



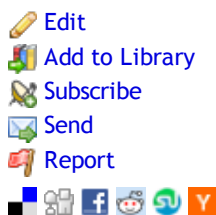
Stay up to date - [embedded code](#) automatically updates, [each snippet](#) and article has a feed  
[Join Siafoo Now](#) or [Learn More](#)



# Type Checking in Python



2

By [David](#)Updated over 3 years ago (09 May 2008 at 04:48 PM) [recent activity](#)

*You shouldn't type-check in Python, but if you do, know what method is best. We discuss the pros and cons of built-in functions 'type' and 'isinstance' and property '\_\_class\_\_'.*

Languages [Python](#)

## Python Programming Course

Hands-on Python developer training - public & on-site / custom options  
[www.frameworktraining.co.uk](http://www.frameworktraining.co.uk)



About ads on Siafoo

Need to know the type of an object? Let me make a brief argument: **No, you don't.** Just use the object as if it was whatever you expect it to be, and handle any errors that result.

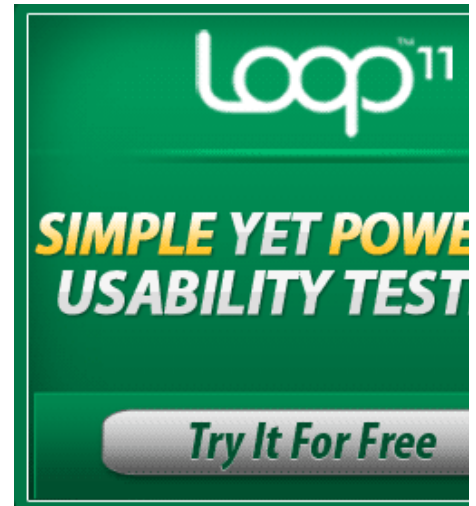
On the other hand, type-checking is convenient, easy to implement, and can save your life when debugging. Sometimes there *isn't* an obvious other way to attack a problem. (Look for one first though. Really.)

You have a few choices on how to type-check: the built-in functions `type` and `isinstance` and the instance property `__class__`. To compare the object to known classes, you can compare to the class directly or import the 'types' module (safe to do in your global scope) to get access to some types that don't provide user-accessible classes (like functions).

Before we start, note that Python has two types of classes: 'new-style' classes which inherit from the `object` class, and 'old-style' classes which don't. Most (all?) native Python types are now new-style classes, but classes you've created probably aren't.

Let's try some experiments, trying to print out the name of a class, and then comparing the type to a known class. Say we have the following classes and instances defined:

#'s



About ads on Siafoo



License

Creative Commons  
Attribution Non-  
Commercial Share Alike  
3.0-US

Keywords

[classes](#) (2) [type checking](#) (1)

Related

- [Python Tips, Tricks, and Hacks](#)
- [Python \\_\\_Underscore\\_\\_ Methods](#)
- [Python Decorators Don't Have to be \(that\) Scary](#)
- [Brainfuck Hello World](#)
- [BogoSort](#)

Permissions

- Owner: [David](#)
- Group Owner: [iCapsid](#)
- Viewable by **Everyone**
- Editable by **All Siafoo Users**

```

1 from types import *
2
3 # Instances of new-style classes
4 class NewClass(object): pass
5
6 new_class_instance = NewClass()
7
8 def function(): pass
9
10 string = 'This is a string!'
11
12 # Instances of old-style classes
13 class OldClass: pass
14
15 old_class_instance = OldClass()

```

Let's print out the name of the class using `type` and `__class__`. There's no way to do this with `isinstance`.

#'s

```

1 print type(new_class_instance)
2 print new_class_instance.__class__
3 # both print "<class '__main__.NewClass'>"
4
5 print type(function)
6 print function.__class__
7 # both print "<type 'function'>"
8
9 print type(string)
10 print string.__class__
11 # both print "<type 'str'>"
12
13 print type(old_class_instance)
14 # prints "<type 'instance'>"
15
16 print old_class_instance.__class__
17 # prints something like "class '__main__.OldClass' at 0x00F7F"
18 # Note that this is different than what was printed by type()

```

It looks like `type` doesn't work for old-style classes. Now let's try checking against a known class or type, using `type`, `isinstance`, and `__class__`:

#'s

```

1 type(new_class_instance) == NewClass
2 isinstance(new_class_instance, NewClass)
3 new_class_instance.__class__ == NewClass
4 # All return True
5
6 type(function) == FunctionType # from 'types' module
7 isinstance(function, FunctionType)
8 function.__class__ == FunctionType
9 # All return True
10
11 type(string) == str # we could also use StringType from 'types' module
12 isinstance(string, str)
13 string.__class__ == str
14 # All return True
15
16 type(old_class_instance) == OldClass
17 # Returns False, even though old_class_instance is an instance of OldClass.
18 # 'type' just can't understand these.
19
20 isinstance(old_class_instance, OldClass)
21 old_class_instance.__class__ == OldClass
22 # Both return True as expected

```

Apparently, `type` is completely useless for both type printing and type comparisons, as it can't understand old-style classes while `__class__` can.

`__class__` is messy, but is the only method that both works for both types of classes and will print out the name of the class if desired. It is the best choice for **debugging**, when you have no idea what type of object you're looking at.

`isinstance` also works on all objects. It also has another couple of perks: it actually checks to see if your object is an instance of a class *or subclass* of the class you pass it. Generally if you're type-checking you're interested in the existence of a behavior or method, and all subclasses of the target class will probably have it. So, `isinstance` is **more accurate**. Additionally, you

can pass it a tuple of classes, and it will check against **all** of them. These perks make it the best choice for **legitimate type-checking** in a real program.

Here's a summary of what we found:

Method	Works on old-style classes?	Need to know type before calling?	Best For
<code>type</code>	No	No	Nothing, really
<code>isinstance</code>	Yes	Yes	Legitimate type-checking: When you're checking type against a known class
<code>__class__</code>	Yes	No	Debugging: When you have no idea what class the instance is

[Add a Comment](#)

---

Problems? Questions? Ideas? Learn more [About Siafoo](#), [Get Help](#), or send us your [Feedback](#)

---

Copyright © 2011 Siafoo.net, All Rights Reserved. An Information Capsid project. [Privacy](#)  
Siafoo r10-dev-20110903    Timezone: US/Pacific