How can I access the current executing module or class name in Python? [duplicate]

Asked 15 years, 9 months ago Modified 11 months ago Viewed 155k times

73

This question already has answers here:

How to get a reference to a module inside the module itself? (8 answers)

Closed 6 months ago.

I would like to be able to dynamically retrieve the current executing module or class name from within an imported module. Here is some code:

foo.py:

```
def f():
    print __name__
```

bar.py:

```
from foo import f

def b(): f()
```

This obviously does not work as `__name__` is the name of the module that contains the function. What I would like to be access inside the `foo` module is the name of the current executing module that is using `foo`. So in the case above it would be `bar` but if any other module imported `foo` I would like `foo` to dynamically have access to the name of that module.

Edit: The `inspect` module looks quite promising but it is not exactly what I was looking for. What I was hoping for was some sort of global or environment-level variable that I could access that would contain the name of the current executing module. Not that I am unwilling to traverse the stack to find that information - I just thought that Python may have exposed that data already.

Edit: Here is how I am trying to use this. I have two different Django applications that both need to log errors to file. Lets say that they are called "AppOne" and "AppTwo". I also have a place to which I would like to log these files: "`/home/hare/app_logs`". In each application at any given point I would like to be able to import my logger module and call the log function which writes the log string to file. However what I would like to do is create a directory under `app_logs` that is the name of the current application

("AppOne" or "AppTwo") so that each application's log files will go in their respective logging directories.

In order to do this I thought that the best way would be for the logger module to have access to some sort of global variable that denotes the current application's name as it is responsible for knowing the location of the parent logging directory and creating the application's logging directory if it does not yet exist.

python module

Share

Improve this question

Follow

edited Mar 2, 2009 at 16:34

asked Mar 2, 2009 at 15:55

Andrew Hare

351k ● 75 ● 645 ● 641

Why would you want to break the allocation of responsibility like this? – S.Lott Mar 2, 2009 at 16:06

5 I am using this within a logging module - I want to be able to tell which outer module is attempting to log without the caller having to pass a key of some sort. – Andrew Hare Mar 2, 2009 at 16:10

Are you looking to find the module that started your program execution (ie: the module whose **name** == '**main**'? Or literally the module that called foo.f(), which may be different each time foo.f() is called? – Jarret Hardie Mar 2, 2009 at 16:18

1 @Andrew Hare: are you saying that (a) you wrote your own logger, different from the built-in logging module and (b) your own logger doesn't have a simple configuration class or function? And you are tying to work around this? – S.Lott Mar 2, 2009 at 16:46

3 @S.Lott - a) yes, b) yes, and no workaround at all - just learning the ropes a bit. – Andrew Hare Mar 2, 2009 at 16:57

11 Answers

Sorted by: Highest score (default)

76

From the comment -- not the question.

> I am simply curious to see if what I am trying to do is possible.

The answer to "is it possible" is always "yes". Always. Unless your question involves time travel, anti-gravity or perpetual motion.

Since the answer is always "yes", your question is ill-formed. The real question is "what's a good way to have my logging module know the name of the client?" or something like that.

The answer is "Accept it as a parameter." Don't mess around with inspecting or looking for mysterious globals or other tricks.

Just follow the design pattern of `logging.getLogger()` and use explicitly-named loggers. A common idiom is the following

```
logger= logging.getLogger( __name__ )
```

That handles almost all log naming perfectly.

Share

Improve this answer

Follow

edited Feb 25, 2022 at 13:25

Neuron

5,763 ● 5 ● 43 ● 62

answered Mar 2, 2009 at 18:36

S.Lott

391k ● 82 ● 517 ● 788

3 I think you are right in that I will be better off adjusting my approach - thanks! – Andrew Hare Mar 2, 2009 at 19:34

24 Though if I run the script instead of import it, `__name__` will contain `__main__` instead of the real path to the script (like `modA.submodB.pack1`). Is there a value that *always* return the module path regardless of how it's called? – estani Aug 13, 2012 at 8:25

2 @estani: if you run it as a script, its name will *be* `__main__`. There is no such thing as a module "real" name, or `modA.submodB.pack1` when running it this way. Do not confuse module hierarchy with filenames. – MestreLion Nov 28, 2014 at 11:27

9 "The answer to 'is it possible' is always 'yes'." Or solving NP problems in polynomial time. Then it's "maybe, but most people think not". ;) – jpmc26 Jan 30, 2017 at 18:37

15 Actually there is an antigravity module in python (though not sure for time travel and perpetual motion) – Tanguy Feb 22, 2018 at 16:23

67

This should work for referencing the current module:

```
import sys
sys.modules[__name__]
```

Share Improve this answer Follow

answered Oct 18, 2011 at 16:27

newtover

32k ● 11 ● 88 ● 89

5 Then you can use `import sys; module_name = vars(sys.modules[__name__])['__package__']` to get the name of the current module, or `None` if not a module. Ex: If the code is executed with `python foo/bar.py`, *module_name* will be `None`; but with `python -m foo.bar`, *module_name* will be `"foo"`. – roipoussiere Sep 17, 2019 at 14:30

24

To obtain a reference to the `__main__` module when in another:

```
import sys
sys.modules['__main__']
```

To then obtain the module's file path, which includes its name:

```
sys.modules['__main__'].__file__  # type: str
```

If within the `__main__` module, simply use: `__file__`

To obtain just the file name from the file path:

```
import os
os.path.basename(file_path)
```

To separate the file name from its extension:

```
file_name.split(".")[0]
```

To obtain the name of a class instance:

```
instance.__class__.__name__
```

To obtain the name of a class:

```
class.__name__
```

Share

Improve this answer

Follow

edited Jan 5 at 20:25

wjandrea

32.6k ● 9 ● 67 ● 94

answered Dec 20, 2016 at 0:30

Sjshovan

320 ● 2 ● 5

1 using **file** does not work if you are inside a package if you have foo/__init__.py, **file** will return **init**.py instead of foo – Mart10 Nov 5, 2020 at 16:15

23

The "currently executing module" clearly is foo, as that's what contains the function currently running - I think a better description as to what you want is the module of foo's immediate caller (which may itself be foo if you're calling a f() from a function in foo called by a function in bar. How far you want to go up depends on what you want this for.

In any case, assuming you want the immediate caller, you can obtain this by walking up the call stack. This can be accomplished by calling `sys._getframe`, with the aprropriate number of levels to walk.

```
def f():
    caller = sys._getframe(1)  # Obtain calling frame
    print "Called from module", caller.f_globals['__name__']
```

[Edit]: Actually, using the inspect module as suggested above is probably a cleaner way of obtaining the stack frame. The equivalent code is:

```
def f():
    caller = inspect.currentframe().f_back
    print "Called from module", caller.f_globals['__name__']
```

(sys._getframe is documented as being for internal use - the inspect module is a more reliable API)

Share

Improve this answer

Follow

edited Mar 2, 2009 at 16:33

answered Mar 2, 2009 at 16:27

Brian

119k ● 29 ● 110 ● 114

19

`__file__` is the path of current module the call is made.

Share Improve this answer Follow

answered Mar 2, 2009 at 18:49

utku_karatas

6,285 ● 4 ● 42 ● 52

Adding to this solution. In order to obtain the string filename on Windows. Forgive the lack of newlines. `python module_name, ext = os.path.splitext(__file__) module_name = module_name.split('\\')[-1]` – MinneapolisCoder9 Oct 21, 2022 at 14:44

11

I think what you want to use is the inspect module, to inspect the python runtime stack. Check out this tutorial. I think it provides an almost exact example of what you want to do.

Share

Improve this answer

Follow

edited Mar 2, 2017 at 16:01

answered Mar 2, 2009 at 16:04

nstehr

8,128 ● 2 ● 19 ● 22

7

Using `__file__` alone gives you a relative path for the main module and an absolute path for imported modules. Being aware this we can get the module file constantly either way with a little help from our `os.path` tools.

For filename only use `__file__.split(os.path.sep)[-1]`.

For complete path use `os.path.abspath(__file__)`.

Demo:

```
/tmp $ cat f.py
from pprint import pprint
import os
import sys

pprint({
    'sys.modules[__name__]': sys.modules[__name__],
    '__file__': __file__,
    '__file__.split(os.path.sep)[-1]': __file__.split(os.path.sep)[-1],
    'os.path.abspath(__file__)': os.path.abspath(__file__),
})

/tmp $ cat i.py
import f
```

Results:

```
## on *Nix ##

/tmp $ python3 f.py
{'sys.modules[__name__]': <module '__main__' from 'f.py'>,
 '__file__': 'f.py',
 '__file__.split(os.path.sep)[-1]': 'f.py',
 'os.path.abspath(__file__)': '/tmp/f.py'}

/tmp $ python3 i.py
{'sys.modules[__name__]': <module 'f' from '/tmp/f.pyc'>,
 '__file__': '/tmp/f.pyc',
 '__file__.split(os.path.sep)[-1]': 'f.pyc',
 'os.path.abspath(__file__)': '/tmp/f.pyc'}

## on Windows ##

PS C:\tmp> python3.exe f.py
{'sys.modules[__name__]': <module '__main__' from 'f.py'>,
 '__file__': 'f.py',
 '__file__.split(os.path.sep)[-1]': 'f.py',
 'os.path.abspath(__file__)': 'C:\\tools\\cygwin\\tmp\\f.py'}

PS C:\tmp> python3.exe i.py
{'sys.modules[__name__]': <module 'f' from 'C:\\tools\\cygwin\\tmp\\f.py'>,
 '__file__': 'C:\\tools\\cygwin\\tmp\\f.py',
 '__file__.split(os.path.sep)[-1]': 'f.py',
 'os.path.abspath(__file__)': 'C:\\tools\\cygwin\\tmp\\f.py'}
```

If you want to strip the '.py' off the end, you can do that easily. (But don't forget that you may run a '.pyc' instead.)

Share Improve this answer Follow

answered Jan 9, 2017 at 18:02

Bruno Bronosky

70.1k ● 14 ● 172 ● 152

4 any reason to `__file__.split(os.path.sep)[-1]` instead of `os.path.basename(__file__)` ? – cowbert Jul 10, 2017 at 20:20

If you want only the name of the file:

```
file_name = __file__.split("/")[len(__file__.split("/"))-1]
```

Share Improve this answer Follow

answered Nov 3, 2017 at 14:06

RiccardoCh

1,110 ● 1 ● 14 ● 24

3 For a cross-platform solution, consider using `os.path`. Also, you can just use `-1` as the index. – wizzwizz4 Jul 9, 2018 at 19:12

I don't believe that's possible since that's out of `foo`'s scope. foo will only be aware of its internal scope since it may be being called by countless other modules and applications.

Share Improve this answer Follow

answered Mar 2, 2009 at 15:59

Soviut

91.4k ● 53 ● 205 ● 279

It's been a while since I've done python, but I believe that you can get access to the globals and locals of a caller through its traceback.

Share Improve this answer Follow

answered Mar 2, 2009 at 16:28

Mr Fooz

112k ● 7 ● 76 ● 103

To get the current file module, containing folder, here is what worked for me:

```
import os
parts = os.path.splitext(__name__)
module_name = parts[len(parts) - 2]
```

Share Improve this answer Follow

answered Mar 18, 2019 at 17:42

sveilleux2

1,512 ● 12 ● 16