

`inspect.CO_GENERATOR`

The flag is set when the code object is a generator function, i.e. a generator object is returned when the code object is executed.

`inspect.CO_COROUTINE`

The flag is set when the code object is a coroutine function. When the code object is executed it returns a coroutine object. See [PEP 492](#) for more details.

New in version 3.5.

`inspect.CO_ITERABLE_COROUTINE`

The flag is used to transform generators into generator-based coroutines. Generator objects with this flag can be used in `await` expression, and can `yield` from coroutine objects. See [PEP 492](#) for more details.

New in version 3.5.

`inspect.CO_ASYNC_GENERATOR`

The flag is set when the code object is an asynchronous generator function. When the code object is executed it returns an asynchronous generator object. See [PEP 525](#) for more details.

New in version 3.6.

Note: The flags are specific to CPython, and may not be defined in other Python implementations. Furthermore, the flags are an implementation detail, and can be removed or deprecated in future Python releases. It's recommended to use public APIs from the `inspect` module for any introspection needs.

29.13.9 Command Line Interface

The `inspect` module also provides a basic introspection capability from the command line.

By default, accepts the name of a module and prints the source of that module. A class or function within the module can be printed instead by appended a colon and the qualified name of the target object.

--details

Print information about the specified object rather than the source code

29.14 `site` — Site-specific configuration hook

Source code: [Lib/site.py](#)

This module is automatically imported during initialization. The automatic import can be suppressed using the interpreter's `-S` option.

Importing this module will append site-specific paths to the module search path and add a few builtins, unless `-S` was used. In that case, this module can be safely imported with no automatic modifications to the module search path or additions to the builtins. To explicitly trigger the usual site-specific additions, call the `site.main()` function.

Changed in version 3.3: Importing the module used to trigger paths manipulation even when using `-S`.

It starts by constructing up to four directories from a head and a tail part. For the head part, it uses `sys.prefix` and `sys.exec_prefix`; empty heads are skipped. For the tail part, it uses the empty string and then `lib/site-packages` (on Windows) or `lib/pythonX.Y/site-packages` (on Unix and macOS). For each of the distinct head-tail combinations, it sees if it refers to an existing directory, and if so, adds it to `sys.path` and also inspects the newly added path for configuration files.

Changed in version 3.5: Support for the “site-python” directory has been removed.

If a file named “pyvenv.cfg” exists one directory above `sys.executable`, `sys.prefix` and `sys.exec_prefix` are set to that directory and it is also checked for site-packages (`sys.base_prefix` and `sys.base_exec_prefix` will always be the “real” prefixes of the Python installation). If “pyvenv.cfg” (a bootstrap configuration file) contains the key “include-system-site-packages” set to anything other than “true” (case-insensitive), the system-level prefixes will not be searched for site-packages; otherwise they will.

A path configuration file is a file whose name has the form `name.pth` and exists in one of the four directories mentioned above; its contents are additional items (one per line) to be added to `sys.path`. Non-existing items are never added to `sys.path`, and no check is made that the item refers to a directory rather than a file. No item is added to `sys.path` more than once. Blank lines and lines beginning with `#` are skipped. Lines starting with `import` (followed by space or tab) are executed.

Note: An executable line in a `.pth` file is run at every Python startup, regardless of whether a particular module is actually going to be used. Its impact should thus be kept to a minimum. The primary intended purpose of executable lines is to make the corresponding module(s) importable (load 3rd-party import hooks, adjust `PATH` etc). Any other initialization is supposed to be done upon a module’s actual import, if and when it happens. Limiting a code chunk to a single line is a deliberate measure to discourage putting anything more complex here.

For example, suppose `sys.prefix` and `sys.exec_prefix` are set to `/usr/local`. The Python X.Y library is then installed in `/usr/local/lib/pythonX.Y`. Suppose this has a subdirectory `/usr/local/lib/pythonX.Y/site-packages` with three subsubdirectories, `foo`, `bar` and `spam`, and two path configuration files, `foo.pth` and `bar.pth`. Assume `foo.pth` contains the following:

```
# foo package configuration

foo
bar
bletch
```

and `bar.pth` contains:

```
# bar package configuration

bar
```

Then the following version-specific directories are added to `sys.path`, in this order:

```
/usr/local/lib/pythonX.Y/site-packages/bar
/usr/local/lib/pythonX.Y/site-packages/foo
```

Note that `bletch` is omitted because it doesn’t exist; the `bar` directory precedes the `foo` directory because `bar.pth` comes alphabetically before `foo.pth`; and `spam` is omitted because it is not mentioned in either path configuration file.

After these path manipulations, an attempt is made to import a module named `sitecustomize`, which can perform arbitrary site-specific customizations. It is typically created by a system administrator in the site-packages directory. If this import fails with an `ImportError` or its subclass exception, and the exception’s `name` attribute equals to `'sitecustomize'`, it is silently ignored. If Python is started without output streams available, as with `pythonw.exe` on Windows (which is used by default to start IDLE), attempted output from `sitecustomize` is ignored. Any other exception causes a silent and perhaps mysterious failure of the process.

After this, an attempt is made to import a module named `usercustomize`, which can perform arbitrary user-specific customizations, if `ENABLE_USER_SITE` is true. This file is intended to be created in the user site-packages directory (see below), which is part of `sys.path` unless disabled by `-s`. If this import fails with an `ImportError` or its subclass exception, and the exception’s `name` attribute equals to `'usercustomize'`, it is silently ignored.

Note that for some non-Unix systems, `sys.prefix` and `sys.exec_prefix` are empty, and the path manipulations are skipped; however the import of `sitecustomize` and `usercustomize` is still attempted.

29.14.1 Readline configuration

On systems that support `readline`, this module will also import and configure the `rlcompleter` module, if Python is started in interactive mode and without the `-S` option. The default behavior is enable tab-completion and to use `~/.python_history` as the history save file. To disable it, delete (or override) the `sys.__interactivehook__` attribute in your `sitecustomize` or `usercustomize` module or your `PYTHONSTARTUP` file.

Changed in version 3.4: Activation of `rlcompleter` and history was made automatic.

29.14.2 Module contents

`site.PREFIXES`

A list of prefixes for site-packages directories.

`site.ENABLE_USER_SITE`

Flag showing the status of the user site-packages directory. `True` means that it is enabled and was added to `sys.path`. `False` means that it was disabled by user request (with `-s` or `PYTHONNOUSERSITE`). `None` means it was disabled for security reasons (mismatch between user or group id and effective id) or by an administrator.

`site.USER_SITE`

Path to the user site-packages for the running Python. Can be `None` if `getusersitepackages()` hasn't been called yet. Default value is `~/.local/lib/pythonX.Y/site-packages` for UNIX and non-framework macOS builds, `~/Library/Python/X.Y/lib/python/site-packages` for macOS framework builds, and `%APPDATA%\Python\PythonXY\site-packages` on Windows. This directory is a site directory, which means that `.pth` files in it will be processed.

`site.USER_BASE`

Path to the base directory for the user site-packages. Can be `None` if `getuserbase()` hasn't been called yet. Default value is `~/.local` for UNIX and macOS non-framework builds, `~/Library/Python/X.Y` for macOS framework builds, and `%APPDATA%\Python` for Windows. This value is used by Distutils to compute the installation directories for scripts, data files, Python modules, etc. for the user installation scheme. See also `PYTHONUSERBASE`.

`site.main()`

Adds all the standard site-specific directories to the module search path. This function is called automatically when this module is imported, unless the Python interpreter was started with the `-S` flag.

Changed in version 3.3: This function used to be called unconditionally.

`site.addsitedir(sitedir, known_paths=None)`

Add a directory to `sys.path` and process its `.pth` files. Typically used in `sitecustomize` or `usercustomize` (see above).

`site.getsitepackages()`

Return a list containing all global site-packages directories.

New in version 3.2.

`site.getuserbase()`

Return the path of the user base directory, `USER_BASE`. If it is not initialized yet, this function will also set it, respecting `PYTHONUSERBASE`.

New in version 3.2.

`site.getusersitepackages()`

Return the path of the user-specific site-packages directory, `USER_SITE`. If it is not initialized yet, this function will also set it, respecting `USER_BASE`. To determine if the user-specific site-packages was added to `sys.path` `ENABLE_USER_SITE` should be used.

New in version 3.2.

29.14.3 Command Line Interface

The `site` module also provides a way to get the user directories from the command line:

```
$ python3 -m site --user-site
/home/user/.local/lib/python3.3/site-packages
```

If it is called without arguments, it will print the contents of `sys.path` on the standard output, followed by the value of `USER_BASE` and whether the directory exists, then the same thing for `USER_SITE`, and finally the value of `ENABLE_USER_SITE`.

--user-base

Print the path to the user base directory.

--user-site

Print the path to the user site-packages directory.

If both options are given, user base and user site will be printed (always in this order), separated by `os.pathsep`.

If any option is given, the script will exit with one of these values: 0 if the user site-packages directory is enabled, 1 if it was disabled by the user, 2 if it is disabled for security reasons or by an administrator, and a value greater than 2 if there is an error.

See also:

- **PEP 370** – Per user site-packages directory
- *The initialization of the `sys.path` module search path* – The initialization of `sys.path`.