Objects and Classes

# Introduction

For the eighth assignment, the concept of **object oriented programming** **(OOP)** was explored. OOP is a different way of thinking in programming where the basic building block is a **software object**. These objects allow one to create an object instance that can be repeated throughout the running of a script. These objects combine characteristics (**attributes**) and behaviors (**methods**). These objects are held within classes. Overall, these classes organize the objects to allow easy access to properties, attributes and methods within themselves or in a main script. Finally, I built on my experience of creating a Github webpage by installing GitHub desktop and learning how **Git software** manages versions of one or more files online and.

# Objects versus Classes

**Objects** are instances when code is used indirectly while its variables are utilized with commands. In contrast, classes can not use objects and reference code directly. An advantage of using the code indirectly is that one can have multiple object instances, each with a different address in memory. The data for each instance is kept separate for each object and each object would hold data about a different customer. A class is used directly to process data and indirectly if its focus is to store data. For further clarification, classes have fields, constructors, properties and methods. The data in classes can be thought of as being like rows of data. These “rows” are often collected into a list object or table of data.

# Fields and Attributes

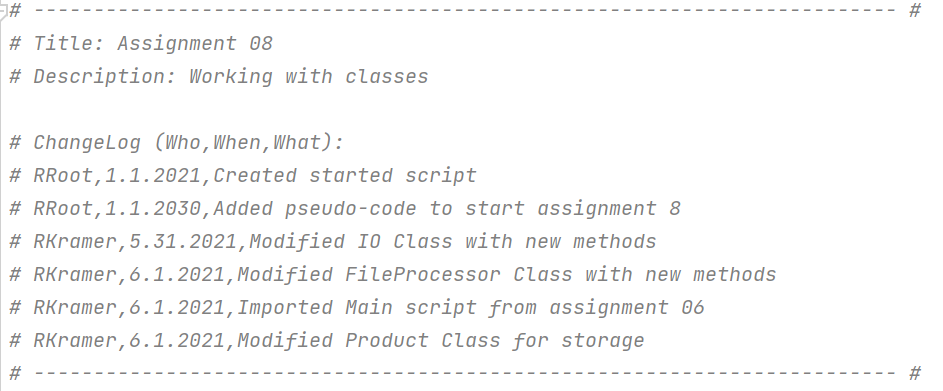
**Fields** are the data members of a class. They are created using variables and constants. Each object instance can hold data unique to that single object. **Attributes** are virtual fields that hold internal data for a class.

# Methods and Functions

**Methods** are functions that organize processing statements into named groups like functions do in scripts. Examples include the “\_\_str\_\_” method that turns some or all of a class’s data in to a string. A static method allows methods to be called directly from a class while an instance method uses the self keyword to refer to data or functions found in an object instance but not directly in the class. A private method is used for internal processing only and can be used to track how many objects are currently created from a class. One special method is a **constructor**. They automatically run when you create an object form the class. Constructors are often used to set the initial values of field data. They use the double underscore format in Python. Python automatically calls the method and passes any arguments through the method each time a new object is created. **Destructors** are methods that automatically run when an object instance is removed from memory. They clean up any resources that are not needed once the object is gone yet Python has most of its resources self-clean. **Properties** are functions used to manage field or attribute data. Two properties frequently used are the **Setter** and **Getter**. The **setter** lets one add code for both validation and error handling. If a value passed into the properties parameter is valid, it is assigned to the field or attribute. A getter property lets one add code to format a field or attribute’s data. They are often include even there is no formatting code. For best practice, one should work with data in a class only through a method or property. This creates a layer of abstraction and protects software from making unintentional internal changes to fields or attributes.

# Code Outlining

My example code follows the same logic as Assignment 6’s script in that the user opens a file automatically when they run the script, albeit this time it’s recording products and price instead of tasks and priority and data is stored in the products class. The user is still prompted to either add or delete data, upload their changes to the file, reload the file or exit the program while the Product class serves as a storage place much like the .txt file. As much of the code stayed the same, I will focus only on the areas where the use of class features was found and utilized. Starting from the script header (Figure 1), one can see I imported the IO class from the previous assignment and modified the inputs and outputs for the new data categories. The FileProcessor Class was then updated with the methods from the previous assignment. The main script was imported. Finally, the Product class was created systematically with the main script and other classes adapted to reference it when necessary.



**Figure 1. Script Header**

The Input / Output function stayed relatively the same besides the modifications from Task and Priority to Product and Price. The File Processor Class (Figure 2) took values from the input and output class as well as the Product class in order to be consistent in processing the data. The input data from the IO Class is still stored in a list that is directly referenced but this time the Product Class is also referenced with the FileProcessor Class.

*# Processing ------------------------------------------------------------- #***class** FileProcessor(Product):  
 *"""Processes data to and from a file and a list of product objects:  
  
 methods:  
 save\_data\_to\_file(file\_name, list\_of\_product\_objects):  
  
 read\_data\_from\_file(file\_name): -> (a list of product objects)  
  
 changelog: (When,Who,What)  
 RRoot,1.1.2021,Created Class  
 RKramer,6.1.2021,Modified code to complete assignment 8  
 """* @staticmethod  
 **def** save\_data\_to\_file(file\_name, lstOfProductObjects):  
 *""" Writes product data from list of rows to file* **:param** *file\_name: (string) with name of file being overwritten:* **:param** *lstOfProductObjects: (list) you read from to fill file:* **:return***: (list) of dictionary rows  
 """* file = open(file\_name, **'w'**)  
 **for** row **in** lstOfProductObjects:  
 file.write(row[**"Product"**] + **", "** + row[**"Price"**] + **"\n"**)  
 file.close()  
 **return** lstOfProductObjects, **'Success'** @staticmethod  
 **def** read\_data\_from\_file(file\_name, lstOfProductObjects):  
 *""" Reads data from a file into a list of dictionary rows* **:param** *file\_name: (string) with name of file:* **:param** *lstOfProductObjects: (list) you want filled with file data:* **:return***: (list) of dictionary rows  
 """* lstOfProductObjects.clear() *# clear current data* file = open(file\_name, **"r"**)  
 **for** line **in** file:  
 product\_name, product\_price = line.split(**","**)  
 row = {**"Product"**: product\_name.strip(), **"Price"**: product\_price.strip()}  
 lstOfProductObjects.append(row)  
 pr = Product(product\_name,product\_price)  
 file.close()  
 **return** lstOfProductObjects, **'Success'** @staticmethod  
 **def** add\_data\_to\_list(product\_name, product\_price, lstOfProductObjects):  
 *""" Adds data to a list of dictionary rows* **:param** *product\_name: (string) with name of product:* **:param** *product\_price: (string) with price:* **:param** *lstOfProductObjects: (list) you want filled with file data:* **:return***: (list) of dictionary rows  
 """* row = {**"Product"**: str(product\_name).strip(), **"Price"**: str(product\_price).strip()}  
 lstOfProductObjects.append(row)  
 **return** lstOfProductObjects, **'Success'  
  
 def** remove\_data\_from\_list(product\_to\_remove, lstOfProductObjects):  
 *""" Removes product data and associated price from list of dictionary rows* **:param** *product\_to\_remove: (string) with name of product to be removed:* **:param** *lstOfProductObjects: (list) you want filled with file data:* **:return***: (list) of dictionary rows  
 """* **for** row **in** lstOfProductObjects:  
 **if** row[**"Product"**].lower() == product\_to\_remove.lower():  
 lstOfProductObjects.remove(row)  
 *#print("row removed")* **return** lstOfProductObjects, **'Success'***# Processing ------------------------------------------------------------- #*

**Figure 2. FileProcessor Class**

The data variables and constants as well as the Product Class were established in the data pseudocode section (Figure 3). The product class stored the inputs with constructor methods. Getter and setter properties were used to handle errors when inputs did not match their requirements. Finally, methods better combined the data for storage and before they were uploaded to the file.

*# Data -------------------------------------------------------------------- #  
# Declare variables and constants*file\_name = **"products.txt"** *# The name of the data file*objFile = **"products.txt"** *# An object that represents a file*row = {} *# A row of data separated into elements of a dictionary {Product,Price}*list\_of\_rows = [] *# A list that acts as a 'table' of rows*strChoice = **""** *# Captures the user option selection*strProduct = **""** *# Captures the user product data*strPrice = **""** *# Captures the user price data*strStatus = **""** *# Captures the status of an processing functions*lstOfProductObjects = [] *# A list that acts as a 'table' of rows***class** Product(object):  
 *"""Stores data about a product:  
  
 properties:  
 product\_name: (string) with the products's name  
 product\_price: (float) with the products's standard price  
 methods:  
 changelog: (When,Who,What)  
 RRoot,1.1.2021,Created Class  
 RKramer,6.1.2021,Added fields, constructor, attributes and properties  
 """  
 # -- Fields --  
 # Product =""  
 # Price = ""  
  
 # -- Constructor --* **def** \_\_init\_\_(self, product\_name, product\_price):  
 *# -- Attributes --* self.\_\_product = product\_name  
 self.\_\_price = product\_price  
  
 *# -- Properties --  
 # product* @property  
 **def** product(self): *# (getter or accessor)* **return** str(self.\_\_product).title() *# Title case* @product.setter  
 **def** product(self, value): *# (setter or mutator)* **try**:  
 **if** str(value).isnumeric() == **False**:  
 self.\_\_product = value  
 **else**:  
 **raise** Exception(**"Products cannot be numbers"**)  
 **except** Exception **as** e:  
 print(**"Prices can not be negative"**)  
 print(e)  
 **return** value  
  
 *# price* @property  
 **def** price(self): *# (getter or accessor)* **return** str(self.\_\_price).title() *# Title case* @price.setter  
 **def** price(self, value): *# (setter or mutator)* **try**:  
 **if** value >= 0:  
 self.\_\_price = value  
 **else**:  
 **raise** Exception(**"Prices cannot be negative"**)  
 **except** Exception **as** e:  
 print(**"Prices can not be negative"**)  
 print(e)  
 **return** value  
  
 *# -- Methods --* **def** \_\_str\_\_(self):  
 **return** self.\_\_str\_\_()  
  
 **def** \_\_str\_\_(self):  
 **return** self.product + **', '** + self.price  
 *# -- End of Class  
# Data -------------------------------------------------------------------- #*

**Figure 3. Product Class**

# Program Test

C:\Python\python.exe C:/\_PythonClass/Assignment08/Assigment08.py

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pen ($5)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

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

Product:Pencils

Price:$4

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pen ($5)

Pencils ($4)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 2

Which product would you like to remove:Pen

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pencils ($4)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pencils ($4)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 4

Warning: Unsaved Data Will Be Lost!

Are you sure you want to reload data from file? (y/n) - y

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pencils ($4)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

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

Goodbye!

Process finished with exit code 0

**Figure 4. Output from running Assignment08.py in pyCharm**

Microsoft Windows [Version 10.0.19041.985]

(c) Microsoft Corporation. All rights reserved.

C:\Users\robsk>CD C:\\_PythonClass\Assignment08\

C:\\_PythonClass\Assignment08>Python Assignment08.py

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pencils ($4)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

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

Product:Pens

Price:$5

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pencils ($4)

Pens ($5)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 2

Which product would you like to remove:Pencils

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pens ($5)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 3

Save this data to file? (y/n) - y

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pens ($5)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

5) Exit Program

Which option would you like to perform? [1 to 5] - 4

Warning: Unsaved Data Will Be Lost!

Are you sure you want to reload data from file? (y/n) - y

Press the [Enter] key to continue.

\*\*\*\*\*\*\* The current Products are: \*\*\*\*\*\*\*

Pens ($5)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Menu of Options

1) Add a new Product

2) Remove an existing Product

3) Save Data to File

4) Reload Data from File

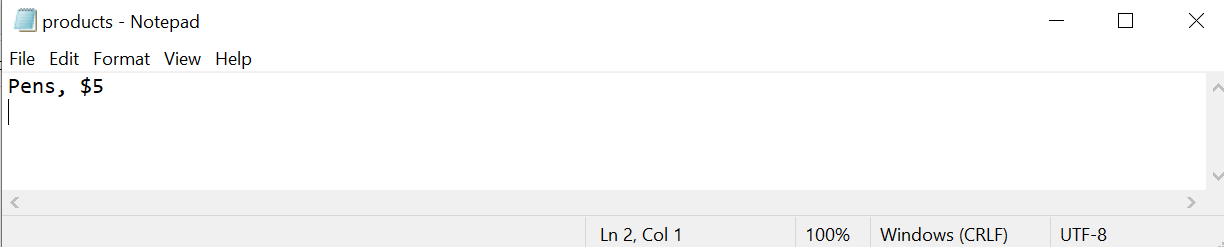
5) Exit Program

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

Goodbye!

C:\\_PythonClass\Assignment08>

**Figure 5. Output from running Assignment08 in the Command Prompt**



**Figure 6. Evidence of the text file’s save**

C:\\_PythonClass\Assignment05\ The program test was run through Pycharm (Figure 4) and through the command prompt (Figure 5). They both proved to be successful as the file was read and the program could display data, add to the list table, delete from the list table, save to the file and exit the program. A text file was created with the appropriate data loaded (Figure 6).

# Summary

This assignment was the most challenging so far as I tried to unpack all the perceived benefits of storing data in a class before having to implement it. The implementation required interconnectivity between the different classes and the main script of which I believe I took into account for the most part. The end product achieves the final goal of storing and processing the appropriate data. Finally, the Git exercise was useful in seeing the ease with how files can be uploaded, modified and tracked.