Python super class reflection

If I have Python code

```
class A():
    pass
class B():
    pass
class C(A, B):
    pass
```

and I have class `C`, is there a way to iterate through it's super classed (`A` and `B`)? Something like pseudocode:

```
>>> magicGetSuperClasses(C)
(<type 'A'>, <type 'B'>)
```

One solution seems to be inspect module and `getclasstree` function.

```
def magicGetSuperClasses(cls):
    return [o[0] for o in inspect.getclasstree([cls]) if type(o[0]) == type]
```

but is this a "Pythonian" way to achieve the goal?

python reflection

edited **Aug 25 '08 at 9:21**

asked **Aug 25 '08 at 9:06**

jelovirt
2,156 6 14 21
88% accept rate

add comment

start a bounty

5 Answers

`C.__bases__` is an array of the super classes, so you could implement your hypothetical function like so:

```
def magicGetSuperClasses(cls):
  return cls.__bases__
```

But I imagine it would be easier to just reference `cls.__bases__` directly in most cases.

edited **Sep 25 at 2:46**
Ethan Furman
1,844 3 21

answered **Aug 25 '08 at 9:22**
John
3,329 12 33

I'd +1 this if it were corrected per cdleary's point below. – Carl Meyer Feb 15 '09 at 19:32

add comment

@John: Your snippet doesn't work -- you are returning the *class* of the base classes (which are also known as metaclasses). You really just want `cls.__bases__`:

```
class A: pass
class B: pass
class C(A, B): pass

c = C() # Instance

assert C.__bases__ == (A, B) # Works
assert c.__class__.__bases__ == (A, B) # Works

def magicGetSuperClasses(clz):
  return tuple([base.__class__ for base in clz.__bases__])

assert magicGetSuperClasses(C) == (A, B) # Fails
```

Also, if you're using Python 2.4+ you can use generator expressions instead of creating a list (via []), then turning it into a tuple (via `tuple`). For example:

```
def get_base_metaclasses(cls):
    """Returns the metaclass of all the base classes of cls."""
    return tuple(base.__class__ for base in clz.__bases__)
```

That's a somewhat confusing example, but genexps are generally easy and cool. :)

edited **Feb 16 '09 at 9:50**

answered **Aug 29 '08 at 19:30**
cdleary
8,332 5 48 109

add comment

The inspect module was a good start, use the getmro function:

> Return a tuple of class cls's base classes, including cls, in method resolution order. No class appears more than once in this tuple. ...

```
>>> class A: pass
>>> class B: pass
>>> class C(A, B): pass
>>> import inspect
>>> inspect.getmro(C)[1:]
(<class __main__.A at 0x8c59f2c>, <class __main__.B at 0x8c59f5c>)
```

The first element of the returned tuple is `C`, you can just disregard it.

answered **Feb 16 '09 at 10:27**
Torsten Marek
12.8k 4 31 53
add comment

---

if you need to know to order in which super() would call the classes you can use `C.__mro__` and don't need `inspect` therefore.

answered **Feb 16 '09 at 16:38**
sinzi
69 2
add comment

I tried this function from cdleary's answer, and I got an error

```
def get_base_metaclasses(cls):
    """Returns the metaclass of all the base classes of cls."""
    return tuple(base.__class__ for base in cls.__bases__)
```

Here is my recursive solution:

```
class A:
    @classmethod
    def get_superclasses(cls):
        """Returns all superclasses of cls."""
        b = list(cls.__bases__)
        for base in b:
            b = b + base.get_superclasses()
        return b
class B(A): pass
class C(B): pass

c = C()
# This works both for an instance and for a class
print c.get_superclasses()
print C.get_superclasses()
```

returns

```
[<class __main__.B at 0xb7ca156c>, <class __main__.A at 0xb7ca144c>]
```

answered **Jun 4 '09 at 20:56**
Emma
252 1 4 8

2 Generator expressions require Python >= 2.4, perhaps that's why you got the error. If that's the case, you just need to add brackets inside the `tuple` invocation – cdleary Jul 10 '09 at 9:58

add comment

Not the answer you're looking for? Browse other questions tagged python reflection or ask your own question.