The __name__ special variable All Versions

The `__name__` special variable is used to check whether a file has been imported as a module or not, and to identify a function, class, module object by their `__name__` attribute.

Examples

__name__ == '__main__'

The special variable `__name__` is not set by the user. It is mostly used to check whether or not the module is being run by itself or run because an `import` was performed. To avoid your module to run certain parts of its code when it gets imported, check if `__name__ == '__main__'`.

Let **module_1.py** be just one line long:

```
import module2.py
```

And let's see what happens, depending on **module2.py**

Situation 1

module2.py

```
print('hello')
```

Running **module1.py** will print `hello`
Running **module2.py** will print `hello`

Situation 2

module2.py

```
if __name__ == '__main__':
    print('hello')
```

Running **module1.py** will print nothing
Running **module2.py** will print `hello`

function_class_or_module.__name__

The special attribute `__name__` of a function, class or module is a string containing its name.

```
import os

class C:
    pass

def f(x):
    x += 2
    return x


print(f)
# <function f at 0x029976B0>
print(f.__name__)
# f

print(C)
# <class '__main__.C'>
print(C.__name__)
# C

print(os)
# <module 'os' from '/spam/eggs/'>
print(os.__name__)
# os
```

The `__name__` attribute is not, however, the name of the variable which references the class, method or function, rather it is the name given to it when defined.

```
def f():
    pass

print(f.__name__)
# f - as expected

g = f
```

```
print(g.__name__)
# f - even though the variable is named g, the function is still named f
```

This can be used, among others, for debugging:

```
def enter_exit_info(func):
    def wrapper(*arg, **kw):
        print '-- entering', func.__name__
        res = func(*arg, **kw)
        print '-- exiting', func.__name__
        return res
    return wrapper

@enter_exit_info
def f(x):
    print 'In:', x
    res = x + 2
    print 'Out:', res
    return res

a = f(2)

# Outputs:
#     -- entering f
#     In: 2
#     Out: 4
#     -- exiting f
```

Use in logging

When configuring the built-in `logging` functionality, a common pattern is to create a logger with the `__name__` of the current module:

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

This means that the fully-qualified name of the module will appear in the logs, making it easier to see where messages have come from.

Syntax

Parameters

Remarks

The Python special variable `__name__` is set to the name of the containing module. At the top level (such as in the interactive interpreter, or in the main file) it is set to `'__main__'`. This can be used to run a block of statements if a module is being run directly rather than being imported.

The related special attribute `obj.__name__` is found on classes, imported modules and functions *(including methods)* , and gives the name of the object when defined.