Type Checking in Python 2

By David

Updated over 3 years ago (09 May 2008 at 04:48 PM) recent activity

Edit
Add to Library
Subscribe
Send
Report

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

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

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**

```
from types import *

# Instances of new-style classes
class NewClass(object): pass

new_class_instance = NewClass()

def function(): pass

string = 'This is a string!'

# Instances of old-style classes
class OldClass: pass

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

```
print type(new_class_instance)
print new_class_instance.__class__
# both print "<class '__main__.new_class'>"

print type(function)
print function.__class__
# both print "<type 'function'>"

print type(string)
print string.__class__
# both print "<type 'str'>"

print type(old_class)
# prints "<type 'instance'>"

print old_class.__class__
# prints something like "class __main__.OldClass at 0x00F7F"
# 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

```
type(new_class_instance) == NewClass
isinstance(new_class_instance, NewClass)
new_class_instance.__class__ == NewClass
# All return True

type(function) == FunctionType # from 'types' module
isinstance(function, FunctionType)
function.__class__ == FunctionType
# All return True

type(string) == str # we could also use StringType from 'types' module
isinstance(string, str)
string.__class__ == str
# All return True

type(old_class_instance) == OldClass
# Returns False, even though old_class_instance is an instance of OldClass.
# 'type' just can't understand these.

isinstance(old_class_instance, OldClass)
old_class_instance.__class__ == OldClass
# 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
`type`	No	No	Nothing, really
`isinstance`	Yes	Yes	Legitimate type-checking: When you're checking type against a known class
`__class__`	Yes	No	Debugging: When you have no idea what class the instance is

Add a Comment