What Is

and

What Can Be

An Exploration from `type` to metaclasses

I'm Dustin

http://github.com/di


promptworks

Irked.

jeffdeville 3:40 PM

In python, how can I create an empty object, and just start assigning properties to it. I don't want to have to define a class that is empty just to do this.

j = object()

j.hi = "there"

doesn't work… I'm irked

Irked.

```
>>> j = object()
```

Irked.

```
>>> j = object()
>>> j.hi = 'there'
```

Irked.

```
>>> j = object()
>>> j.hi = 'there'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'object' object has no
attribute 'hi'
```

type

type

```
>>> type(['foo', 'bar'])
```

type

```
>>> type(['foo', 'bar'])
<class 'list'>
```

type

```
>>> type(list)
```

type

```
>>> foo = list()
```

type

```
>>> foo = list()
>>> foo
[]
```

type

```
>>> foo = list()
>>> foo
[]
>>> type(foo)
<class 'list'>
```

type

```
>>> type(list)
```

type

```
>>> type(list)
<class 'type'>
```

type

```
>>> type(type)
```

type

```
>>> type(type)
<class 'type'>
```

type

```
>>> type(None)
```

type

```
>>> type(None)
<class 'NoneType'>
```

type

```
>>> def func():
...     pass
...
>>> type(func)
```

type

```
>>> def func():
...     pass
...
>>> type(func)
<class 'function'>
```

type

```
>>> import types
>>> dir(types)
['BuiltinFunctionType',
'BuiltinMethodType', 'CodeType',
'CoroutineType', 'DynamicClassAttribute',
'FrameType', 'FunctionType',
'GeneratorType', 'GetSetDescriptorType',
'LambdaType', 'MappingProxyType',
'MemberDescriptorType', 'MethodType',
'ModuleType', 'SimpleNamespace',
'TracebackType', ... ]
```

type

```
>>> import types
>>> types.FunctionType
<class 'function'>
```

type

```
>>> import types
>>> type(types)
```

type

```
>>> import types
>>> type(types) is types.ModuleType
True
>>> type(types)
<class 'module'>
```

type

```
>>> class FooClass:
...     pass
...
>>> type(FooClass)
```

type

```
>>> class FooClass:
...     pass
...
>>> type(FooClass())
<class '__main__.FooClass'>
```

type

```
>>> class FooClass:
...     pass
...
>>> type(FooClass)
```

type

```
>>> class FooClass: # Python 2.7
...     pass
...
>>> type(FooClass)
<type 'classobj'>
```

type

```
>>> class FooClass: # Python 3.5
...     pass
...
>>> type(FooClass)
<class 'type'>
```

type

```
>>> class FooClass(object): # Python 2.7
...     pass
...
>>> type(FooClass)
<type 'type'>
```

Python is dead.

Long live Python

type

```
>>> type(42)
```

type

```
>>> type(42)
<class 'int'>
```

type

```
>>> type(42) is int
True
```

type

```
>>> type(42) is int
True
>>> type(42)()
0
```

type

```
>>> type(42) is int
True
>>> type(42)()
0
>>> int()
0
```

type

```
>>> j = type()
```

type

```
>>> j = type()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: type() takes 1 or 3 arguments
```

type

```
>>> class
```

type

```
>>> class FooClass
```

type

```
>>> class FooClass(object):
```

type

```
>>> class FooClass:
```

type

```
>>> class FooClass:
...       hi = 'there'
```

type

```
>>> j = type(
```

type

```
>>> j = type(
...         'FooClass',
```

type

```
>>> j = type(
...         'FooClass',
...         (object,),
```

type

```
>>> j = type(
...         'FooClass',
...         (object,),
...         {'hi': 'there'},
...     )
```

type

```
>>> j = type(
...         'FooClass',
...         (object,),
...         {'hi': 'there'},
...     )
>>> type(j)
<class 'type'>
```

type

```
>>> j = type(
...         'FooClass',
...         (object,),
...         {'hi': 'there'},
... )
>>> j.hi
'there'
```

type

```
>>> j = type(
...         '',
...         (object,),
...         {'hi': 'there'},
... )
>>> j.hi
'there'
```

type

```
>>> j = type(
...         '',
...         (),
...         {'hi': 'there'},
... )
>>> j.hi
'there'
```

type

```
>>> j = type(
...         '',
...         (),
...         {},
... )
>>> j.hi = 'there'
>>> j.hi
'there'
```

(side note)

stub

```
$ pip install pretend

>>> from pretend import stub
>>> j = stub(hi='there')
>>> j.hi
'there'
```

Metaclasses

Metaclasses

"The subject of metaclasses in Python has caused hairs to raise and even brains to explode."

Metaclasses

"The subject of metaclasses in Python has caused hairs to raise and even brains to explode."

– Guido van Rossum

Metaclasses

classes : instances : : metaclasses : classes

Metaclasses

```
>>> j = type(
...         'FooClass',
...         (object,),
...         {'hi': 'there'},
...     )
>>> type(j)
<class 'type'>
```

Metaclasses

```
>>> class MyMeta(type):
...     pass
...
```

Metaclasses

```
>>> class MyMeta(type):
...     pass
...
>>> class FooClass(metaclass=MyMeta):
...     pass
...
```

Metaclasses

```
>>> class MyMeta(type):
...     def __new__(meta, name, bases, attrs):
...         return super().__new__(
...             meta, name, bases, attrs
...         )
...
>>> class FooClass(metaclass=MyMeta):
...     pass
...
```

Metaclasses

```
>>> class MyMeta(type):
...     def __new__(meta, name, bases, attrs):
...         print('New {}'.format(name))
...         return super().__new__(
...             meta, name, bases, attrs
...         )
...
>>> class FooClass(metaclass=MyMeta):
...     pass
...
New FooClass
```

Metaclasses

```
>>> class MyMeta(type):
...     def __call__(cls, *args, **kwargs):
...         print('Call {}'.format(
...             cls.__name__
...         ))
...         return super().__call__(
...             *args, **kwargs
...         )
...
>>> class FooClass(metaclass=MyMeta):
...     pass
...
>>> f = FooClass()
Call FooClass
```

Metaclasses

"Metaclasses are deeper magic than 99% of users should ever 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)."

– Tim Peters

Metadog

Metadog

```
>>> class Dog():
...         def sit(self):
...                 print("*sitting*")
...
```

Metadog

```
>>> class Dog():
...         def sit(self):
...                 print("Growl!")
...                 print("*sitting*")
...
```

Metadog

```
>>> class Dog():
...         def sit(self):
...             print("Growl!")
...             print("*sitting*")
...         def stay(self):
...             print("Growl!")
...             print("*staying*")
...
```

Metadog

```
>>> class Dog():
...         def _woof(self):
...             print("Woof!")
...         def sit(self):
...             self._woof()
...             print("*sitting*")
...         def stay(self):
...             self._woof()
...             print("*staying*")
...
```

Metadog

```
>>> from functools import wraps
>>> def woof(f):
...     @wraps(f)
...     def wrapper(*args, **kwargs):
...         print('Woof!')
...         return f(*args, **kwargs)
...     return wrapper
...
```

Metadog

```
>>> class Dog():
...         @woof
...         def sit(self):
...             print("*sitting*")
...         @woof
...         def stay(self):
...             print("*staying*")
...
```

Metadog

```
>>> class Dog():
...         @woof
...         def sit(self):
...             print("*sitting*")
...         @woof
...         def stay(self):
...             print("*staying*")
...         def play_dead(self):
...             print("*playing_dead*")
...
```

Metadog

```
>>> from inspect import isfunction
>>>
```

Metadog

```
>>> from inspect import isfunction
>>> class MetaDog(type):
...
```

Metadog

```
>>> from inspect import isfunction
>>> class MetaDog(type):
...     def __new__(meta, name, bases, attrs):
...
```

Metadog

```
>>> from inspect import isfunction
>>> class MetaDog(type):
...     def __new__(meta, name, bases, attrs):
...         for name, attr in attrs.items():
...
```

Metadog

```
>>> from inspect import isfunction
>>> class MetaDog(type):
...     def __new__(meta, name, bases, attrs):
...         for name, attr in attrs.items():
...             if isfunction(attr):
...
```

Metadog

```
>>> from inspect import isfunction
>>> class MetaDog(type):
...     def __new__(meta, name, bases, attrs):
...         for name, attr in attrs.items():
...             if isfunction(attr):
...                 attrs[name] = woof(attr)
...
```

Metadog

```
>>> from inspect import isfunction
>>> class MetaDog(type):
...     def __new__(meta, name, bases, attrs):
...         for name, attr in attrs.items():
...             if isfunction(attr):
...                 attrs[name] = woof(attr)
...         return type.__new__(
...             meta, name, bases, attrs
...         )
...
```

Metadog

```
>>> class Dog(metaclass=MetaDog):
...         def sit(self):
...             print("(sitting)")
...         def stay(self):
...             print("(staying)")
...         def play_dead(self):
...             print("*playing_dead*")
...
```

Metadog

```
>>> my_dog = Dog()
```

Metadog

```
>>> my_dog = Dog()
>>> my_dog.sit()
Woof!
*sitting*
```

Metadog

```
>>> my_dog = Dog()
>>> my_dog.sit()
Woof!
*sitting*
>>> my_dog.play_dead()
Woof!
*playing dead*
```

Metadog

Metaclasses

```
>>> class FooClass():
...     pass
...
```

Metaclasses

```
>>> class FooClass():
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
False
```

Singleton

Singleton

```
>>> class Singleton():
...
```

```
>>> class FooClass(Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton():
...     _instance = None
...
```

```
>>> class FooClass(Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton():
...     _instance = None
...     def __new__(cls, *args, **kwargs):
...
```

```
>>> class FooClass(Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton():
...     _instance = None
...     def __new__(cls, *args, **kwargs):
...         if not cls._instance:
...
```

```
>>> class FooClass(Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton():
...     _instance = None
...     def __new__(cls, *args, **kwargs):
...         if not cls._instance:
...             cls._instance = object.__new__(
...                 cls, *args, **kwargs
...             )
...

>>> class FooClass(Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton():
...     _instance = None
...     def __new__(cls, *args, **kwargs):
...         if not cls._instance:
...             cls._instance = object.__new__(
...                 cls, *args, **kwargs
...             )
...         return cls._instance
...
>>> class FooClass(Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...
```

```
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...
```

```
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...         attrs['_instance'] = None
...
```

```
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...         attrs['_instance'] = None
...         return super().__new__(meta, name, bases, attrs)
...
```

```
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...         attrs['_instance'] = None
...         return super().__new__(meta, name, bases, attrs)
...     def __call__(cls, *args, **kwargs):
...
```

```
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...         attrs['_instance'] = None
...         return super().__new__(meta, name, bases, attrs)
...     def __call__(cls, *args, **kwargs):
...         if not cls._instance:
...
```

```
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...         attrs['_instance'] = None
...         return super().__new__(meta, name, bases, attrs)
...     def __call__(cls, *args, **kwargs):
...         if not cls._instance:
...             cls._instance = super().__call__(*args, **kwargs)
...

>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

```
>>> class Singleton(type):
...     def __new__(meta, name, bases, attrs):
...         attrs['_instance'] = None
...         return super().__new__(meta, name, bases, attrs)
...     def __call__(cls, *args, **kwargs):
...         if not cls._instance:
...             cls._instance = super().__call__(*args, **kwargs)
...         return cls._instance
...
>>> class FooClass(metaclass=Singleton):
...     pass
...
>>> a, b = FooClass(), FooClass()
>>> a is b
True
```

Singleton

Still irked, probably.

Still irked, probably.

```
>>> j = object()
>>> j.hi = 'there'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'object' object has no
attribute 'hi'
```

Still irked, probably.

```
>>> class FooClass():
...     pass
...
>>> hasattr(FooClass(), '__dict__')
True
>>> hasattr(object(), '__dict__')
False
```

Still irked, probably.

```
>>> from stackoverflow import getsize[1]
>>> getsize(object())
16
>>> getsize(0)
24
>>> getsize(dict())
280
>>> getsize(FooClass())
344
```

---

[1] http://stackoverflow.com/a/30316760/4842627

Conclusion

Conclusion

```
>>> j = type('', (), {})
```

Conclusion

```
>>> class MetaDog(type):
...
```

Conclusion

```
>>> class FooClass(metaclass=Singleton):
...
```

Conclusion

Thanks!

http://github.com/di