



Модуль platform в Python, информация о системе



Развивайте и масштабируйте бизнес легко



mango-office.ru

Виртуальная АТС Расширенная

До 120 функций. Когда нужно распределять звонки по группам сотрудников и по типам...

Узнать больше

РЕКЛАМА

[Стандартная библиотека Python3.](#) / Модуль platform в Python, информация о системе

Различная информацию об операционной системе и языке Python

Python часто используется в качестве кроссплатформенного языка и иногда необходимо знать, на какой системе работает программа.

Например, инструмент для управления сетевой конфигурацией операционной системы может определять переносимое представление сетевых интерфейсов, псевдонимов, IP-адресов и т. д., но когда приходит время редактировать файлы конфигурации, то Python должен знать больше о хосте, чтобы использовать правильные команды и файлы конфигурации операционной системы.

[Модуль platform](#) включает в себя инструменты для получения сведений об аппаратной платформе, операционной системе и интерпретаторе на которой выполняется программа.

Содержание:

- Кроссплатформенные функции:
 - [Информации об архитектуре](#) `platform.architecture()`,
 - [Тип машины](#) `platform.machine()`,
 - [Сетевое имя компьютера](#) `platform.node()`,
 - [Сведения о базовой платформе](#) `platform.platform()`,
 - [Реальное имя процессора](#) `platform.processor()`,
 - [Номер и дата сборки Python](#) `platform.python_build()`,
 - [Версия компилятора](#) `platform.python_compiler()`,
 - [Ветвь SCM реализации Python](#) `platform.python_branch()`,
 - [Реализация Python](#) `platform.python_implementation()`,
 - [Ревизия SCM реализации Python](#) `platform.python_revision()`,
 - [Версия Python как строка](#) `platform.python_version()`,
 - [Версия Python как кортеж](#) `platform.python_version_tuple()`,
 - [Сведения о выпуске системы](#) `platform.release()`,
 - [Имя операционной системы](#) `platform.system()`,
 - `platform.system_alias()`,
 - [Версия выпуска системы](#) `platform.version()`,
 - [Сведения команды терминала uname](#) `platform.uname()`,
- Функции платформы Java:
 - [Версия интерфейса для Jython](#) `platform.java_ver()`,
- Функции платформы Windows:
 - [Информация о версии из реестра Windows](#) `platform.win32_ver()`,
 - [Текущая редакция Windows](#) `platform.win32_edition()`,
 - [True, если Windows, распознается как IoT](#) `platform.win32_is_iot()`,
- Функции платформы Mac OS:
 - [Информация о версии Mac OS](#) `platform.mac_ver()`,
- Функции платформы Unix:
 - [Версия библиотеки libc](#) `platform.libc_ver()`,
- Функции платформы Linux:
 - [Идентификатор ОС из os-release](#) `platform.freedesktop_os_release()` (доступна в Python 3.10).

Кроссплатформенные функции.

platform.architecture(executable=sys.executable, bits='', linkage=''):

Функция platform.architecture() запрашивает исполняемый файл executable для получения различной информации об архитектуре. По умолчанию двоичный файл интерпретатора Python.

Возвращает [кортеж](#) (bits, linkage), который содержит информацию об архитектуре битов и формате связи, используемом для исполняемого файла. Оба значения возвращаются в виде [строк](#).

Значения, которые не могут быть определены, возвращаются в соответствии с предустановками параметров. Если биты заданы как '', то sizeof(pointer) используется в качестве индикатора для поддерживаемого размера указателя.

Функция полагается на системную команду file для выполнения реальной работы. Это доступно на большинстве, если не на всех платформах Unix и некоторых не-Unix платформах и только в том случае, если исполняемый файл указывает на интерпретатор Python.

```
>>> import platform
>>> platform.architecture()
# ('64bit', 'ELF')
```

Примечание.

В Mac OS X (и, возможно, на других платформах) исполняемые файлы могут быть универсальными файлами с несколькими архитектурами. Чтобы получить 64-разрядность текущего интерпретатора, более надежно запросить атрибут [sys.maxsize](#):

```
is_64bits = sys.maxsize > 2**32
```

platform.machine():

Функция platform.machine() возвращает тип машины, например 'I386'. Если значение не может быть определено, то возвращается пустая строка.

```
>>> import platform
>>> platform.machine()
# 'x86_64'
```

platform.node():

Функция platform.node() возвращает сетевое имя компьютера, может быть не полностью! Если значение не может быть определено, то возвращается пустая строка..

```
>>> import platform
>>> platform.node()
# 'IdeaCentre'
```

platform.platform(aliased=0, terse=0):

Функция platform.platform() возвращает одну строку, идентифицирующую базовую платформу, с максимально возможным количеством полезной информации.

Вывод предназначен для чтения человеком, а не для машинного анализа. Это может выглядеть по-разному на разных платформах, и это предназначено.

Если aliased имеет значение True, функция будет использовать псевдонимы для различных платформ, которые сообщают имена систем, которые отличаются от их общих имен, например SunOS будет сообщаться как Solaris.

При установке значения terse в значение True функция возвращает только абсолютный минимум информации, необходимой для идентификации платформы.

```
>>> import platform
>>> platform.platform()
# 'Linux-5.3.0-51-generic-x86_64-with-Ubuntu-18.04-bionic'
```

platform.processor():

Функция platform.processor() возвращает реальное имя процессора, например 'Amdk6'.

Возвращается пустая строка, если значение не может быть определено. Обратите внимание, что многие платформы не предоставляют эту информацию или просто возвращают то же значение, что и для [platform.machine\(\)](#).

```
>>> import platform
>>> platform.processor()
# 'x86_64'
```

platform.python_build():

Функция platform.python_build() возвращает кортеж (buildno, builddate) с указанием номера и даты сборки Python в виде строк.

```
>>> import platform
>>> platform.python_build()
# ('default', 'Apr 18 2020 01:56:04')
```

platform.python_compiler():

Функция platform.python_compiler() возвращает строку, идентифицирующую компилятор, используемый для компиляции Python.

```
>>> import platform
>>> platform.python_compiler()
# 'GCC 8.4.0'
```

platform.python_branch():

Функция platform.python_branch() возвращает строку, идентифицирующую ветвь SCM реализации Python.

```
>>> import platform
>>> platform.python_branch()
# ''
```

platform.python_implementation():

Функция platform.python_implementation() возвращает строку, идентифицирующую реализацию Python. Возможные возвращаемые значения: 'CPython', 'IronPython', 'Jython', 'PyPy'.

```
>>> import platform
>>> platform.python_implementation()
# 'CPython'
```

platform.python_revision():

Функция platform.python_revision() возвращает строку, идентифицирующую ревизию SCM реализации Python.

```
>>> import platform
>>> platform.python_revision()
# ''
```

platform.python_version():

Функция platform.python_version() возвращает версию Python в виде строки 'major.minor.patchlevel'.

Обратите внимание, что в отличие от Python [sys.version\(\)](#), возвращаемое значение всегда будет включать уровень исправления (по умолчанию 0).

```
>>> import platform
>>> platform.python_version()
# '3.7.4'
```

platform.python_version_tuple():

Функция platform.python_version_tuple() возвращает версию Python как [кортеж](#) (major, minor, patchlevel) строк.

Обратите внимание, что в отличие от Python sys.version, возвращаемое значение всегда будет включать уровень исправления (по умолчанию он равен 0).

```
>>> import platform
>>> platform.python_version_tuple()
# ('3', '7', '4')
```

platform.release():

Функция platform.release() возвращает выпуск системы, например, '2.2.0' или 'NT'. Если значение не может быть определено, то возвращается пустая строка.

```
>>> import platform
>>> platform.release()
# '5.3.0-51-generic'
```

platform.system():

Функция platform.system() возвращает имя операционной системы такое как 'Linux', 'Darwin', 'Java', 'Windows'. Если значение не может быть определено, то возвращается пустая строка.

```
>>> import platform
>>> platform.system()
# 'Linux'
```

platform.system_alias(system, release, version):

Функция platform.system_alias() возвращает (system, release, version), связанные с общими маркетинговыми именами, используемыми в некоторых системах.

```
>>> import platform
>>> platform.system_alias('Ubuntu', '18', '4')
# ('Ubuntu', '18', '4')
```

platform.version():

Функция platform.version() возвращает версию выпуска системы, например '#3 on degas'. Если значение не может быть определено, то возвращается пустая строка.

```
>>> import platform
>>> platform.version()
'#44~18.04.2-Ubuntu SMP Thu Apr 23 14:27:18 UTC 2020'
```

platform.uname():

Функция platform.uname() представляет портативный интерфейс команды uname. Возвращает [именованный кортеж](#), содержащий шесть атрибутов: [system](#), [node](#), [release](#), [version](#), [machine](#) и [processor](#).

Обратите внимание, что функция platform.uname() добавляет шестой атрибут ([processor](#)), отсутствующий в результате [os.uname\(\)](#). Кроме того, имена атрибутов отличаются для первых двух атрибутов. Функция os.uname() называет их sysname и nodename.

Записи, которые не могут быть определены, выводятся как ''.

```
>>> import platform
>>> platform.uname()
# uname_result(system='Linux', node='IdeaCentre', release='5.3.0-51-generic',
# version='#44~18.04.2-Ubuntu SMP Thu Apr 23 14:27:18 UTC 2020',
# machine='x86_64', processor='x86_64')
```

Функции платформы Java

platform.java_ver(release='', vendor='', vminfo=('', '', ''), osinfo=('', '', '')):

Функция platform.java_ver() возвращает версию интерфейса для Jython.

Возвращает [кортеж](#) (release, vendor, vminfo, osinfo), где vminfo является кортежем (vm_name, vm_release, vm_vendor), а osinfo - кортежем (os_name, os_version, os_arch). Значения, которые не могут быть определены, устанавливаются на значения по умолчанию - пустая строка ''.

Функции платформы Windows

platform.win32_ver(release='', version='', csd='', ptype=''):

Функция platform.win32_ver() получает дополнительную информацию о версии из реестра Windows и возвращает [кортеж](#) (release, version, csd, ptype), относящийся к выпуску ОС, номеру версии, уровню CSD (пакет обновления) и типу ОС (многопроцессорный/однопроцессорный).

Подсказка: ptype это 'Uniprocessor Free' на однопроцессорных машинах NT и 'Multiprocessor Free' на многопроцессорных машинах. 'Free' относится к версии ОС, свободной от кода отладки. Он также может указывать 'Checked', что означает, что версия ОС использует код отладки, то есть код, который проверяет аргументы, диапазоны и т. д.

Примечание. Эта функция лучше всего работает с установленным пакетом win32all Марка Хаммонда. Очевидно, что функция platform.win32_ver() работает только на Win32-совместимых платформах.

platform.win32_edition():

Функция platform.win32_edition() возвращает [строку](#), представляющую текущую редакцию Windows. Возможные значения включают, но не ограничиваются: 'Enterprise', 'IoTAP', 'ServerStandard' и 'nanoserver'.

platform.win32_is_iot():

Функция platform.win32_is_iot() возвращает True, если версия Windows, возвращенная [platform.win32_edition\(\)](#), распознается как версия IoT.

Функции платформы Mac OS.

platform.mac_ver(release='', versioninfo=('', '', ''), machine=''):

Функция platform.mac_ver() Получите информацию о версии Mac OS и возвращает ее как [кортеж](#) (release, versioninfo, machine), а versioninfo так-же является кортежем (version, dev_stage, non_release_version).

Записи, которые не могут быть определены, выводятся как пустая строка ''. Все записи кортежа являются [строками](#).

Функции платформы Unix.

platform.libc_ver(executable=sys.executable, lib='', version='', chunksize=16384):

Функция platform.libc_ver() пытается определить версию libc, с которой связан исполняемый файл, по умолчанию интерпретатор Python.

Возвращает [кортеж строк](#) (lib, version), которые по умолчанию соответствуют заданным параметрам в случае сбоя поиска.

Обратите внимание, что эта функция хорошо знает, как разные версии libc добавляют символы к исполняемому файлу. Можно использовать только для исполняемых файлов, скомпилированных с использованием gcc.

Файл читается и сканируется кусками байтов.

```
>>> import platform
>>> platform.libc_ver()
# ('glibc', '2.26')
```

Функции платформы Unix.

platform.freedesktop_os_release():

Функция platform.freedesktop_os_release() (доступна в Python 3.10) получает идентификатор операционной системы из файла os-release и возвращает его как [словарь dict](#). Файл os-release является стандартом freedesktop.org и доступен в большинстве дистрибутивов Linux. Заметным исключением являются дистрибутивы на базе Android.

Вызывает [ошибку OSError](#) или подкласс OSError, если невозможно прочитать ни /etc/os-release, ни /usr/lib/os-release.

В случае успеха функция возвращает словарь, в котором ключи и значения являются [строками](#). В значениях есть специальные символы, такие как " и \$ без кавычек. Поля NAME, ID и PRETTY_NAME всегда определяются в соответствии со стандартом. Все остальные поля являются необязательными. Поставщики могут включать дополнительные поля.

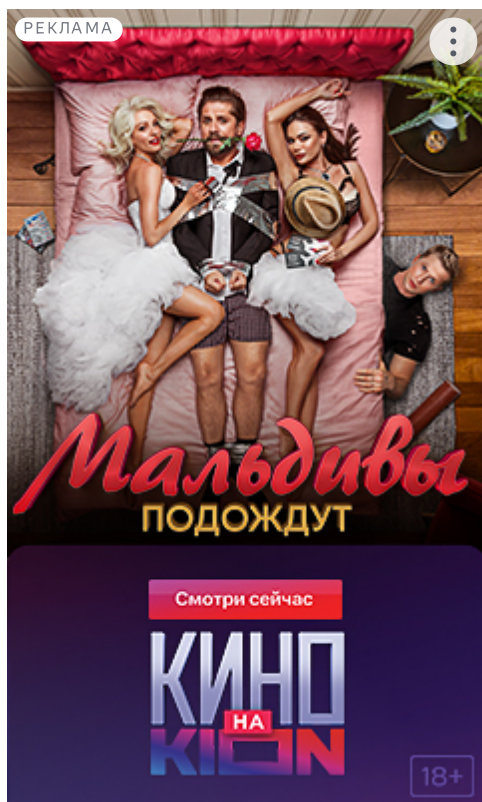
Обратите внимание, что такие поля, как NAME, VERSION и VARIANT, представляют собой строки, подходящие для представления пользователям. Программы должны использовать такие поля, как ID, ID_LIKE, VERSION_ID или VARIANT_ID для идентификации дистрибутивов Linux.

Пример:

```
def get_like_distro():
    info = platform.freedesktop_os_release()
    ids = [info["ID"]]
    if "ID_LIKE" in info:
        # ids are space separated and ordered by precedence
        ids.extend(info["ID_LIKE"].split())
    return ids
```

Новое в Python 3.10.

ХОЧУ ПОМОЧЬ
ПРОЕКТУ



[DOCS-Python.ru](https://docs-python.ru)™, 2023 г.

(Внимание! При копировании материала ссылка на источник обязательна)

[@docs_python_ru](https://t.me/docs_python_ru)