Hedy Khalatbari

August 21, 2021

Foundations of Programming: Python

Assignment08

GitHub URL:

Programming with Python: Module 8

# Introduction

This module covers classes in Python. Understanding this module requires some background in object-oriented programming – which I did not have. I started this Module by listening to the 90-minute recorded video. Contrary to the other modules, I did not pause the video multiple times and seek clarification and understanding from additional resources.

After finishing listening to the video once I reviewed the section on Classes in the Python 3 course in Codecademy. I then used several web references to understand the material better and subsequently went back to the video and solved the LAB exercises. My notes from the resources are in section Step 1B. I then went back and watched the Module 8 vidoe one more time, taking some notes and solving the lab exercises.

# Step 1A – Watch the video for Module 8

I watched the video but did not take any notes as the material was somewhat difficult to comprehend and I needed to read more broadly on this topic.

1. Classes: objects vs. classes. Classes may be used directly or indirectly
2. A standard class pattern:

**class MyClassName (MyBaseClassName)”**

**# -- Fields --**

**# --Constructor - -**

**# -- Attributes - -**

**# -- Properties - -** *# manage data in class*

**# -- Methods - -** *# a generic term for a function inside a class*

1. Fields: are data (variables or constants); you work with it indirectly or directly
2. Constructors: special methods that automatically run whenever you create an object; used to set initial values of fields. ‘**\_\_init\_\_**’. Only runs once when you create the object; can not be called later on

* constructors do not return a value

1. Destructors: historically, used to clean up resources that are not needed once object is gone. Python self-cleans and destructors are not used as commonly now. ‘**\_\_del\_\_**’
2. The self keyword: refers to the data found in the object instance. The keyword ‘this’ is used in other languages. Use the keyword ‘self’ in any methods that will used by an object instance.
3. Attributes: are a virtual (invisible) field. The variables are not initially declared in ‘field’ but are set in the ‘constructor’. This is not common in other languages.

* fields and attributes are variables and Python accepts any type of input. You can control the type of input accepted using ‘properties’

1. Properties: functions used to manage field or attribute data; getters and setters (accessors and mutators)

* @name\_of\_method.setter *# performs a validation to ensure the data input is clean*

def name\_of\_method(self, attribute\_name):

…

* @property *# typically formats (e.g., changing to lower case)*

def name\_of\_method(self):

…

* best practice to work with data in a class only through properties (i.e., **abstraction**)
* force the usage of properties by making fields and attributes **private**: use two underscores before the attribute’s name: ‘**\_\_**attribute\_name’
* note that the names of the getter and setter functions and the attribute must match

1. Methods: other than properties (which are used to manage field or attribute data) all other functions inside a class are called methods. Methods can be used directly are indirectly via object instances
2. the “\_\_str\_\_” method: returns some/all class’ data as a string. You can overwrite the default method and return more useful data (such as all attributes in a class)

* str(object\_name) will return the overwritten data
* object\_name.\_\_str\_\_()

1. **static methods**: @staticmethod directive is used to call methods directly from a class without making an object; do not use the keyword self in this situation. You will only have one copy of the class in memory. This is in contradistinction to the ‘***instance methods****’,* which also have the self keyword
2. private methods: use double underscores to indicate that a method is private
3. Type hints: typically used for function parameter and return types; not enforced by Python
4. DocStrings: put one in your classes (like what you did with functions); put it in triple double quotes. Can include description of properties and methods in it

* object\_name.\_\_doc\_\_
* class\_name.\_\_doc\_\_

1. Using Classes:
2. GitHub Desktop
3. GIT
4. GITHub Desktop

# LAB 8-1.Fields

|  |  |
| --- | --- |
| **Script** | **Running** |
| *# LAB8-3 08/21/2021 , fields*  **class** Person():  strFirstName = **''** strLastName = **''** person\_1 = Person() person\_1.strFirstName = **'Hedy'** person\_1.strLastName = **'Khalatbari'** print(person\_1.strFirstName) print(person\_1.strLastName) | C:\Python39\python.exe C:/\_PythonClass/ModDemos/LAB8-1.py  Hedy  Khalatbari  Process finished with exit code 0 |

# LAB 8-2. Constructors

|  |  |
| --- | --- |
| **Script** | **Running** |
| *# LAB8-2 08/21/2021 , constructors*  **class** Person():  strFirstName = **''** strLastName = **''   def** \_\_init\_\_(self, FirstName, LastName):  self.strFirstName = FirstName  self.strLastName = LastName   person\_1 = Person(**'Bob'**, **'Smith'**) person\_2 = Person(**'Sue'**, **'Jones'**)  print(person\_1.strFirstName, person\_1.strLastName) print(person\_2.strFirstName, person\_2.strLastName) | C:\Python39\python.exe C:/\_PythonClass/ModDemos/LAB8-2.py  Bob Smith  Sue Jones  Process finished with exit code 0 |

# LAB 8-3. Attributes

|  |  |
| --- | --- |
| **Script** | **Running** |
| *# LAB8-3 08/21/2021 , attributes* **class** Person():  *# strFirstName = ''  # strLastName = ''* **def** \_\_init\_\_(self, FirstName, LastName):  self.First\_Name = FirstName  self.Last\_Name = LastName   person\_1 = Person(**'Bob'**, **'Smith'**) person\_2 = Person(**'Sue'**, **'Jones'**)  print(person\_1.First\_Name, person\_1.Last\_Name) print(person\_2.First\_Name, person\_2.Last\_Name) | C:\Python39\python.exe C:/\_PythonClass/ModDemos/LAB8-3.py  Bob Smith  Sue Jones  Process finished with exit code 0 |

# LAB 8-4.

|  |  |
| --- | --- |
| **Script** | **Running** |
| *# LAB8-4 08/21/2021 , getters and setters  # --- Make the class ---* **class** Person(object):  *# --Fields--  # strFirstName = ""  # strLastName = ""    # -- Constructor --* **def** \_\_init\_\_(self, first\_name=**''**, last\_name=**''**):  *# -- Attributes --* self.\_\_first\_name = first\_name  self.\_\_last\_name = last\_name   *# -- Properties --  # FirstName* @property *# DON'T USE NAME for this directive!* **def** first\_name(self): *# (getter or accessor)* **return** str(self.\_\_first\_name).title() *# Title case* @first\_name.setter *# The NAME MUST MATCH the property's!* **def** first\_name(self, value): *# (setter or mutator)* **if** str(value).isnumeric() == **False**:  self.\_\_first\_name = value  **else**:  **raise** Exception(**"Names cannot be numbers"**)   *# LastName* @property *# DON'T USE NAME for this directive!* **def** last\_name(self): *# (getter or accessor)* **return** str(self.\_\_last\_name).title() *# Title case* @last\_name.setter *# The NAME MUST MATCH the property's!* **def** last\_name(self, value): *# (setter or mutator)* **if** str(value).isnumeric() == **False**:  self.\_\_last\_name = value  **else**:  **raise** Exception(**"Names cannot be numbers"**)   *# -- Methods --  # --End of class--  # --- Use the class ----* person\_1 = Person(**'Bob'**, **'Smith'**) person\_2 = Person(**'Sue'**, **'Jones'**)  print(person\_1.first\_name, person\_1.last\_name) print(person\_2.first\_name, person\_2.last\_name)  person\_1.first\_name = **'Robert'** *# set new first name* person\_2.first\_name = **'Jones-Smith'** *# set new last name* print(person\_1.first\_name, person\_1.last\_name) print(person\_2.first\_name, person\_2.last\_name) | C:\Python39\python.exe C:/\_PythonClass/ModDemos/LAB8-4.py  Bob Smith  Sue Jones  Robert Smith  Jones-Smith Jones  Process finished with exit code 0 |

# LAB 8-5. \_\_str\_\_()

|  |  |
| --- | --- |
| **Script** | **Running** |
| *# LAB8-5 08/21/2021 , attributes* **class** Person():  *# strFirstName = ''  # strLastName = ''* **def** \_\_init\_\_(self, FirstName, LastName):  self.First\_Name = FirstName  self.Last\_Name = LastName   **def** \_\_str\_\_(self):  **return** self.First\_Name + **','** + self.Last\_Name  person\_1 = Person(**'Bob'**, **'Smith'**) person\_2 = Person(**'Sue'**, **'Jones'**)  print(person\_1.First\_Name, person\_1.Last\_Name) print(person\_2.First\_Name, person\_2.Last\_Name) print(**'\nUsing string method:'**) print(str(person\_1)) print(str(person\_2)) print(**'\nUsing .\_\_str\_\_():'**) print(person\_1.\_\_str\_\_()) print(person\_2.\_\_str\_\_()) | C:\\_PythonClass\Assignment08\venv\Scripts\python.exe C:/\_PythonClass/ModDemos/LAB8-5.py  Bob Smith  Sue Jones  Using string method:  Bob,Smith  Sue,Jones  Using .\_\_str\_\_():  Bob,Smith  Sue,Jones  Process finished with exit code 0 |

# Step 1B – Classes: additional reading

1. source [9. Classes — Python 3.9.6 documentation](https://docs.python.org/3/tutorial/classes.html) (external site). These notes are direct quotes from the website

* ‘Classes provide a means of bundling data and functionality together. Creating a new class creates a new **type**of object, allowing new **instances** of that type to be made. Each class instance can have **attributes** attached to it for maintaining its state. Class instances can also have **methods** (defined by its class) for modifying its state.’
* ‘The class **inheritance** mechanism allows multiple base classes, a derived class can override any methods of its **base class** or classes, and a method can call the method of a base class with the same name. Objects can contain arbitrary amounts and kinds of data.’
  + ‘The method function is declared with an explicit first argument representing the object, which is provided implicitly by the call’
  + ‘Most *built-in operators* with special syntax (arithmetic operators, subscripting etc.) can be *redefined* for class instances’
  + ‘**Attribute references** use the standard syntax used for all attribute references in Python: obj.name. Valid attribute names are all the names that were in the class’s namespace when the class object was created’ (Box 1).

|  |
| --- |
| **Box 1- Class attributes: data attributes & methods** |
| **class** MyClass:  *"""A simple example class"""* i = 12345 *# MyClass.i is an attribute (a data attribute)* **def** f(self): *# MyClass.f is an attribute (a method)* **return 'hello world'** |

* ‘The **instantiation** operation (“calling” a class object) creates an empty object. Many classes like to create objects with instances customized to a specific initial state. Therefore, a class may define a special method named **\_\_init\_\_()**’ (Box 2).

|  |
| --- |
| **Box 2 – Class instantiation** |
| **class** MyClass:  i = 12345  new\_class = MyClass() *# class instantiation* **class** Complex:  **def** \_\_init\_\_(self, real\_part, imag\_part)  self.real = real\_part  self.imag = imag\_part  new\_complex = Complex(**'nerd'**, **'painter'**) *# class instantiation* print(new\_complex.real) *# prints 'nerd'* print(new\_complex.imag) *# prints 'painter'* |

* Instance objects understand two operations which are the two attribute references: data attributes and methods
* the instance object is passed as the first argument of the method
* **Instance variables** and **class variables**: ‘instance variables are for data unique to each instance and class variables are for attributes and methods shared by all instances of the class’
* Inheritance and multiple inheritance
* **Private variables**: ‘private instance variables that cannot be accessed except from inside an object don’t exist in Python. However, by convention a name prefixed with an underscore (e.g., \_spam) should be treated as a non-public part of the API (whether it is a function, a method, or a data member). It should be considered an implementation detail and subject to change without notice.’
* Private variables and **name mangling**: ‘Since there is a valid use-case for class-private members (namely to avoid name clashes of names with names defined by subclasses), there is limited support for such a mechanism, called name mangling. Any identifier of the form \_\_spam (at least two leading underscores, at most one trailing underscore) is textually replaced with \_classname\_\_spam, where classname is the current class name with leading underscore(s) stripped.’
* An empty class definition may be *used as a data type to bundle together a few named data*; for example, an employee’s name, department, and hire date my be bundled as attributes of an instance object (employee\_1.name, employee\_1.dept, and employee\_1.hire\_date)
* **Iterators**: ‘most container objects can be looped over with a **for statement**. Behind the scenes, the [for](https://docs.python.org/3/reference/compound_stmts.html#for) statement calls [**iter()**](https://docs.python.org/3/library/functions.html#iter) on the container object. The function returns an *iterator object* that defines the **method**[**\_\_next\_\_()**](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) which accesses elements in the container one at a time. When there are no more elements, [\_\_next\_\_()](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) raises a [StopIteration](https://docs.python.org/3/library/exceptions.html#StopIteration) exception which tells the for loop to terminate. You can call the [\_\_next\_\_()](https://docs.python.org/3/library/stdtypes.html#iterator.__next__) method using the [**next()**](https://docs.python.org/3/library/functions.html#next)**built-in function**’ (Box 2).
  + you can define the iter() and \_\_next\_\_() methods for a class

|  |  |
| --- | --- |
| **Box 3 – Iterators** |  |
| **Script** | **Running** |
| str\_test = **'iteration'** str\_test\_iteration = iter(str\_test) print((str\_test\_iteration))  counter = 0 **while** counter < len(str\_test):  print(next(str\_test\_iteration))  counter += 1 | <str\_iterator object at 0x000002493F572DC0>  i  t  e  r  a  t  i  o  n |

* **generators**:  ‘a simple and powerful tool for creating iterators. They are written like regular functions but use the [**yield**](https://docs.python.org/3/reference/simple_stmts.html#yield)**statement** whenever they want to return data. Each time[**next()**](https://docs.python.org/3/library/functions.html#next) is called on it, the generator resumes where it left off’. Generators have:
  1. automatic method creation: the [\_\_iter\_\_()](https://docs.python.org/3/reference/datamodel.html#object.__iter__) and [\_\_next\_\_()](https://docs.python.org/3/reference/expressions.html#generator.__next__) methods are created automatically
  2. saving program state: the local variables and execution state are automatically saved between calls.
  3. termination is automatic: when generators terminate, they automatically raise [StopIteration](https://docs.python.org/3/library/exceptions.html#StopIteration)
* **generator expressions**

1. source [Python - Object Oriented - Tutorialspoint](https://www.tutorialspoint.com/python/python_classes_objects.htm) (external site). Portions of these notes are that are direct quotes from the website are enclosed in quotation marks
2. **Object-oriented programming terminology**: summarized in Table1

|  |  |
| --- | --- |
| **Table 1- Object-oriented programming terminology**  *All definitions are direct quotes from* [*Python - Object Oriented - Tutorialspoint*](https://www.tutorialspoint.com/python/python_classes_objects.htm) *(external site)* | |
| **Terminology** | **Definition** |
| **Class** | A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance  variables) and methods, accessed via dot notation |
| **Class variable** | A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are |
| **Data member** | A class variable or instance variable that holds data associated with a class and its objects |
| **Function overloading** | The assignment of more than one behavior to a particular function. The operation performed varies by the types of objects or arguments involved |
| **Instance variable** | A variable that is defined inside a method and belongs only to the current instance of a class |
| **Inheritance** | The transfer of the characteristics of a class to other classes that are derived from it |
| **Instance** | An individual object of a certain class. An object obj that belongs to a class Circle, for example, is an instance of the class Circle |
| **Instantiation** | The creation of an instance of a class |
| **Method** | A special kind of function that is defined in a class definition |
| **Object** | A unique instance of a data structure that's defined by its class. An object comprises both data members (class variables and instance variables) and methods. |
| **Operator overloading** | The assignment of more than one function to a particular operator |

1. **Creating classes**

* class statement: class Class\_Name
* documentation string: accessed via *ClassName.\_\_doc\_\_*
* class variables
* class **constructor** or **initialization method**: *\_\_init\_\_()*
* class methods: declared like normal functions with the exception that the first argument to each method is *self*

1. **Creating instance objects**

* call the class using class name and pass in the arguments for its *\_\_init\_\_* method

1. **Accessing attributes**

* accessing the object's attributes: use the dot operator with object

*new\_complex.real*

* accessing the class variable: use the dot operator with class name and class variable

*Complex.complex\_count*

* function to access the attribute of and object:  **getattr**(obj, name[, default])
* function to check if an attribute exists or not: **hasattr**(obj,name)
* function to set (or create) an attribute: **setattr**(obj,name,value)
* function to delete an attribute: **delattr**(obj, name)

1. **Built-in class attributes**

* **\_\_dict\_\_** : dictionary containing the class's namespace
* **\_\_doc\_\_** : class documentation string (none, if undefined)
* **\_\_name\_\_** : class name
* **\_\_module\_\_** : module name in which the class is defined
* **\_\_bases\_\_** : a tuple containing the base classes, in the order of their occurrence in the base class list (may be empty if there are no base classes)

1. **Destroying objects (garbage collection)**

* ‘Python deletes unneeded objects (built-in types or class instances) automatically to free the memory space’
* *\_\_del\_\_()* method (aka, a destructor): a class can implement the delete method and the method is invoked when the instance is about to be destroyed. This method can also clean up any non-memory resources used by an instance

1. **Class inheritance**

* ‘can create a class by deriving it from a preexisting class by listing the parent class in parentheses after the new class name’
* ‘the child class inherits the attributes of its parent class, and you can use those attributes as if they were defined in the child class’
* ‘a child class can also override data members and methods from the parent’
* functions to check relationships of two classes and instances
* **issubclass(sub,sup)** function: a Boolean function. Returns true if the subclass *sub* is a subclass of the superclass *sup*
* **isinstance(obj,Class) function:** a Boolean function. Returns true if *obj* is an instance of class *Class* or a subclass of *Class*

1. **Base overloading methods**

* a class can override certain base methods, such as:

\_\_init\_\_(), \_\_del\_\_(), \_\_repr\_\_(), \_\_str\_\_(), \_\_cmp\_\_()

1. **Overloading operators**

* operators, such as ‘+’, can have additional functionality in a class by using \_\_add\_\_() method to define that functionality

1. **Data hiding**

* when you name attributes with a double underscore prefix, Python internally changes name to include the class name. These attributes can be accessed as  *object.\_className\_\_attrName*

# Step 2 - Read Web Articles

<https://en.wikibooks.org/wiki/A_Beginner%27s_Python_Tutorial/Classes> (external site). Read the web article.

* **object-oriented-programming** ‘puts functions and variables together in a way that they can see each other and work together, be replicated, and altered as needed, and not when unneeded. And we use a thing called a **class** to do this’
* a class is a blueprint and the objects created from the blueprint are **instances**

# Step 3 - Read a book chapter

I read chapter eight in the course textbook.

# Step 4 - Apply your knowledge

* Created a new sub-folder called Assignment08 inside the \_PythonClass folder
* Created a new project in PyCharm that uses the \_PythonClass\Assignment08 folder as its location
* Created a python script file within the project
* I added code to the pseudocode and in each step iteratively ran small sections of the code to ensure that it was working as expected and if not to tweak it or check my logic
* I initially created the Product class and defined getters and setters for the product name and price. I defined a \_\_str\_\_ that contained the product name and price as a string; I would add this string to the product object list
* I decided to use the menu that we had used for assignment 6; I renamed the options and variables accordingly. The main script starts by loading product names and prices from a file. I used ‘os’ to check whether the file existed
* I rewrote the IO methods to enter the ‘product name’ and ‘product price’ via two separate methods.

I first asked for the product name -> created a product object -> used the setter property to ensure the name was appropriate (i.e., composed of letters)

* I then asked the user to input the product price and once again used the setter property to ensure that it was all numeric. If the entries for name and price were both appropriate the product name and price were added to a list. Otherwise, if either one does not past the setter tests the value remains empty – I check whether it is empty or not and if affirmative return a generic message 'Data rejected. Products names should only contain letters and product prices should only contain numbers.'
* As usual, I ran into multiple errors which I tweaked along the way. I was not sure how we were supposed to save an object class in a text file and assumed that we were only supposed to save the product name and price in a file.
* Boxes 4 and 5 demonstrate the script and a sample run in PyCharm. Screen captures of the text file (Figure 1) and the script running in Command console (Figure 2) follow.
* I was not able to figure out how to name the created individual objects from the product name and price entries and enter the individual objects into a list. Therefore, I added the product name and price to a list as a \_\_str\_\_ () method of an object instance.

|  |
| --- |
| **Box 4 – Script for Assignment 8** |
| *# ------------------------------------------------------------------------ # # Title: Assignment 08: Working with classes # Description: Script will ask for user input of product name # and price, create a product object and use the setter property to determine # whether the product name and price are appropriately entered as letters and numbers # respectively. Reading and writing and menu choices are similar to Assignment06 # ChangeLog (Who,When,What): # RRoot,1.1.2030,Created started script # RRoot,1.1.2030,Added pseudo-code to start assignment 8 # Hedy Khalatbari,08.21.2021,Modified code to complete assignment 8 # --------------------------------------------------------------------------------- #* **import** os *# imports os module  # Data, start -------------------------------------------------------------------- #* strFileName = **'products.txt'** *# name of the data file* file = **None** *# object that represents a file* lstOfProductObjects = [] *# list of product objects* strChoice = **""** *# Captures the user option selection* strProduct = **""** *# Captures the product name* objProduct = **""** *# Captures a newly created object from the Product class* strPrice = **""** *# Captures the product price* status = **""** *# Captures a message to return & print for user feedback* **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:  \_\_str\_\_: (string) product\_info (name and price)  changelog: (When,Who,What)  RRoot,1.1.2030,Created Class  Hedy Khalatbari,08.21.2021,Modified code to complete assignment 8  """   # define constructor, getters and setters  # define \_\_str\_\_ that will return product\_info (product name & price)* status = **''** *# used to capture and return exception messages* **def** \_\_init\_\_(self, product\_name=**''**, product\_price=**''**):  self.\_\_product\_name = product\_name  self.\_\_product\_price = product\_price   **def** \_\_str\_\_(self):  product\_price = str(self.\_\_product\_price)  product\_info = self.\_\_product\_name + **', '** + product\_price  **return** product\_info *# a string with product name and price; saved to the list   # getter and setter for product\_name* @property  **def** product\_name(self):  **return** str(self.\_\_product\_name)   @product\_name.setter  **def** product\_name(self, value):  **if** str(value).isnumeric() == **False**: *# checks that it is not numeric* value = value.strip().lower() *# lower case & strip* self.\_\_product\_name = value *# sets it if condition fulfilled  # otherwise, the product\_name remains as '' (i.e., an empty string)   # getter and setter for product\_price* @property  **def** product\_price(self):  **return** str(self.\_\_product\_price)   @product\_price.setter  **def** product\_price(self, value):  value = value.strip() *# strip* **if** str(value).isnumeric() == **True**: *# checks if numeric* self.\_\_product\_price = float(value) *# sets it as a float  # otherwise, the product\_price remains as '' (i.e., an empty string)   # Data, end -------------------------------------------------------------------- #   # Processing, start ------------------------------------------------------------- #* **class** FileProcessor:  *"""Processes data to and from a file and a list of product objects:   methods:  read\_data\_from\_file(file\_name): -> (a list of product objects)   save\_data\_to\_file(file\_name, list\_of\_product\_objects):   changelog: (When,Who,What)  RRoot,1.1.2030,Created Class  Hedy Khalatbari,08.21.2021,Modified code to complete assignment 8  """   # Code to process data to a file* @staticmethod  **def** read\_data\_from\_file(file\_name):  *""" Reads data from a file into a list of dictionary rows* **:param** *file\_name: string with name of file:* **:return***: list of product objects, status  """* status = **''** *# added this here as was getting an error* list\_of\_product\_objects = [] *# local variable, list of products names & prices* **if** os.path.exists(file\_name): *# used to check if the text file exists and run script if True* file = open(file\_name, **"r"**)  **for** line **in** file:  **if "," in** line: *# added this condition as was getting an error when text file empty* list\_of\_product\_objects.append(line)  status = **'Data read from file.'  else**:  status = **'There is no data saved in the file.'** file.close()  **else**:  status = **'There is no data saved in the file.'  return** list\_of\_product\_objects, status   @staticmethod  **def** add\_product\_to\_list(product, price, lstOfProductObjects):  *""" Adds user input data (product, price) to a list of dictionary rows* **:param** *product: (string) product we want to add:* **:param** *price: (string) price of product to add:* **:param** *lstOfProductObjects: (list) of dictionary rows* **:return***:(list) of dictionary rows  """* new\_product = Product(product, price) *# from a class instance object* lstOfProductObjects.append(str(new\_product)) *# save the instance object to list* status = **'Product name and price were added to list.'  return** lstOfProductObjects, status   @staticmethod  **def** remove\_data\_from\_list(product, lstOfProductObjects):  *""" Removes product from list of dictionaries* **:param** *product: (string) product we want to remove:* **:param** *lstOfProductObjects: (list) of products* **:return***: (list) of products, success/failure of task  """* product = product.strip() *# product to be removed* product = product.lower()  **for** row **in** lstOfProductObjects:  product\_local, price = row.split(**','**)  **if** product == product\_local.strip(): *# remove the user selected product* lstOfProductObjects.remove(row)  status = **'Product was removed.'  else**:  status = **'Product was not in list.'  return** lstOfProductObjects, status   @staticmethod  **def** write\_data\_to\_file(file\_name, lstOfProductObjects):  *""" Write data from a list of dictionary rows into a file* **:param** *file\_name: (string) with name of file:* **:param** *lstOfProductObjects: (list) of dictionary rows:* **:return***: status of task  """* file = open(file\_name, **"w"**)  **for** row **in** lstOfProductObjects:  file.write(row + **'\n'**)  status = **'Data was written to file.'** file.close()  **return** status    *# Presentation (Input/Output) -------------------------------------------- #* **class** IO:  *""" Performs Input and Output products """* @staticmethod  **def** print\_menu\_products():  *""" Display a menu of choices to the user* **:return***: nothing  """* print(**'''  Menu of Options:  1) Add a new product name & price  2) Remove an existing product  3) Save Data to File   4) Reload Data from File  5) Show product name & price list  6) Exit Program  '''**)  print() *# Add an extra line for looks* @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 6] - "**)).strip()  print() *# Add an extra line for looks* **return** choice   @staticmethod  **def** print\_current\_Products\_in\_list(lstOfProductObjects):  *""" Shows the current products* **:param** *lstOfProductObjects: list of product names and prices* **:return***: nothing  """* print(**"\*\* Current product names and prices are: \*\*"**)  **if** lstOfProductObjects != []: *# check whether list is empty* **for** row **in** lstOfProductObjects:  print(row.strip())  **else**:  print(**"There are no entries in the list."**)  print(**"\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*"**)  print() *# Add an extra line for looks* @staticmethod  **def** input\_yes\_no\_choice(message):  *""" Gets a yes or no choice from the user* **:return***: string  """* choice = str(input(message))  choice = choice.strip()  choice = choice.lower()  **return** choice   @staticmethod  **def** input\_press\_to\_continue(optional\_message=**''**):  *""" Pause program and show a message before continuing* **:param** *optional\_message: An optional message you want to display* **:return***: nothing  """* print(optional\_message)  input(**'Press the [Enter] key to continue.'**)   @staticmethod  **def** input\_new\_product():  *""" Asks user to input new product* **:return***: (strings) product name  """* productName = str(input(**"Enter product name: "**))  product\_1 = Product() *# instantiate an object from the class Product* product\_1.product\_name = productName *# set the product name* **return** product\_1, product\_1.product\_name   @staticmethod  **def** input\_new\_price(product\_object):  *""" Asks user to input new price* **:return***: (strings) product price  """* productPrice = str(input(**"Enter product price: "**))  product\_object.product\_price = productPrice  **return** product\_object.product\_price   @staticmethod  **def** input\_product\_to\_remove():  *""" Asks user which product they would like to remove* **:return***: (string) product  """* product = str(input(**"Enter product to remove: "**))  **return** product   *# Main Body of Script ------------------------------------------------------ #   # Step 1 - When the program starts, Load data from products.txt and print list of product names & prices* lstOfProductObjects, status = FileProcessor.read\_data\_from\_file(strFileName) *# read file data into list table* print(status) *# feedback to user regarding file contents* IO.print\_current\_Products\_in\_list(lstOfProductObjects) *# shows current data in the list  # Step 2 - Display a menu of choices to the user* **while** (**True**):  *# Step 3 Show menu and ask user to choose a menu option* IO.print\_menu\_products() *# Shows menu* strChoice = IO.input\_menu\_choice() *# Get menu option   # Step 4 - Process user's menu choice* **if** strChoice == **'1'**: *# Add a new product* objProduct,strProduct = IO.input\_new\_product()  **if** strProduct != **''**: *# checks that it is not numeric* strPrice = IO.input\_new\_price(objProduct)  **if** strPrice != **''**: *# checks to ensure value was assigned* lstOfProductObjects, status = FileProcessor.add\_product\_to\_list(strProduct, strPrice,  lstOfProductObjects)  print(status) *# message to user* **else**:  print(**'Data rejected.Products names should only contain letters and '** *# message to user* **'product prices should only contain numbers.'**)  **else**:  print(**'Data rejected. Products names should only contain letters and '** *# message to user* **'product prices should only contain numbers.'**)  IO.input\_press\_to\_continue()  **continue** *# to show the menu* **elif** strChoice == **'2'**: *# Remove an existing product* strProduct = IO.input\_product\_to\_remove()  lstOfProductObjects, status = FileProcessor.remove\_data\_from\_list(strProduct, lstOfProductObjects)  print(status)  IO.input\_press\_to\_continue()  **continue** *# to show the menu* **elif** strChoice == **'3'**: *# Save Data to File* strChoice = IO.input\_yes\_no\_choice(**"Save this data to file? (y/n) - "**)  **if** strChoice.lower() == **"y"**:  status = FileProcessor.write\_data\_to\_file(strFileName, lstOfProductObjects)  print(status)  IO.input\_press\_to\_continue()  **else**:  IO.input\_press\_to\_continue(**"Save Cancelled!"**)  **continue** *# to show the menu* **elif** strChoice == **'4'**: *# Reload Data from File* print(**"Warning: Unsaved Data Will Be Lost!"**)  strChoice = IO.input\_yes\_no\_choice(**"Are you sure you want to reload data from file? (y/n) - "**)  **if** strChoice.lower() == **'y'**:  lstOfProductObjects, status = FileProcessor.read\_data\_from\_file(strFileName)  print(status)  IO.print\_current\_Products\_in\_list(lstOfProductObjects)  IO.input\_press\_to\_continue()  **else**:  IO.input\_press\_to\_continue(**"File Reload Cancelled!"**)  **continue** *# to show the menu* **elif** strChoice == **'5'**: *# Show current data in the list of dictionary rows* IO.print\_current\_Products\_in\_list(lstOfProductObjects) *# Show current data in the list of dictionary rows* **elif** strChoice == **'6'**: *# Exit Program* print(**"Goodbye!"**)  **break** *# and Exit* **else**:  print(**"Please choose from menu options"**) |

|  |
| --- |
| **Box 5 – Script running in PyCharm** |
| C:\\_PythonClass\Assignment08\venv\Scripts\python.exe C:/\_PythonClass/Assignment08/Assignment08\_v11fileTweak.py  Data read from file.  \*\* Current product names and prices are: \*\*  computer, 345.0  laptop, 453546.0  \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*  Menu of Options:  1) Add a new product name & price  2) Remove an existing product  3) Save Data to File  4) Reload Data from File  5) Show product name & price list  6) Exit Program    Which option would you like to perform? [1 to 6] - 1  Enter product name: chaIR  Enter product price: 2345  Product name and price were added to list.  Press the [Enter] key to continue.  Menu of Options:  1) Add a new product name & price  2) Remove an existing product  3) Save Data to File  4) Reload Data from File  5) Show product name & price list  6) Exit Program    Which option would you like to perform? [1 to 6] - 5  \*\* Current product names and prices are: \*\*  computer, 345.0  laptop, 453546.0  chair, 2345.0  \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*  Menu of Options:  1) Add a new product name & price  2) Remove an existing product  3) Save Data to File  4) Reload Data from File  5) Show product name & price list  6) Exit Program    Which option would you like to perform? [1 to 6] - 3  Save this data to file? (y/n) - y  Data was written to file.  Press the [Enter] key to continue.  Menu of Options:  1) Add a new product name & price  2) Remove an existing product  3) Save Data to File  4) Reload Data from File  5) Show product name & price list  6) Exit Program    Which option would you like to perform? [1 to 6] - 6  Goodbye!  Process finished with exit code 0 |

Graphical user interface, text, application

Description automatically generated

**Figure 1.** Screen capture of text saved in product.txt

Text

Description automatically generated

**Figure 2.** Screen capture of script running in Command console

# Step 6 - Document your knowledge

I documented by learning process throughout the assignment steps in this word document.

# Step 7 - Post Files to GitHub using GitHub Desktop

* Watched ‘GitHub Desktop (for Everyone)’ [**https://youtu.be/77W2JSL7-r8**](https://youtu.be/77W2JSL7-r8) (external site)
* Installed GitHub Desktop and logged into my GitHub account.
* Created a new repository called "IntroToProg-Python-Mod08" on my local computer.
* Copied the script and word document to the repository
* Published the repository
* Confirmed that the repository had been published to the GitHub website; URL is

……

# Step 7 -(optional); post a link to GitHub in MS word document; and submit your work

I did not create a Github webpage.

I posted the link to Github in the header of the MS word assignment document.

I submitted my Python script and Word document as a Canvas assignment for grading and posted a link to your GitHub site on the assignment textbox.

# Summary

In this module we learned more about classes. After watching the video, reading the book chapter and recommended web article as well as additional reading from the Python documentation and TutorialsPoint (references cited in Step1B section above) I still fill that I do not have a solid understanding of this subject. I will try to find more resources to increase my understanding and grasp of classes in Python – and object-oriented programming in general.