code: lstOfProductObjects = FileProcessor.read_data_from_file(strFileName). The file name is passed to the method within the class and data is loaded from the file into memory.

```
@staticmethod
def read_data_from_file(file_name):  # read data from file
    lstOfProductObjects.clear()  # clear list of all data
    try:  # check to make sure the file exists
        with open(file_name, 'r') as file:  # open file in read mode
            for line in file:  # for each line in file
                product_name, product_price = line.split(",")  # split each
line and save to vars
                product = Product(product_name=product_name,
product_price=product_price)  # make product list
                lstOfProductObjects.append(product)  # append each product to
list of product objects
            return lstOfProductObjects  # return the list of product objects
    except:  # if the file doesn't exist
        print("File \'product.txt\' not found")
        print("Creating a new file")
        with open(file_name, 'w') as file:  # create a new file
            file.write("Product,Price\n")  # write header to file
```

Listing 3. Method to read data from the file

The method checks to see if the products.txt file exists and creates a new file with a header if it doesn't exist. Next, the program enters a while loop to present the current list of products and their price to the console, prints a menu of options to the console, and waits for user input.

Printing the menu and capturing user option input have been covered previously and will not be discussed. The program has four options: add new product to list, save product list to the file, load product list from the file into memory, and exit.

If the user wants to add a new product, a new product object is created using the Product class.

```
if strChoice.strip() == '1':  # Let user add data to the list of product
objects
    try:
        strNewProduct = str(input("Add a Product: "))  # prompt user to add a
Product
        if strNewProduct.isnumeric():  # if name is numeric
            raise ProductNameError()  # raise custom error
        strNewPrice = str(input("Add Price: ") + "\n")  # prompt user to
define Price
        if strNewPrice.strip().isalpha():  # if price is alphanumeric
            raise ProductPriceError()  # raise custom error
        objNewProduct = Product(product_name=strNewProduct,
product_price=strNewPrice)  # create new object
```

Listing 4. Adding a new product.

The user is prompted to enter a product name and price. If the name is numeric, or the price is not numeric, it will raise the custom error in the exception class. If there are no user input errors, a new object objNewProduct is created by calling the Product class, and passing the strNewProductName and strNewPrice string variables to the class. This instantiates the new product object.

```
class Product(object):
    """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
        RLupinski,02.26.2022,Modified code to complete assignment 8
        RLupinski,02.27.2022, setter/getter clean up
    """

    # -- Fields --
    # -- Constructor -- (special method that auto runs on instantiation)
    def __init__(self, product_name, product_price):
        # -- Attributes --
        self.__product_name = product_name  # sets product obj name
        self.__product_price = product_price  # sets product obj price

    # -- Properties -- (functions used to manage field attributes.
setter/getter)
    # product_name
    @property
    def product_name(self):  # (getter or accessor)
        return str(self.__product_name).title()  # returns uppercase product
name
```

Listing 5. Product class

When the new name and price are passed to the Product class, the constructor initializes the name a price attributes. The properties section defines a function to return to set these values directly (only product_name property name is shown for space reasons). Next, the product is added to the list using IO.add_product_to_list(objNewProduct, lstOfProductObjects)

```
@staticmethod
def add_product_to_list(new_product, list_of_products):  # add product to
list
    list_of_products.append(new_product)  # append list of product w/ new
product
    return list_of_products  # return list of products
```

Listing 6. Appending the list with a new object

The next two options allow the user to save the current list to the file or load the data from the file into memory.

```
elif strChoice == '2':  # let user save current data to file and exit program
    strChoice = input("Do you want to save the product list to the file?\n"
                      "Continue: (y/n)")  # option to abort saving to file
    if strChoice == 'y':
        FileProcessor.save_data_to_file(strFileName, lstOfProductObjects)  #
save data to file
        print(f"Data saved to {strFileName}!")
    else:
        print("Save aborted")
    IO.input_press_to_continue()  # hold for user

elif strChoice == '3':
    strChoice = input("WARNING: THIS WILL OVERWRITE UNSAVED DATA\n"
                      "Continue: (y/n)")  # option to abort loading from file
    if strChoice == 'y':
        lstOfProductObjects = FileProcessor.read_data_from_file(strFileName)
# load data from file
        print(f"Data from {strFileName} loaded!")
        IO.input_press_to_continue()  # hold for user
    else:
        print("Data load aborted")
        IO.input_press_to_continue()  # hold for user
```

Listing 7. Save / Read data options

The FileProcessor class has two methods to save data or to read data.

```
@staticmethod
def save_data_to_file(file_name, list_of_product_objects):  # save data to
file
    file = open(file_name, "w+")  # open connection to file in write mode +
    for row in list_of_product_objects:  # for each row in list of objects
        file.write(str(row.product_name) + "," + str(row.product_price))  #
write object to file
    file.close()  # close connection
```

Listing 8. Save / Read data methods

The file name and the list of product objects are passed to the save_data_to_file method. A connection to the file is opened in write + mode and each row is written to the file using the write method. The method shows product_name and and product_price, the attributes from the product class, being called to write the product data to the file. Reading data from the file is shown above. Each options gives the user the chance to abort to save or read action.

The last option allows the user to exit the program, with a final prompt to abort exiting if data has not been saved.

Figure 1. Option 1, adding a product and price to the list in memory

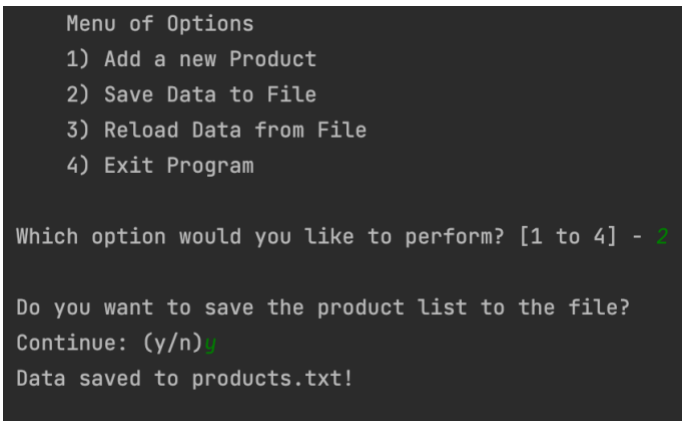

Figure 2. Saving the updated list to the file.

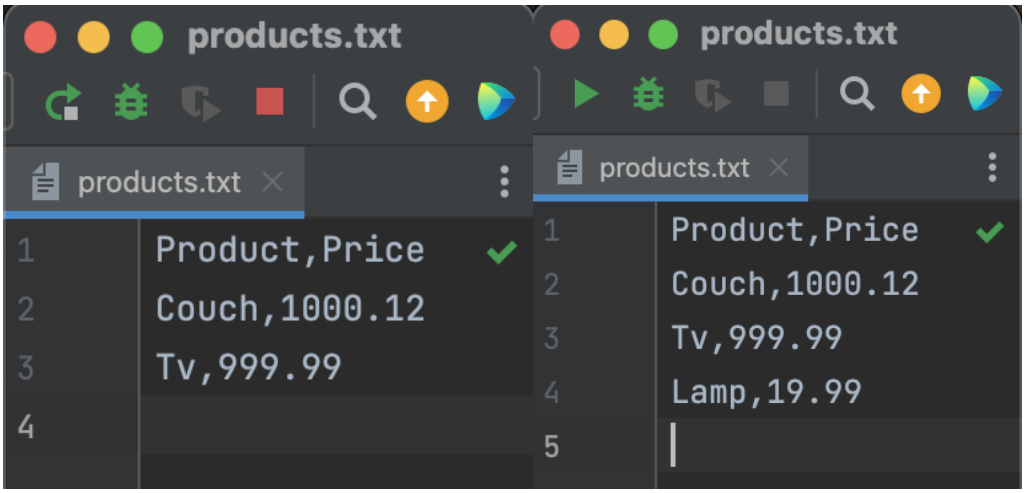

Figure 3 Before and after the data is saved to the file.

5 Summary

Objects are classes are import components of all medium to high complexity Python programs. The creation of objects from classes is critical in creating and modifying attributes of individual objects via the use of properties. Methods in a class are useful for performing special functions on the object, or for processing functions like saving data to a file or reading data from a file.