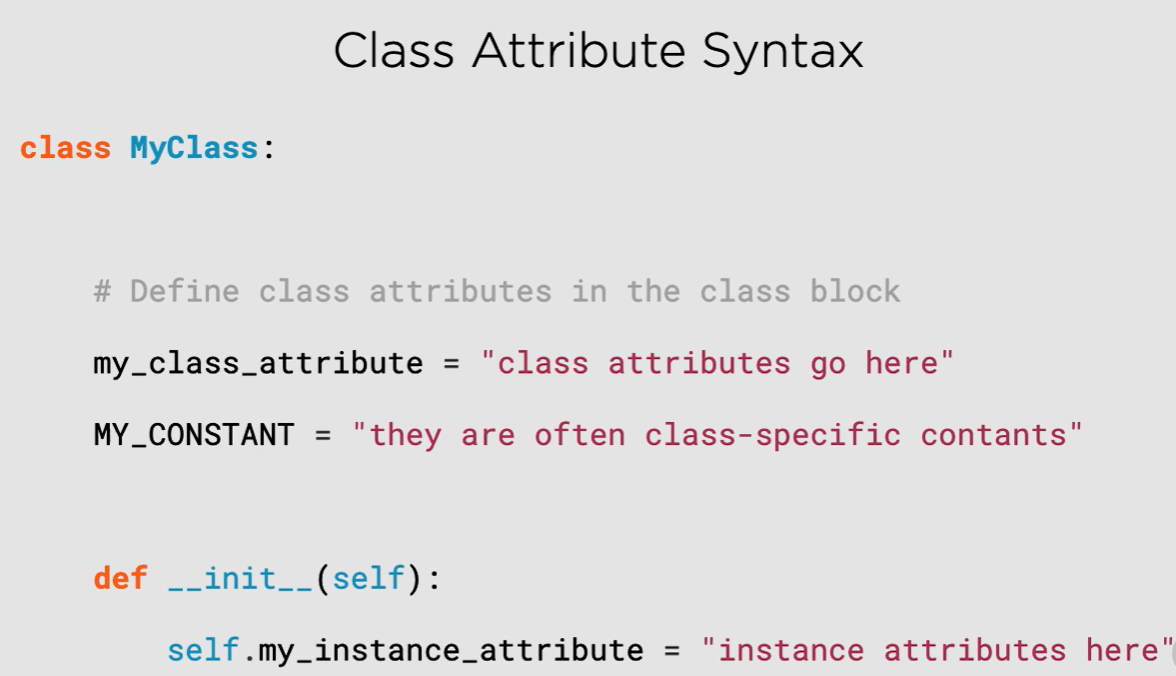
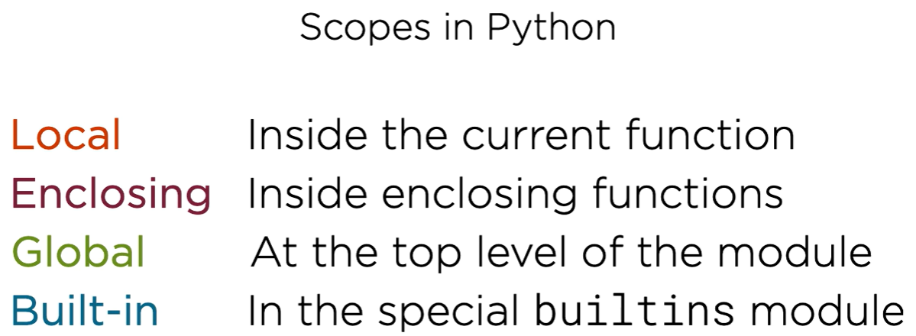
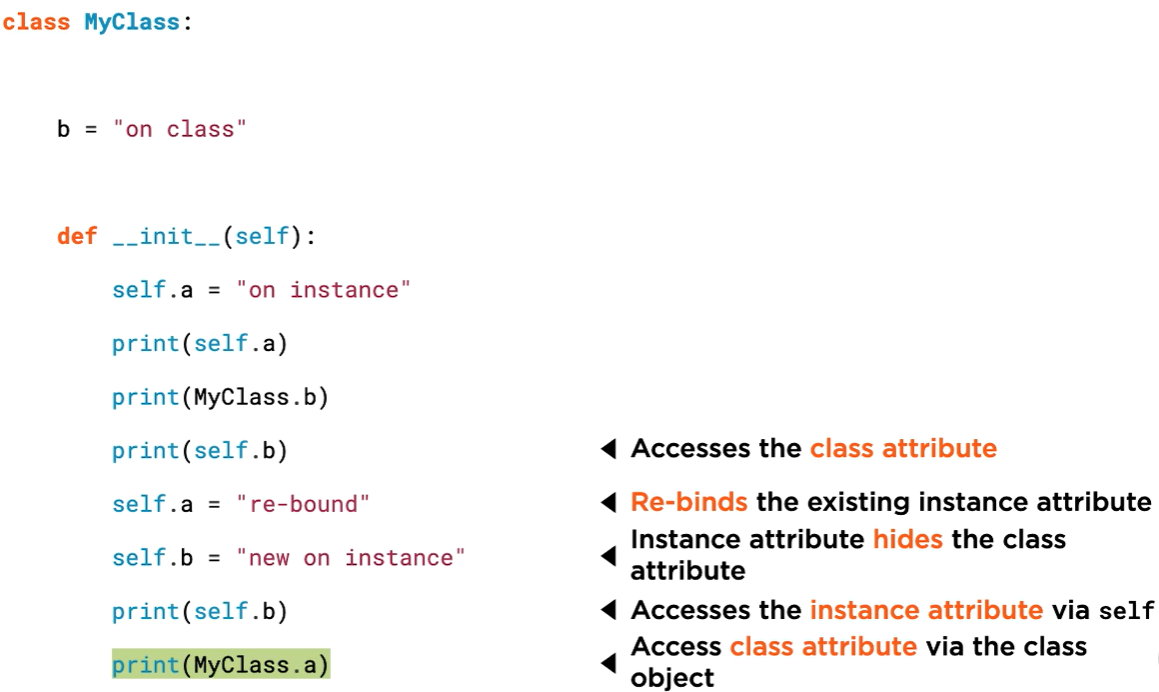
**Class Attributes, Methods and Properties**

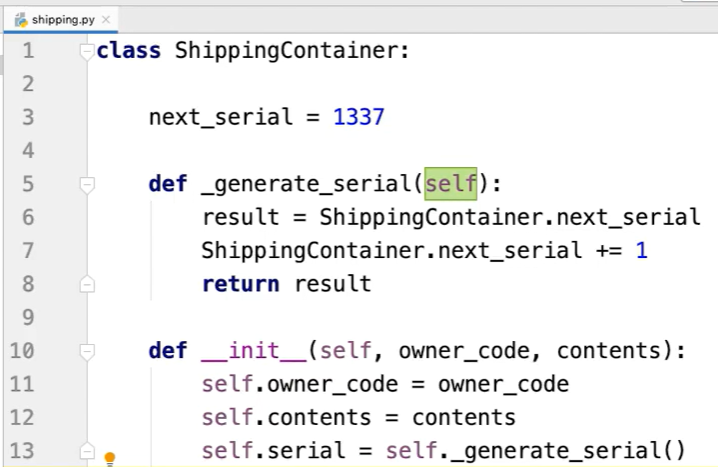
**Class Attributes**

****

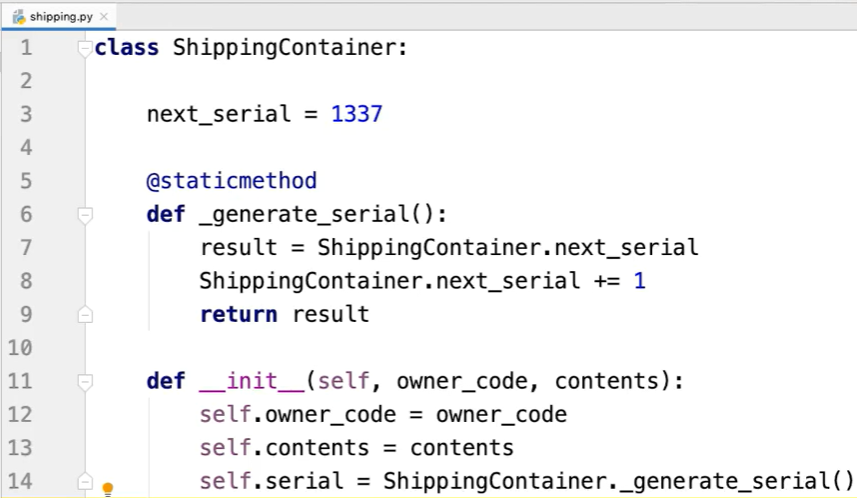
****

****

**Static Methods**

****

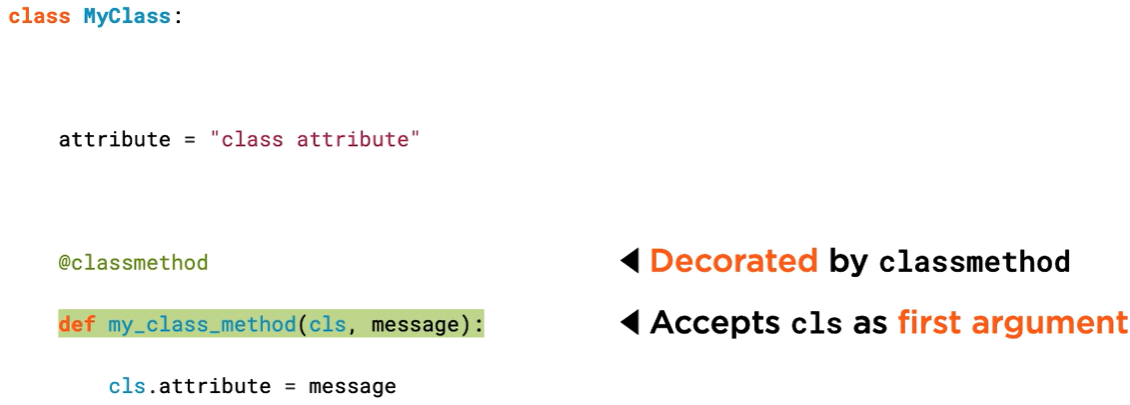
Instead of above code we should write like this

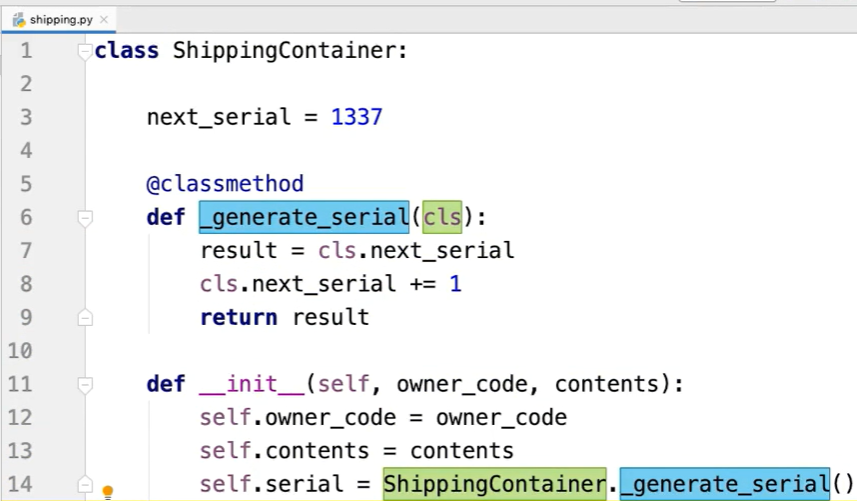


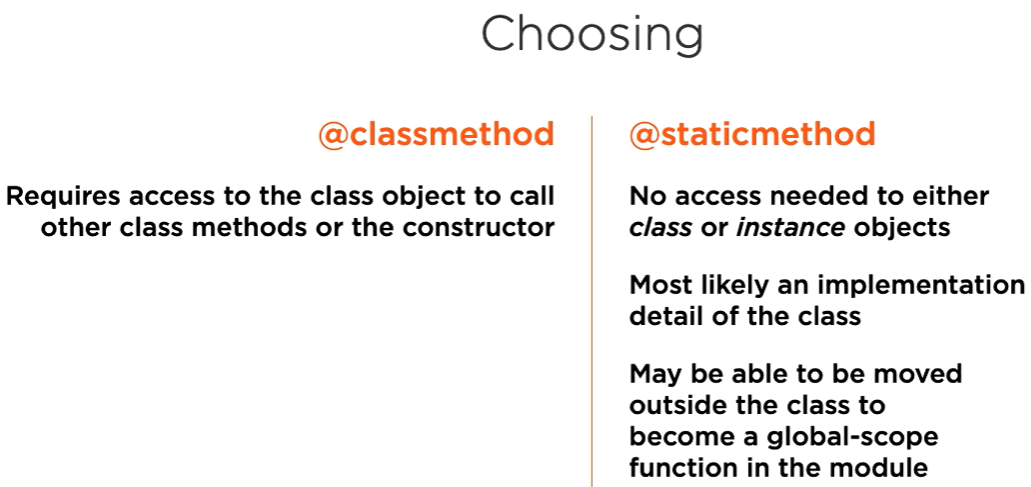
**Output:**

****

**Class Methods**

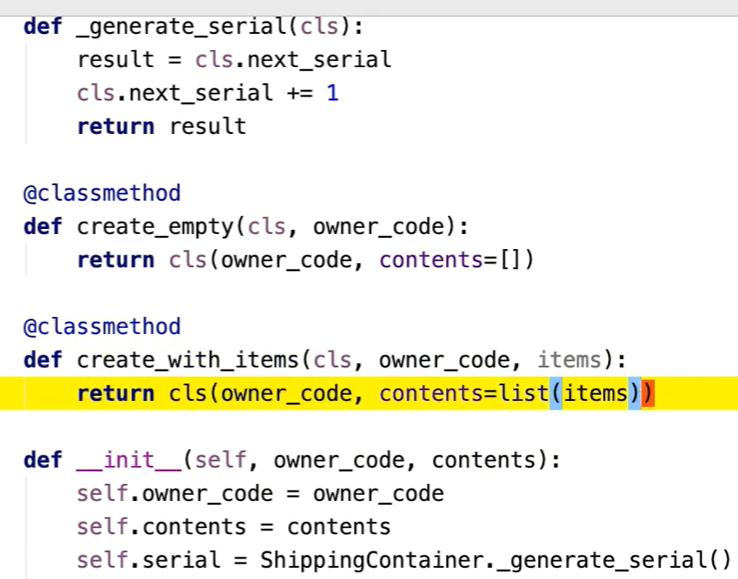
****

****

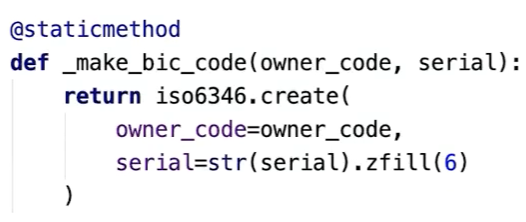
****

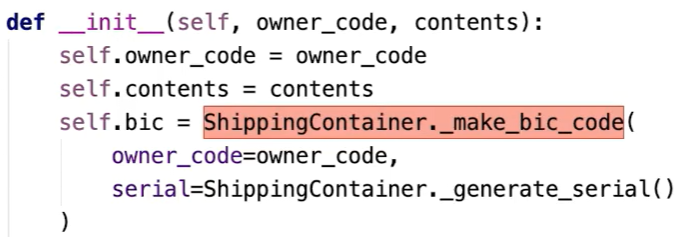
* If you need to refer to the class object within the method, for example to access a class attribute, prefer to use @classmethod.
* If you don't need access to the class object, use @staticmethod.
* use @staticmethod. In practice, most static methods will be internal implementation details of the class, marked as such with a leading underscore since having no access to either the class object or the instance object they rarely form a useful part of the class interface. In principle, it would also be possible to implement any static method completely outside the class at the global module scope without any loss of functionality. So you may want to consider carefully whether a particular function should be a module scope function or a static method. The @staticmethod decorator merely facilitates a particular logical organization of the code, allowing us to place what could otherwise be free functions within classes. Sometimes you would like a class to support named constructors, also
* known as factory functions, which construct objects with certain configurations.

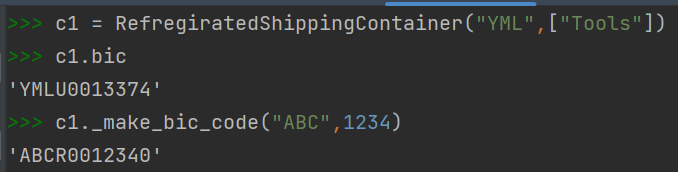
**Named Constructors**

****

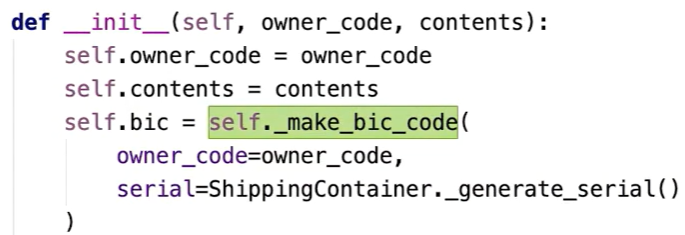
**Static Methods with Inheritance**

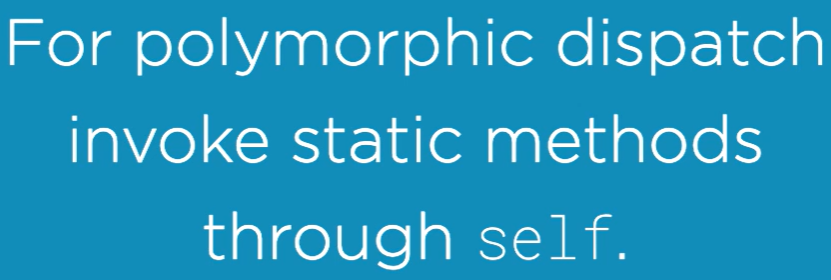
****

****

****

So we can get polymorphic dispatch of static methods, but only when we call the method through an instance, not when we call the method through a class.



****