 Zefeng Shen

August 30, 2020

IT FDN 110A

Assignment08

<https://github.com/zzsocool/IntroToProg-Python-Mod08->

**Classes in Python**

**Introduction:**

In this week, I learned how to create a class with multiple class instances and how to use code directly and indirectly from class. In inside classes, there are constructor, destructor, attributes, properties, and methods. I also learned how to use Github desktop software.

**Classes:**

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.

Compared with other programming languages, Python’s class mechanism adds classes with a minimum of new syntax and semantics. It is a mixture of the class mechanisms found in C++ and Modula-3. Python classes provide all the standard features of Object Oriented Programming: 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. As is true for modules, classes partake of the dynamic nature of Python: they are created at runtime, and can be modified further after creation.

(Doc.python, <https://docs.python.org/3/tutorial/classes.html>) (External website)

**Instance:**

An instance attribute is a Python variable belonging to only one object. It is only accessible in the scope of the object and it is defined inside the constructor function of a class. For example, \_\_init\_\_(self,..).

The difference is that class attributes is shared by all instances. When you change the value of a class attribute, it will affect all instances that share the same exact value. The attribute of an instance on the other hand is unique to that instance.

(Medium, <https://medium.com/@mohamethseck/class-and-instance-attributes-bb2ab2a36227>) (External website)

**Constructor:**

Constructors are for instantiating an object.The task of constructors is to initialize the data members of the class when an object of class is created. The \_\_init\_\_() method is called the constructor in Python and is always called when an object is created.

(Edureka, <https://www.edureka.co/community/42444/what-is-a-constructor-in-python#:~:text=Constructors%20are%20for%20instantiating%20an,when%20an%20object%20is%20created>.) (External website)

**Attributes:**

* An instance attribute is a Python variable belonging to one, and only one, object. This variable is only accessible in the scope of this object and it is defined inside the constructor function, \_\_init\_\_(self,..) of the class.
* A class attribute is a Python variable that belongs to a class rather than a particular object. It is shared between all the objects of this class and it is defined outside the constructor function, \_\_init\_\_(self,...), of the class.

（Dzone，<https://dzone.com/articles/python-class-attributes-vs-instance-attributes>） (External website)

**Properties:**

In Python, getters and setters are not the same as those in other object-oriented programming languages. Basically, the main purpose of using getters and setters in object-oriented programs is to ensure data encapsulation. [Private variables in python](https://www.geeksforgeeks.org/private-variables-python/) are not actually hidden fields like in other object oriented languages. Getters and Setters in python are often used when:

* We use getters & setters to add validation logic around getting and setting a value.
* To avoid direct access of a class field i.e. private variables cannot be accessed directly or modified by external user.   
    
  (Geeksfogeeks, <https://www.geeksforgeeks.org/getter-and-setter-in-python/>) (External website)

**Methods：**

A method in python is somewhat similar to a function, except it is associated with object/classes. Methods in python are very similar to functions except for two major differences.

* The method is implicitly used for an object for which it is called.
* The method is accessible to data that is contained within the class.

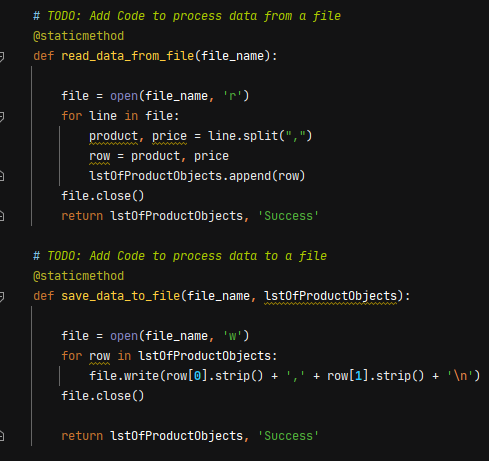
**Script:**

First, I added in class product. I used self() to make the code can be used for multiple instance. And I used getter and setter to add validation logic around user’s input value. I also add method and used \_\_str\_\_ to concatenate two attributes as shown in figure 1.



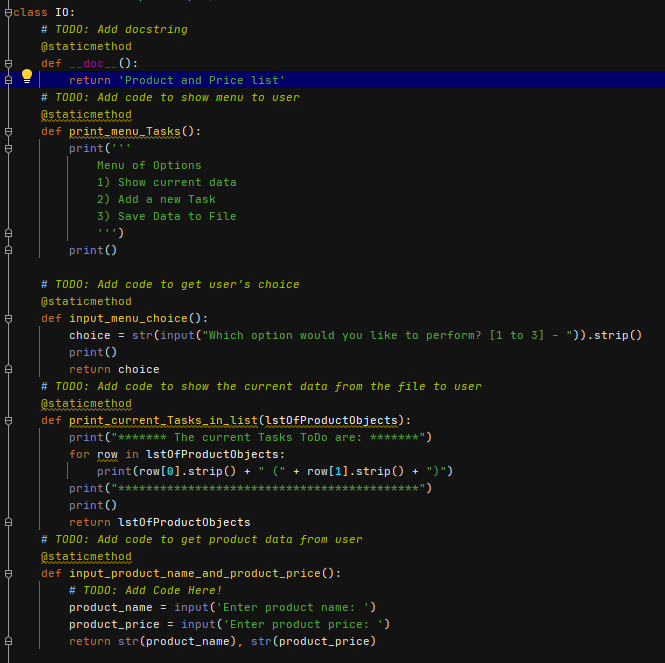
**Figure 1**: Class Product.

Then I worked on “class Fileprocessor”. I used @staticmethod so no instance would be made for this class. I created two methods for read data from file and save data to the file. This code was very similar to the assignment I did before. So, I just edit with it a little bit as shown in figure 2.



**Figure 2**: Class fileprocessor.

Next, presentation part for class IO, same as the class fileprocessor, I used @staticmethod so no instance would be made for this class. Basically, just create methods to capture user’s input. When capture input for price, it should be number so I use float(). However, when using the price input, I must convert back to string again. That was the issue I encountered when I was running the script as shown in figure 3.



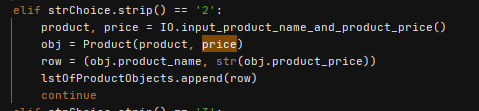
**Figure 3**: Class IO.

For the main body of the script, it was very similar with assignment 6. The only issue I had was, I cannot collect the data information from the class product. When I tried to append the user’s input to the existing list, it displayed the information I did not expected as shown in figure 4.



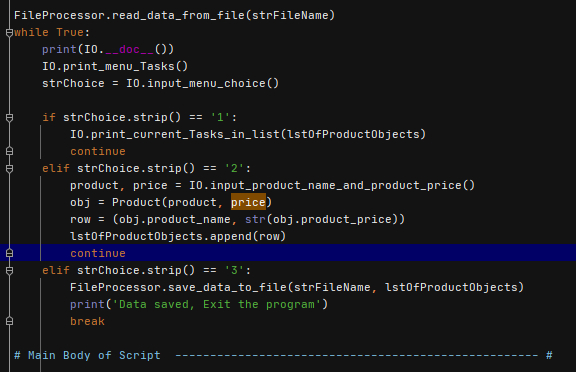
Figure 4: Unexpected data.

First, I thought I could fomat\_\_str\_\_ as the same way show in the list and directly append into the existing list, but it failed. After I tried couple attempts, I created an instance named obj of class product. Then I called out the methods from the instance and passed the attributes into a list named row; I appended the data to the existing list as shown in figure 5.



**Figure 5:** Using data from class instance.

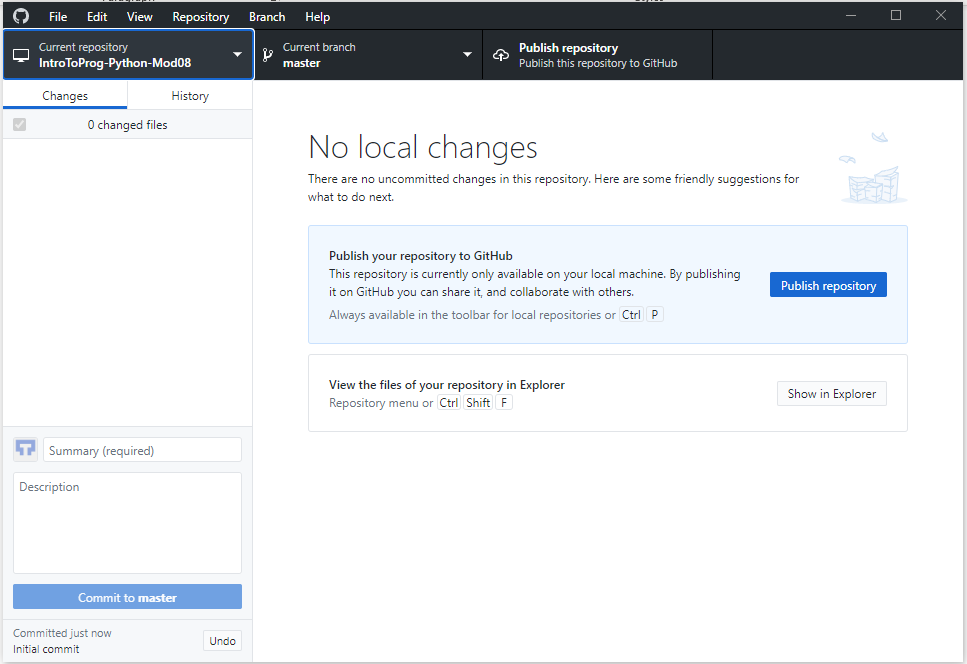
Here is my main body of script as shown in figure 6.



**Figure 6:** main body of script.

**Github:**

I followed the document and created a new repository on my Github desktop as show in figure 7.



**Figure 7:** Github desktop.

**Summary:**

Class and instance are very handy to organize the code for repeatable usage. However, it takes time to remember what is inside the class and what attributes the class will return. I have to go back and forth to check several times. It definitely makes the main body of script short and neat. Github desktop is a very useful tool when using offline. it can synchronize the data from local computer to Github website.