

Common Python Libraries and Modules for Security



Michael Woolard

Risk and Compliance Manager

@wooly6bear wooly6bear.wordpress.com



Sys



docs.python.org/3/library/sys.html

Python » English » 3.10.4 » 3.10.4 Documentation » The Python Standard Library » Python

Previous topic
Python Runtime Services

Next topic
sysconfig — Provide access to Python's configuration information

This Page
Report a Bug
Show Source

sys — System-specific parameters

This module provides access to some variables used or maintained by the interpreter. It is always available.

sys.abiflags
On POSIX systems where Python was built with the standard configuration as specified by [PEP 3149](#).

Changed in version 3.8: Default flags became an empty string (in file).

New in version 3.2.

sys.addaudithook(hook)
Append the callable *hook* to the list of active auditing hooks for the interpreter.

When an auditing event is raised through the `sys.audit()` function, the hooks are called in the order they were added. Native hooks are called first, followed by hooks added in the current (sub)interpreter. A hook can raise an exception to abort the operation, or terminate the process entirely.

Calling `sys.addaudithook()` will itself raise an auditing event named `sys.addaudithook` with the hook as the first argument. If any existing hooks raise an exception derived from `RuntimeError`, the hook is not added and the exception suppressed. As a result, callers cannot assume that a hook has been added unless they control all existing hooks.

See the [audit events table](#) for all events raised by CPython, and [PEP 3113](#) for details.

New in version 3.8.

Changed in version 3.8.1: Exceptions derived from `Exception` but not `RuntimeError` are not suppressed.

CPython implementation detail: When tracing is enabled (see [sys.settrace\(\)](#)), the hook is called if the callable has a `__cantrace__` member that is set to a true value.

sys.argv
The list of command line arguments passed to a Python script. `argv[0]` is the script name (system dependent whether this is a full pathname or not). If the command line starts with `-`, `argv` is a list of the arguments.

Sys

<https://docs.python.org/3/library/sys.html>

Standard module of python

When you want to interact with the interpreter



Sys



.argv

.path

.stderr

.version (*Version_info*)



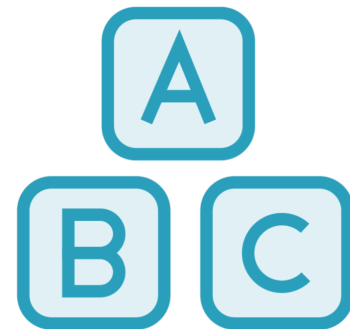
OS



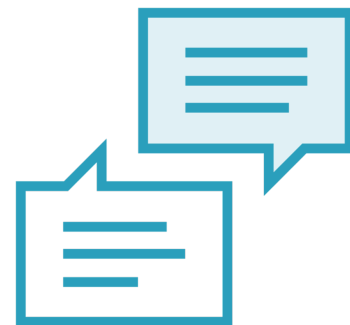
OS



<https://docs.python.org/3/library/os.html>



Standard module of python



**When you want to interact with the
Operating System**



OS

.open

.close

.remove

.getcwd

.getpid

.system *(subprocess)*



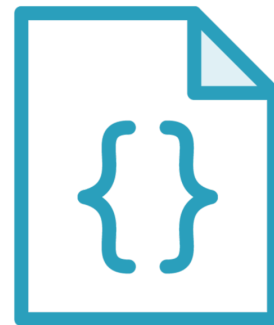
Re



Re



<https://docs.python.org/3/library/re.html>



Pattern Recognition for Searching and Manipulating Strings



Indicator of Compromise

Log Check





`\A`

Matches only at the start of the string.

`\b`

Matches the empty string, but only at the beginning or end of a word. A word is defined as a sequence of word characters. Note that formally, `\b` is defined as the boundary between a `\w` and a `\W` character (or vice versa), or between `\w` and the beginning/end of the string. This means that `r'\bfoo\b'` matches `'foo'`, `'foo.'`, `'(foo)'`, `'bar foo baz'` but not `'foobar'` or `'foo3'`.

By default Unicode alphanumerics are the ones used in Unicode patterns, but this can be changed by using the `ASCII` flag. Word boundaries are determined by the current locale if the `LOCALE` flag is used. Inside a character range, `\b` represents the backspace character, for compatibility with Python's string literals.

`\B`

Matches the empty string, but only when it is *not* at the beginning or end of a word. This means that `r'py\B'` matches `'python'`, `'py3'`, `'py2'`, but not `'py'`, `'py.'`, or `'py!'`. `\B` is just the opposite of `\b`, so word characters in Unicode patterns are Unicode alphanumerics or the underscore, although this can be changed by using the `ASCII` flag. Word boundaries are determined by the current locale if the `LOCALE` flag is used.

`\d`

For Unicode (str) patterns:

Matches any Unicode decimal digit (that is, any character in Unicode character category `[Nd]`). This includes `[0-9]`, and also many other digit characters. If the `ASCII` flag is used only `[0-9]` is matched.

For 8-bit (bytes) patterns:

Matches any decimal digit; this is equivalent to `[0-9]`.

`\D`

Matches any character which is not a decimal digit. This is the opposite of `\d`. If the `ASCII` flag is used this becomes the equivalent of `[^0-9]`.

`\s`

For Unicode (str) patterns:

Matches Unicode whitespace characters (which includes `[\t\n\r\f\v]`, and also many other characters, for example the non-breaking spaces mandated by typography rules in many languages). If the `ASCII` flag is used, only `[\t\n\r\f\v]` is matched.

For 8-bit (bytes) patterns:

Matches characters considered whitespace in the ASCII character set; this is equivalent to `[\t\n\r\f\v]`.

Regex Format

String: **r“The tool costs \$3000”**

\d\d* : Will return ['3000']

\d\d? : Will return ['30','00']

\d\d+ : Will return ['3000']



Regex Format

String: **r“The tool costs \$3”**

\d\d* : Will return ['3']

\d\d? : Will return ['3']

\d\d+ : Will return []



Re

Regular Expression Syntax

Ordinary Characters

Metacharacters

Special Sequences

.compile()

.search()

.match()

.findall()



PSUtil



5.1 CPU

`psutil.cpu_times(percpu=False)`

Return system CPU times as a named tuple. Every attribute represents the seconds the CPU has spent in the given mode. The attributes availability varies depending on the platform:

- **user**: time spent by normal processes executing in user mode; on Linux this also includes **guest** time
- **system**: time spent by processes executing in kernel mode
- **idle**: time spent doing nothing

Platform-specific fields:

- **nice** (*UNIX*): time spent by niced (prioritized) processes executing in user mode; on Linux this also includes **guest_nice** time
- **iowait** (*Linux*): time spent waiting for I/O to complete. This is *not* accounted in **idle** time counter.
- **irq** (*Linux, BSD*): time spent for servicing hardware interrupts
- **softirq** (*Linux*): time spent for servicing software interrupts
- **steal** (*Linux 2.6.11+*): time spent by other operating systems running in a virtualized environment
- **guest** (*Linux 2.6.24+*): time spent running a virtual CPU for guest operating systems under the control of the Linux kernel
- **guest_nice** (*Linux 3.2.0+*): time spent running a niced guest (virtual CPU for guest operating systems under the control of the Linux kernel)
- **interrupt** (*Windows*): time spent for servicing hardware interrupts (similar to “irq” on UNIX)
- **dpc** (*Windows*): time spent servicing deferred procedure calls (DPCs); DPCs are interrupts that run at a lower priority than standard interrupts.

PSUtil

.net_connections

.process

.oneshot

.cpu_...

.disk_...

.virtual_memory...



Cryptography



Cryptography



Provides cryptographic recipes (symmetric encryption) and primitives



<https://cryptography.io/en/latest/>

<https://pypi.org/project/cryptography>



pip install cryptography



Cryptography



fernet

`.generate_key()`

`.Fernet()`

`.encrypt`

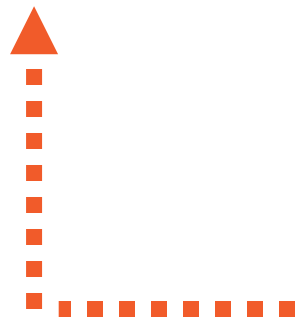
`.decrypt`



Fernet Class

fernet

Fernet(KEY)



Fernet.generate_key()

Encrypt()

Decrypt()



Cryptography



script

Script()

.derive

.verify



Fernet Class

script

Script(salt, length, n, r, p)

├── Derive()

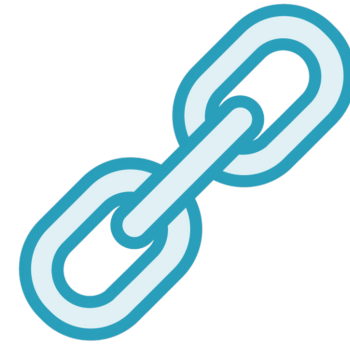
└── Verify()



YARA



YARA



<https://pypi.org/project/yara/>



<https://yara.readthedocs.io/en/stable/yarapython.html>



`pip install yara`



YARA Rule Template

Rule (rule name)

```
{  
  meta:  
    created = ""  
    modified = ""  
    author = ""  
    vendor = ""  
  
  strings:  
    $variable = ""  
  
  condition:  
    (condition to be met to kick off rule)  
  
}
```

rule ExampleRule

```
{  
  meta:  
    author = "Cylance Spear Team"  
    description = "An example Yara rule"  
    threat_level = 10  
    in_the_wild = true  
  strings:  
    $a = "Google Sym"   
    $b = "\\pipe\\1[12345678]"  
    $c = {66 0F FC C1 0F 11 40 D0 0F 10 40 D0  
        66 0F EF C2 0F 11 40 D0 0F 10 40 E0}  
    $a = {4A 10 40 0A 42 AD 80 4B 00 10}  
    $b = {6A B0 99 59 2B C0 F7 16 0A 00 24 AA C7 D9}  
    $c = "HISDBCIBUNSDCLLJSXQZAKCBGMT"  
  condition:  
    $a or $b or $c  
}
```



YARA



.compile

.match



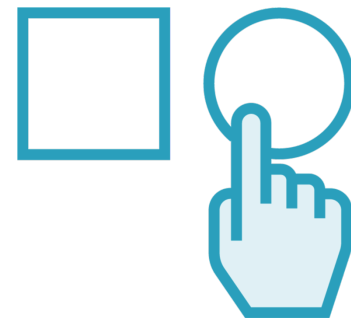
Socket



Socket



<https://docs.python.org/3/library/socket.html#functions>



Communicate back to central dashboard
Penetration Test



Socket

.socket()

.bind()

.listen()

.accept()

.connect()

.send() / sendall()

.recv()

.close()



Scapy



Scapy

<https://scapy.net/>

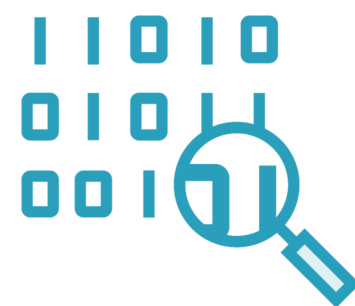
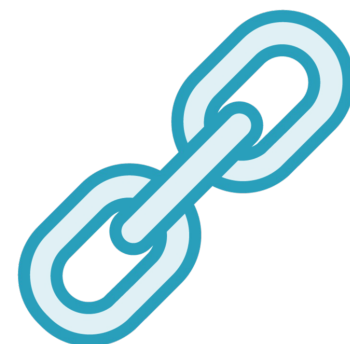
<https://pypi.org/project/scapy/>

<https://scapy.readthedocs.io/en/latest/installation.html>

<https://github.com/secdev/scapy>

Network Packets Crafting / Sniffing

pip install scapy



Scapy



conf()

sr() / sr1()

send(IP/TCP/DNS)

sniff()

show()



Requests



Requests



[**https://pypi.org/project/requests/**](https://pypi.org/project/requests/)



[**https://app.pluralsight.com/guides/web-scraping-with-request-python**](https://app.pluralsight.com/guides/web-scraping-with-request-python)

[**https://app.pluralsight.com/guides/implementing-web-scraping-with-requests**](https://app.pluralsight.com/guides/implementing-web-scraping-with-requests)



pip install requests



Requests



.get()

.post()

.status_code()

.text() / .content()

.header()



Beautifulsoup4



Beautifulsoup4



[**https://pypi.org/project/beautifulsoup4/**](https://pypi.org/project/beautifulsoup4/)



[**https://www.crummy.com/software/BeautifulSoup/bs4/doc/**](https://www.crummy.com/software/BeautifulSoup/bs4/doc/)



pip install beautifulsoup4



Beautifulsoup4



Beautifulsoup(web, parser)

- Can provide local html file
- Requests.get to pull in external

Parsers

- *html.parser*
- *lxml*
- *lxml-xml*
- *html5lib*



Beautifulsoup4



.find()

.find_all()

.find_parent()

.find_next()

.find_previous()



Course Summary



Sys

OS

PSUtil

Re

Cryptography

YARA

Socket

Scapy

Requests

Beautifulsoup

