
Python Standard Library List Documentation

Release 0.11.1

Jack Maney

Feb 18, 2025

CONTENTS

1	Contents	3
1.1	Installation	3
1.2	Usage	3
1.3	Module inclusion policy	5
2	Indices and tables	7
	Python Module Index	9
	Index	11

This package includes lists of all of the standard libraries for Python 2.6 through 3.13.

Note: If you're on Python 3.10 or newer, you **probably don't need this library**. See [sys.stdlib_module_names](#) and [sys.builtin_module_names](#) for similar functionality.

CONTENTS

1.1 Installation

Most end users should use `pip` to install this package:

```
python -m pip install stdlib-list
```

If for whatever reason you need to install `stdlib-list` from the source repository instead:

```
git clone https://github.com/pypi/stdlib-list
cd stdlib-list
python -m pip install .
```

1.2 Usage

1.2.1 Getting The List of Libraries

`stdlib_list.stdlib_list` returns the list of libraries in `stdlib` for any given version (by default, current python version).

In particular:

```
In [1]: from stdlib_list import stdlib_list

In [2]: libs = stdlib_list("3.4")

In [3]: libs[:6]
Out[3]: ['__future__', '__main__', '_dummy_thread', '_thread', 'abc', 'aifc']
```

1.2.2 Checking if a Module is part of stdlib

`stdlib_list.in_stdlib` provides an efficient way to check if a module name is part of stdlib. It relies on `@lru_cache` to cache the stdlib list and query results for similar calls. Therefore it is much more efficient than `module_name in stdlib_list()` especially if you wish to perform multiple checks.

In particular:

```
>>> from stdlib_list import in_stdlib
>>> in_stdlib('zipimport') # built in
True
>>> in_stdlib('math')      # C-API stdlib module, but linked as extension (on my_
↳ machine)
True
>>> in_stdlib('numpy')     # C-API extension, not stdlib
False
>>> in_stdlib('sys')       # built-in (and special)
True
>>> in_stdlib('os')        # Python code in stdlib
True
>>> in_stdlib('requests')  # Python code, not stdlib
False
```

`stdlib_list.in_stdlib(module_name: str, version: str | None = None) → bool`

Return a bool indicating if module `module_name` is in the list of stdlib symbols for python version `version`. If `version` is `None` (default), the version of current python interpreter is used.

Note that `True` will be returned for built-in modules too, since this project considers they are part of stdlib. See [issue:21](#).

It relies on `@lru_cache` to cache the stdlib list and query results for similar calls. Therefore it is much more efficient than `module_name in stdlib_list()` especially if you wish to perform multiple checks.

Parameters

- **module_name** (*str*/*None*) – The module name (as a string) to query for.
- **version** (*str*/*None*) – The version (as a string) whose list of libraries you want (formatted as `X.Y`, e.g. `"2.7"` or `"3.10"`).

If not specified, the current version of Python will be used.

Returns

A bool indicating if the given module name is part of standard libraries for the specified version of Python.

Return type

list

`stdlib_list.stdlib_list(version: str | None = None) → list[str]`

Given a version, return a list of names of the Python Standard Libraries for that version.

Parameters

version (*str*/*None*) – The version (as a string) whose list of libraries you want (formatted as `X.Y`, e.g. `"2.7"` or `"3.10"`).

If not specified, the current version of Python will be used.

Returns

A list of standard libraries from the specified version of Python

Return type

list

1.3 Module inclusion policy

Python is a dynamic language with a complex module system, including modules that are created only at runtime or appear on specific supported platforms.

This page exists to document `stdlib-list`'s approach to module detection and subsequent inclusion. It is not intended to be permanent, and may change over time as Python itself changes (or our approach to module detection improves).

1.3.1 Current guiding rules

- Missing top-level modules **are a bug**: if a new version of Python adds a new top-level module, our failure to detect it should be considered a bug.

Concretely: if `examplemodule` is present in Python 3.999, then it should be included in the `stdlib_list("3.999")` listing.

- Missing sub-modules are **best-effort**: if `examplemodule` contains `examplemodule.foo.bar.baz.deeply.nested`, we make a best-effort attempt to detect each inner module but make no guarantee about doing so.

Our rationale for this is that “stdlib-ness” is inherited from the parent module, even when not explicitly listed. In other words: anything that matches `examplemodule.*` is in the standard library by definition so long as `examplemodule` is in the standard library.

- Platform-specific modules are **best-effort**: `stdlib-list` is currently collected from Linux builds of CPython. This means that Windows- and macOS-specific modules (i.e., modules that aren't installed except for on those hosts) are not necessarily included.

This includes top-level modules.

- Missing non-CPython modules are **not supported**: `stdlib-list` is implicitly a list of CPython's standard library modules, which are expected to be mirrored in other implementations of Python.
- Psuedo-modules are **not supported**: Python sometimes makes use of “pesudo-modules”, i.e. namespaces placed into `sys.modules` that don't pass `inspect.ismodule()`. We don't currently support these, since the semantics for doing so are unclear. See [stdlib-list#117](#) for additional details.

If you have a scenario not covered by the rules above, please file an issue!

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

S

`stdlib_list`, 4

INDEX

I

`in_stdlib()` (*in module `stdlib_list`*), 4

M

module
 `stdlib_list`, 4

S

`stdlib_list`
 module, 4
`stdlib_list()` (*in module `stdlib_list`*), 4