

---

# **Python Standard Library List Documentation**

***Release 0.9.0***

**Jack Maney**

**Jun 29, 2023**



# CONTENTS

<b>1</b>	<b>Contents</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Usage . . . . .	3
<b>2</b>	<b>Indices and tables</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



This package includes lists of all of the standard libraries for Python 2.6 through 3.11.

---

**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



## 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