platform — System Version Information

Purpose: Probe the underlying platform's hardware, operating system, and interpreter version information.

Although Python is often used as a cross-platform language, it is occasionally necessary to know what sort of system a program is running on. Build tools need that information, but an application might also know that some of the libraries or external commands it uses have different interfaces on different operating systems. For example, a tool to manage the network configuration of an operating system can define a portable representation of network interfaces, aliases, IP addresses, etc. But when the time comes to edit the configuration files, it must know more about the host so it can use the correct operating system configuration commands and files. The `platform` module includes the tools for learning about the interpreter, operating system, and hardware platform where a program is running.

Note

The example output in this section was generated on three systems: a Mac mini running macOS 10.14, a Dell PC running Ubuntu Linux 14.04, and a VirtualBox VM running Windows 10. Python was installed on the OS X and Windows systems using the pre-compiled installers from python.org. The Linux system is running a version in a system package.

Special thanks to Patrick Kettner (@patrickkettner) for helping to collect the example output on Windows.

Interpreter

There are four functions for getting information about the current Python interpreter. `python_version()` and `python_version_tuple()` return different forms of the interpreter version with major, minor, and patch level components. `python_compiler()` reports on the compiler used to build the interpreter. And `python_build()` gives a version string for the build of the interpreter.

```
# platform_python.py

import platform

print('Version      :', platform.python_version())
print('Version tuple:', platform.python_version_tuple())
print('Compiler     :', platform.python_compiler())
print('Build        :', platform.python_build())
```

OS X:

```
$ python3 platform_python.py

Version      : 3.7.1
Version tuple: ('3', '7', '1')
Compiler     : Clang 6.0 (clang-600.0.57)
Build        : ('v3.7.1:260ec2c36a', 'Oct 20 2018 03:13:28')
```

Linux:

```
$ python3 platform_python.py

Version      : 3.5.2
Version tuple: ('3', '5', '2')
Compiler     : GCC 4.8.4
Build        : ('default', 'Jul 17 2016 00:00:00')
```

Windows:

```
C:\>Desktop\platform_python.py

Version      : 3.5.1
Version tuple: ('3', '5', '1')
Compiler     : MSC v.1900 64 bit (AMD64)
```

```
Build        : ('v3.5.1:37a07cee5969', 'Dec  6 2015 01:54:25')
```

Platform

The `platform()` function returns a string containing a general purpose platform identifier. The function accepts two optional Boolean arguments. If `aliased` is True, the names in the return value are converted from a formal name to their more common form. When `terse` is true, a minimal value with some parts dropped is returned instead of the full string.

```
# platform_platform.py

import platform

print('Normal :', platform.platform())
print('Aliased:', platform.platform(aliased=True))
print('Terse  :', platform.platform(terse=True))
```

OS X:

```
$ python3 platform_platform.py

Normal : Darwin-18.0.0-x86_64-i386-64bit
Aliased: Darwin-18.0.0-x86_64-i386-64bit
Terse  : Darwin-18.0.0
```

Linux:

```
$ python3 platform_platform.py

Normal : Linux-3.13.0-55-generic-x86_64-with-Ubuntu-14.04-trusty
Aliased: Linux-3.13.0-55-generic-x86_64-with-Ubuntu-14.04-trusty
Terse  : Linux-3.13.0-55-generic-x86_64-with-glibc2.9
```

Windows:

```
C:\>platform_platform.py

Normal : Windows-10-10.0.10240-SP0
Aliased: Windows-10-10.0.10240-SP0
Terse  : Windows-10
```

Operating System and Hardware Info

More detailed information about the operating system and hardware the interpreter is running under can be retrieved as well. `uname()` returns a tuple containing the system, node, release, version, machine, and processor values. Individual values can be accessed through functions of the same names, listed in the table below.

Platform Information Functions

Function	Return Value
`system()`	operating system name
`node()`	host name of the server, not fully qualified
`release()`	operating system release number
`version()`	more detailed system version
`machine()`	a hardware-type identifier, such as `'i386'`
`processor()`	a real identifier for the processor (the same value as `machine()` in many cases)

```
# platform_os_info.py

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())
```

OS X:

```
$ python3 platform_os_info.py

uname: uname_result(system='Darwin', node='hubert.local',
release='18.0.0', version='Darwin Kernel Version 18.0.0: Wed Aug
22 20:13:40 PDT 2018; root:xnu-4903.201.2~1/RELEASE_X86_64',
machine='x86_64', processor='i386')

system   : Darwin
node     : hubert.local
release  : 18.0.0
version  : Darwin Kernel Version 18.0.0: Wed Aug 22 20:13:40 PDT
2018; root:xnu-4903.201.2~1/RELEASE_X86_64
machine  : x86_64
processor: i386
```

Linux:

```
$ python3 platform_os_info.py

uname: uname_result(system='Linux', node='apu',
release='3.13.0-55-generic', version='#94-Ubuntu SMP Thu Jun 18
00:27:10 UTC 2015', machine='x86_64', processor='x86_64')

system   : Linux
node     : apu
release  : 3.13.0-55-generic
version  : #94-Ubuntu SMP Thu Jun 18 00:27:10 UTC 2015
machine  : x86_64
processor: x86_64
```

Windows:

```
C:\>Desktop\platform_os_info.py

uname: uname_result(system='Windows', node='IE11WIN10',
release='10', version='10.0.10240', machine='AMD64',
processor='Intel64 Family 6 Model 70 Stepping 1, GenuineIntel')

system   : Windows
node     : IE11WIN10
release  : 10
version  : 10.0.10240
machine  : AMD64
processor: Intel64 Family 6 Model 70 Stepping 1, GenuineIntel
```

Executable Architecture

Individual program architecture information can be probed using the `architecture()` function. The first argument is the path to an executable program (defaulting to `sys.executable`, the Python interpreter). The return value is a tuple containing the bit architecture and the linkage format used.

```
# platform_architecture.py

import platform

print('interpreter:', platform.architecture())
print('/bin/ls    :', platform.architecture('/bin/ls'))
```

OS X:

```
$ python3 platform_architecture.py

interpreter: ('64bit', '')
/bin/ls    : ('64bit', '')
```

Linux:

```
$ python3 platform_architecture.py

interpreter: ('64bit', 'ELF')
/bin/ls    : ('64bit', 'ELF')
```

Windows:

```
C:\>Desktop\platform_architecture.py

interpreter: ('64bit', 'WindowsPE')
/bin/ls    : ('64bit', '')
```

See also

- Standard library documentation for platform
- Python 2 to 3 porting notes for platform

os — Portable access to operating system specific features

resource — System Resource Management

Quick Links

Interpreter
Platform
Operating System and Hardware Info
Executable Architecture

This page was last updated 2018-12-09.

Navigation

os — Portable access to operating system specific features
resource — System Resource Management

The output from all the example programs from PyMOTW-3 has been generated with Python 3.7.1, unless otherwise noted. Some of the features described here may not be available in earlier versions of Python.

Looking for examples for Python 2?