[Using the Platform module in Python](http://www.pythonforbeginners.com/systems-programming/how-to-use-the-platform-module-in-python/)

16 Jan 2013   | [System & OS](http://www.pythonforbeginners.com/category/systems-programming/)

What is it used for?

The platform module in Python is used to access the underlying platform’s data,

such as, hardware, operating system, and interpreter version information.

The platform module includes tools to see the platform’s hardware, operating

system, and interpreter version information where the program is running.

How do I use it?

You start with importing the platform module in your program, like this:

**import platform**

You then specify what you want to find out (more on that below).

For example, if you want to find out which python version you are using,

simply add python\_version() to the platform, like so:

**print platform.python\_version()**

This will return the Python version as a string.

On my computer it looks like this:

**2.7.3**

Platform Functions

Let's have a look at the different platform functions we can use

**platform.architecture()**

returns information about the bit architecture

**platform.machine()**

returns the machine type, e.g. 'i386'.

**platform.node()**

returns the computer’s network name (may not be fully qualified!)

**platform.platform()**

returns a single string identifying the underlying platform with as much useful

information as possible.

**platform.processor()**

returns the (real) processor name, e.g. 'amdk6'.

**platform.python\_build()**

returns a tuple (buildno, builddate) stating the Python build number and

date as strings.

**platform.python\_compiler()**

returns a string identifying the compiler used for compiling Python.

**platform.python\_version()**

returns the Python version as string 'major.minor.patchlevel'

**platform.python\_implementation()**

returns a string identifying the Python implementation.

Possible return values are: ‘CPython’, ‘IronPython’, ‘Jython’, ‘PyPy’.

**platform.release()**

returns the system’s release, e.g. '2.2.0' or 'NT'

**platform.system()**

returns the system/OS name, e.g. 'Linux', 'Windows', or 'Java'.

**platform.version()**

returns the system’s release version, e.g. '#3 on degas'

**platform.uname()**

returns a tuple of strings (system, node, release, version, machine, processor)

identifying the underlying platform.

Operating System and Hardware Info

Let's show some examples with an output, to see how this works.

(you can find more examples [here](http://www.doughellmann.com/PyMOTW/platform/))

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | import platform    print 'uname:', platform.uname()    print  print 'system   :', platform.system()  print 'node     :', platform.node()  print 'release  :', platform.release()  print 'version  :', platform.version()  print 'machine  :', platform.machine()  print 'processor:', platform.processor() |

Output

uname: ('Linux', 'Mwork', '3.5.0-21-generic', '#32-Ubuntu SMP Tue Dec 11 18:51:59

UTC 2012', 'x86\_64', 'x86\_64')

system : Linux

node : Mwork

release : 3.5.0-21-generic

version : #32-Ubuntu SMP Tue Dec 11 18:51:59 UTC 2012

machine : x86\_64

processor: x86\_64

Sources

Please see the following URL's for more information

<http://docs.python.org/2/library/platform.html>

<http://www.doughellmann.com/PyMOTW/platform/>