**Q: *What is a metaclass in Python?***

# A: ……

# What are metaclasses

Metaclasses are the 'stuff' that creates classes.

You define classes in order to create objects, right?

But we learned that Python classes are objects.

Well, metaclasses are what create these objects. They are the classes' classes, you can picture them this way:

MyClass = MetaClass()

MyObject = MyClass()

You've seen that type lets you do something like this:

MyClass = type('MyClass', (), {})

It's because the function type is in fact a metaclass. type is the metaclass Python uses to create all classes behind the scenes.

Now you wonder why the heck is it written in lowercase, and not Type?

Well, I guess it's a matter of consistency with str, the class that creates strings objects, and intthe class that creates integer objects. type is just the class that creates class objects.

You see that by checking the \_\_class\_\_ attribute.

Everything, and I mean everything, is an object in Python. That includes ints, strings, functions and classes. All of them are objects. And all of them have been created from a class:

>>> age = 35

>>> age.\_\_class\_\_

<type 'int'>

>>> name = 'bob'

>>> name.\_\_class\_\_

<type 'str'>

>>> def foo(): pass

>>> foo.\_\_class\_\_

<type 'function'>

>>> class Bar(object): pass

>>> b = Bar()

>>> b.\_\_class\_\_

<class '\_\_main\_\_.Bar'>

Now, what is the \_\_class\_\_ of any \_\_class\_\_ ?

>>> age.\_\_class\_\_.\_\_class\_\_

<type 'type'>

>>> name.\_\_class\_\_.\_\_class\_\_

<type 'type'>

>>> foo.\_\_class\_\_.\_\_class\_\_

<type 'type'>

>>> b.\_\_class\_\_.\_\_class\_\_

<type 'type'>

So, a metaclass is just the stuff that creates class objects.

You can call it a 'class factory' if you wish.

type is the built-in metaclass Python uses, but of course, you can create your own metaclass.

**The**[**\_\_metaclass\_\_**](http://docs.python.org/2/reference/datamodel.html?highlight=__metaclass__#__metaclass__)**attribute**

You can add a \_\_metaclass\_\_ attribute when you write a class:

class Foo(object):

\_\_metaclass\_\_ = something...

[...]

If you do so, Python will use the metaclass to create the class Foo.

Careful, it's tricky.

You write class Foo(object) first, but the class object Foo is not created in memory yet.

Python will look for \_\_metaclass\_\_ in the class definition. If it finds it, it will use it to create the object class Foo. If it doesn't, it will use type to create the class.

Read that several times.

When you do:

class Foo(Bar):

pass

Python does the following:

Is there a \_\_metaclass\_\_ attribute in Foo?

If yes, create in memory a class object (I said a class object, stay with me here), with the name Fooby using what is in \_\_metaclass\_\_.

If Python can't find \_\_metaclass\_\_, it will look for a \_\_metaclass\_\_ at the MODULE level, and try to do the same (but only for classes that don't inherit anything, basically old-style classes).

Then if it can't find any \_\_metaclass\_\_ at all, it will use the Bar's (the first parent) own metaclass (which might be the default type) to create the class object.

Be careful here that the \_\_metaclass\_\_ attribute will not be inherited, the metaclass of the parent (Bar.\_\_class\_\_) will be. If Bar used a \_\_metaclass\_\_ attribute that created Bar with type() (and not type.\_\_new\_\_()), the subclasses will not inherit that behavior.

Now the big question is, what can you put in \_\_metaclass\_\_ ?

The answer is: something that can create a class.

And what can create a class? type, or anything that subclasses or uses it.

译文

问：***Python中的元类是什么？***

答：**什么是元类**

元类就是用来创建类的“东西”。你创建类就是为了创建类的实例对象，不是吗？但是我们已经学习到了Python中的类也是对象。好吧，元类就是用来创建这些类（对象）的，元类就是类的类，你可以这样理解 为：

MyClass = MetaClass()

MyObject = MyClass()

你已经看到了type可以让你像这样做：

MyClass = type('MyClass', (), {})

这是因为函数type实际上是一个元类。type就是Python在背后用来创建所有类的元类。现在你想知道那为什么type会全部采用小写形式而不是Type呢？好吧，我猜这是为了和str保持一致性，str是用来创建字符串对象的类，而int是用来创建整数对象的类。type就是创建类对象的类。你可以通过检查\_\_class\_\_属性来看到这一点。Python中所有的东西，注意，我是指所有的东西——都是对象。这包括整数、字符串、函数以及类。它们全部都是对象，而且它们都是从一个类创建而来。

>>> age = 35

>>> age.\_\_class\_\_

<type 'int'>

>>> name = 'bob'

>>> name.\_\_class\_\_

<type 'str'>

>>> def foo(): pass

>>> foo.\_\_class\_\_

<type 'function'>

>>> class Bar(object): pass

>>> b = Bar()

>>> b.\_\_class\_\_

<class '\_\_main\_\_.Bar'>

现在，对于任何一个\_\_class\_\_的\_\_class\_\_属性又是什么呢？

>>> age.\_\_class\_\_.\_\_class\_\_

<type 'type'>

>>> name.\_\_class\_\_.\_\_class\_\_

<type 'type'>

>>> foo.\_\_class\_\_.\_\_class\_\_

<type 'type'>

>>> b.\_\_class\_\_.\_\_class\_\_

<type 'type'>

因此，元类就是创建类这种对象的东西。如果你喜欢的话，可以把元类称为“类工厂”（不要和工厂类搞混了:D） type就是Python的内建元类，当然了，你也可以创建自己的元类。

[**\_\_metaclass\_\_**](http://docs.python.org/2/reference/datamodel.html?highlight=__metaclass__#__metaclass__)**属性**

你可以在写一个类的时候为其添加\_\_metaclass\_\_属性。

class Foo(object):

\_\_metaclass\_\_ = something...

[...]

如果你这么做了，Python就会用元类来创建类Foo。小心点，这里面有些技巧。你首先写下class Foo(object)，但是类对象Foo还没有在内存中创建。Python会在类的定义中寻找\_\_metaclass\_\_属性，如果找到了，Python就会用它来创建类Foo，如果没有找到，就会用内建的type来创建这个类。把下面这段话反复读几次。当你写如下代码时 :.

class Foo(Bar):

pass

Python做了如下的操作：

Foo中有\_\_metaclass\_\_这个属性吗？如果是，Python会在内存中通过\_\_metaclass\_\_创建一个名字为Foo的类对象（我说的是类对象，请紧跟我的思路）。如果Python没有找到\_\_metaclass\_\_，它会继续在Bar（父类）中寻找\_\_metaclass\_\_属性，并尝试做和前面同样的操作。如果Python在任何父类中都找不到\_\_metaclass\_\_，它就会在模块层次中去寻找\_\_metaclass\_\_，并尝试做同样的操作。如果还是找不到\_\_metaclass\_\_,Python就会用内置的type来创建这个类对象。

现在的问题就是，你可以在\_\_metaclass\_\_中放置些什么代码呢？答案就是：可以创建一个类的东西。那么什么可以用来创建一个类呢？type，或者任何使用到type或者子类化type的东东都可以。

**Discussion**：The answerer [e-satis](http://stackoverflow.com/users/9951/e-satis) from the“stackoverflow”worked out a nearly perfect illustration about “what is metaclass in python”. Actually this is only a part of the article. Before getting to the point, He paid great of writings on explaining that the class is a kind of objects and that we can create a class dynamically with a simple function “type”. It also includes details of how to implement such progress. After that, the author points out that the type function is a metaclass, moreover, it is the metaclass for itself. Python offers users different sorts of metaclass to create those class with different variables. They are the “class factories” in metaphor. The author clearly narrates each step that Python does when it gets the order to build up a class; first finding in memory, then the parent class, next module level, finally using the built-in function to finish creating.

*“Metaclasses are deeper magic that 99% of users should never worry about. If you wonder whether you need them, you don't (the people who actually need them know with certainty that they need them, and don't need an explanation about why).”——* Python Guru Tim Peters

In fact, we seldom use metaclass during programming. The main purpose of metaclass is to establish API (Application Programming Interface). And I don’t know much about that. Anyway, metaclasses give us more choices to create classes in program. And with its’ callable character, it surely reflects the oneness and user-friendliness of Python.