1. Command line and environment

The CPython interpreter scans the command line and the environment for various settings.

CPython implementation detail: Other implementations' command line schemes may differ. See Alternate Implementations for further resources.

1.1. Command line

When invoking Python, you may specify any of these options:

```
python [-bBdEhiIOqsSuvVWx?] [-c command | -m module-name | script | - ] [arg
```

The most common use case is, of course, a simple invocation of a script:

```
python myscript.py
```

1.1.1. Interface options

The interpreter interface resembles that of the UNIX shell, but provides some additional methods of invocation:

- When called with standard input connected to a tty device, it prompts for commands and executes them until an EOF (an end-of-file character, you can produce that with `Ctrl-D` on UNIX or `Ctrl-Z, Enter` on Windows) is read.
- When called with a file name argument or with a file as standard input, it reads and executes a script from that file.
- When called with a directory name argument, it reads and executes an appropriately named script from that directory.
- When called with `-c command`, it executes the Python statement(s) given as *command*. Here *command* may contain multiple statements separated by newlines. Leading whitespace is significant in Python statements!
- When called with `-m module-name`, the given module is located on the Python module path and executed as a script.

In non-interactive mode, the entire input is parsed before it is executed.

An interface option terminates the list of options consumed by the interpreter, all consecutive arguments will end up in `sys.argv` – note that the first element, subscript zero (`sys.argv[0]`), is a string reflecting the program's source.

`-c` `<command>`

Execute the Python code in *command*. *command* can be one or more statements separated by newlines, with significant leading whitespace as in normal module code.

If this option is given, the first element of `sys.argv` will be `"-c"` and the current directory will be added to the start of `sys.path` (allowing modules in that directory to be imported as top level modules).

Raises an auditing event `cpython.run_command` with argument `command`.

-m `<module-name>`

Search `sys.path` for the named module and execute its contents as the `__main__` module.

Since the argument is a *module* name, you must not give a file extension (`.py`). The module name should be a valid absolute Python module name, but the implementation may not always enforce this (e.g. it may allow you to use a name that includes a hyphen).

Package names (including namespace packages) are also permitted. When a package name is supplied instead of a normal module, the interpreter will execute `<pkg>.__main__` as the main module. This behaviour is deliberately similar to the handling of directories and zipfiles that are passed to the interpreter as the script argument.

> **Note:** This option cannot be used with built-in modules and extension modules written in C, since they do not have Python module files. However, it can still be used for precompiled modules, even if the original source file is not available.

If this option is given, the first element of `sys.argv` will be the full path to the module file (while the module file is being located, the first element will be set to `"-m"`). As with the `-c` option, the current directory will be added to the start of `sys.path`.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the current directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

Many standard library modules contain code that is invoked on their execution as a script. An example is the `timeit` module:

```
python -m timeit -s 'setup here' 'benchmarked code here'
python -m timeit -h # for details
```

Raises an auditing event `cpython.run_module` with argument `module-name`.

> **See also:**
>
> `runpy.run_module()`
>
> Equivalent functionality directly available to Python code
>
> **PEP 338** – Executing modules as scripts

Changed in version 3.1: Supply the package name to run a `__main__` submodule.

Changed in version 3.4: namespace packages are also supported

-

Read commands from standard input (`sys.stdin`). If standard input is a terminal, `-i` is implied.

If this option is given, the first element of `sys.argv` will be `"-"` and the current directory will be added to the start of `sys.path`.

Raises an auditing event `cpython.run_stdin` with no arguments.

<script>

Execute the Python code contained in *script*, which must be a filesystem path (absolute or relative) referring to either a Python file, a directory containing a `__main__.py` file, or a zipfile containing a `__main__.py` file.

If this option is given, the first element of `sys.argv` will be the script name as given on the command line.

If the script name refers directly to a Python file, the directory containing that file is added to the start of `sys.path`, and the file is executed as the `__main__` module.

If the script name refers to a directory or zipfile, the script name is added to the start of `sys.path` and the `__main__.py` file in that location is executed as the `__main__` module.

`-I` option can be used to run the script in isolated mode where `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too.

Raises an auditing event `cpython.run_file` with argument `filename`.

See also:

`runpy.run_path()`

Equivalent functionality directly available to Python code

If no interface option is given, `-i` is implied, `sys.argv[0]` is an empty string (`""`) and the current directory will be added to the start of `sys.path`. Also, tab-completion and history editing is automatically enabled, if available on your platform (see Readline configuration).

See also: Invoking the Interpreter

Changed in version 3.4: Automatic enabling of tab-completion and history editing.

1.1.2. Generic options

-?
-h
--help

Print a short description of all command line options.

-V
--version

Print the Python version number and exit. Example output could be:

```
Python 3.8.0b2+
```

When given twice, print more information about the build, like:

```
Python 3.8.0b2+ (3.8:0c076caaa8, Apr 20 2019, 21:55:00)
[GCC 6.2.0 20161005]
```

New in version 3.6: The `-VV` option.

1.1.3. Miscellaneous options

`-b`

Issue a warning when comparing `bytes` or `bytearray` with `str` or `bytes` with `int`. Issue an error when the option is given twice (`-bb`).

Changed in version 3.5: Affects comparisons of `bytes` with `int`.

`-B`

If given, Python won't try to write `.pyc` files on the import of source modules. See also `PYTHONDONTWRITEBYTECODE`.

`--check-hash-based-pycs` `default|always|never`

Control the validation behavior of hash-based `.pyc` files. See Cached bytecode invalidation. When set to `default`, checked and unchecked hash-based bytecode cache files are validated according to their default semantics. When set to `always`, all hash-based `.pyc` files, whether checked or unchecked, are validated against their corresponding source file. When set to `never`, hash-based `.pyc` files are not validated against their corresponding source files.

The semantics of timestamp-based `.pyc` files are unaffected by this option.

`-d`

Turn on parser debugging output (for expert only, depending on compilation options). See also `PYTHONDEBUG`.

`-E`

Ignore all `PYTHON*` environment variables, e.g. `PYTHONPATH` and `PYTHONHOME`, that might be set.

`-i`

When a script is passed as first argument or the `-c` option is used, enter interactive mode after executing the script or the command, even when `sys.stdin` does not appear to be a terminal. The `PYTHONSTARTUP` file is not read.

This can be useful to inspect global variables or a stack trace when a script raises an exception. See also `PYTHONINSPECT`.

`-I`

Run Python in isolated mode. This also implies -E and -s. In isolated mode `sys.path` contains neither the script's directory nor the user's site-packages directory. All `PYTHON*` environment variables are ignored, too. Further restrictions may be imposed to prevent the user from injecting malicious code.

New in version 3.4.

`-O`

Remove assert statements and any code conditional on the value of `__debug__`. Augment the filename for compiled (bytecode) files by adding `.opt-1` before the `.pyc` extension (see **PEP 488**). See also `PYTHONOPTIMIZE`.

Changed in version 3.5: Modify `.pyc` filenames according to **PEP 488**.

-OO

Do `-O` and also discard docstrings. Augment the filename for compiled (bytecode) files by adding `.opt-2` before the `.pyc` extension (see **PEP 488**).

Changed in version 3.5: Modify `.pyc` filenames according to **PEP 488**.

-q

Don't display the copyright and version messages even in interactive mode.

New in version 3.2.

-R

Turn on hash randomization. This option only has an effect if the `PYTHONHASHSEED` environment variable is set to `0`, since hash randomization is enabled by default.

On previous versions of Python, this option turns on hash randomization, so that the `__hash__()` values of str and bytes objects are "salted" with an unpredictable random value. Although they remain constant within an individual Python process, they are not predictable between repeated invocations of Python.

Hash randomization is intended to provide protection against a denial-of-service caused by carefully chosen inputs that exploit the worst case performance of a dict construction, $O(n^2)$ complexity. See http://www.ocert.org/advisories/ocert-2011-003.html for details.

`PYTHONHASHSEED` allows you to set a fixed value for the hash seed secret.

Changed in version 3.7: The option is no longer ignored.

New in version 3.2.3.

-s

Don't add the `user site-packages directory` to `sys.path`.

> **See also:** **PEP 370** – Per user site-packages directory

-S

Disable the import of the module `site` and the site-dependent manipulations of `sys.path` that it entails. Also disable these manipulations if `site` is explicitly imported later (call `site.main()` if you want them to be triggered).

-u

Force the stdout and stderr streams to be unbuffered. This option has no effect on the stdin stream.

See also `PYTHONUNBUFFERED`.

Changed in version 3.7: The text layer of the stdout and stderr streams now is unbuffered.

-v

Print a message each time a module is initialized, showing the place (filename or built-in module) from which it is loaded. When given twice (`-vv`), print a message for each file that is checked for when searching for a module. Also provides information on module cleanup at exit.

Changed in version 3.10: The `site` module reports the site-specific paths and `.pth` files being processed.

See also `PYTHONVERBOSE`.

-W `arg`

Warning control. Python's warning machinery by default prints warning messages to `sys.stderr`.

The simplest settings apply a particular action unconditionally to all warnings emitted by a process (even those that are otherwise ignored by default):

```
-Wdefault  # Warn once per call location
-Werror    # Convert to exceptions
-Walways   # Warn every time
-Wmodule   # Warn once per calling module
-Wonce     # Warn once per Python process
-Wignore   # Never warn
```

The action names can be abbreviated as desired and the interpreter will resolve them to the appropriate action name. For example, `-Wi` is the same as `-Wignore`.

The full form of argument is:

```
action:message:category:module:lineno
```

Empty fields match all values; trailing empty fields may be omitted. For example `-W ignore::DeprecationWarning` ignores all DeprecationWarning warnings.

The *action* field is as explained above but only applies to warnings that match the remaining fields.

The *message* field must match the whole warning message; this match is case-insensitive.

The *category* field matches the warning category (ex: `DeprecationWarning`). This must be a class name; the match test whether the actual warning category of the message is a subclass of the specified warning category.

The *module* field matches the (fully qualified) module name; this match is case-sensitive.

The *lineno* field matches the line number, where zero matches all line numbers and is thus equivalent to an omitted line number.

Multiple `-W` options can be given; when a warning matches more than one option, the action for the last matching option is performed. Invalid `-W` options are ignored (though, a warning message is printed about invalid options when the first warning is issued).

Warnings can also be controlled using the `PYTHONWARNINGS` environment variable and from within a Python program using the `warnings` module. For example, the `warnings.filterwarnings()` function can be used to use a regular expression on the warning message.

See The Warnings Filter and Describing Warning Filters for more details.

`-x`

Skip the first line of the source, allowing use of non-Unix forms of `#!cmd`. This is intended for a DOS specific hack only.

`-X`

Reserved for various implementation-specific options. CPython currently defines the following possible values:

- `-X faulthandler` to enable `faulthandler`;
- `-X showrefcount` to output the total reference count and number of used memory blocks when the program finishes or after each statement in the interactive interpreter. This only works on debug builds.
- `-X tracemalloc` to start tracing Python memory allocations using the `tracemalloc` module. By default, only the most recent frame is stored in a traceback of a trace. Use `-X tracemalloc=NFRAME` to start tracing with a traceback limit of *NFRAME* frames. See the `tracemalloc.start()` for more information.
- `-X int_max_str_digits` configures the integer string conversion length limitation. See also `PYTHONINTMAXSTRDIGITS`.
- `-X importtime` to show how long each import takes. It shows module name, cumulative time (including nested imports) and self time (excluding nested imports). Note that its output may be broken in multi-threaded application. Typical usage is `python3 -X importtime -c 'import asyncio'`. See also `PYTHONPROFILEIMPORTTIME`.
- `-X dev`: enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default.
- `-X utf8` enables the Python UTF-8 Mode. `-X utf8=0` explicitly disables Python UTF-8 Mode (even when it would otherwise activate automatically).
- `-X pycache_prefix=PATH` enables writing `.pyc` files to a parallel tree rooted at the given directory instead of to the code tree. See also `PYTHONPYCACHEPREFIX`.
- `-X warn_default_encoding` issues a `EncodingWarning` when the locale-specific default encoding is used for opening files. See also `PYTHONWARNDEFAULTENCODING`.

It also allows passing arbitrary values and retrieving them through the `sys._xoptions` dictionary.

Changed in version 3.2: The `-X` option was added.

New in version 3.3: The `-X faulthandler` option.

New in version 3.4: The `-X showrefcount` and `-X tracemalloc` options.

New in version 3.6: The `-X showalloccount` option.

New in version 3.7: The `-X importtime`, `-X dev` and `-X utf8` options.

New in version 3.8: The `-X pycache_prefix` option. The `-X dev` option now logs `close()` exceptions in `io.IOBase` destructor.

Changed in version 3.9: Using `-X dev` option, check *encoding* and *errors* arguments on string encoding and decoding operations.

The `-X showalloccount` option has been removed.

New in version 3.10: The `-X warn_default_encoding` option.

New in version 3.10.7: The `-X int_max_str_digits` option.

Deprecated since version 3.9, removed in version 3.10: The `-X oldparser` option.

1.1.4. Options you shouldn't use

`-J`

Reserved for use by Jython.

1.2. Environment variables

These environment variables influence Python's behavior, they are processed before the command-line switches other than -E or -I. It is customary that command-line switches override environmental variables where there is a conflict.

`PYTHONHOME`

Change the location of the standard Python libraries. By default, the libraries are searched in *prefix*`/lib/python`*version* and *exec_prefix*`/lib/python`*version*, where *prefix* and *exec_prefix* are installation-dependent directories, both defaulting to `/usr/local`.

When `PYTHONHOME` is set to a single directory, its value replaces both *prefix* and *exec_prefix*. To specify different values for these, set `PYTHONHOME` to *prefix*`:`*exec_prefix*.

`PYTHONPATH`

Augment the default search path for module files. The format is the same as the shell's `PATH`: one or more directory pathnames separated by `os.pathsep` (e.g. colons on Unix or semicolons on Windows). Non-existent directories are silently ignored.

In addition to normal directories, individual `PYTHONPATH` entries may refer to zipfiles containing pure Python modules (in either source or compiled form). Extension modules cannot be imported from zipfiles.

The default search path is installation dependent, but generally begins with *prefix*`/lib/python`*version* (see `PYTHONHOME` above). It is *always* appended to `PYTHONPATH`.

An additional directory will be inserted in the search path in front of `PYTHONPATH` as described above under Interface options. The search path can be manipulated from within a Python program as the variable `sys.path`.

`PYTHONPLATLIBDIR`

If this is set to a non-empty string, it overrides the `sys.platlibdir` value.

New in version 3.9.

PYTHONSTARTUP

If this is the name of a readable file, the Python commands in that file are executed before the first prompt is displayed in interactive mode. The file is executed in the same namespace where interactive commands are executed so that objects defined or imported in it can be used without qualification in the interactive session. You can also change the prompts `sys.ps1` and `sys.ps2` and the hook `sys.__interactivehook__` in this file.

Raises an auditing event `cpython.run_startup` with the filename as the argument when called on startup.

PYTHONOPTIMIZE

If this is set to a non-empty string it is equivalent to specifying the `-O` option. If set to an integer, it is equivalent to specifying `-O` multiple times.

PYTHONBREAKPOINT

If this is set, it names a callable using dotted-path notation. The module containing the callable will be imported and then the callable will be run by the default implementation of `sys.breakpointhook()` which itself is called by built-in `breakpoint()`. If not set, or set to the empty string, it is equivalent to the value “pdb.set_trace”. Setting this to the string “0” causes the default implementation of `sys.breakpointhook()` to do nothing but return immediately.

New in version 3.7.

PYTHONDEBUG

If this is set to a non-empty string it is equivalent to specifying the `-d` option. If set to an integer, it is equivalent to specifying `-d` multiple times.

PYTHONINSPECT

If this is set to a non-empty string it is equivalent to specifying the `-i` option.

This variable can also be modified by Python code using `os.environ` to force inspect mode on program termination.

PYTHONUNBUFFERED

If this is set to a non-empty string it is equivalent to specifying the `-u` option.

PYTHONVERBOSE

If this is set to a non-empty string it is equivalent to specifying the `-v` option. If set to an integer, it is equivalent to specifying `-v` multiple times.

PYTHONCASEOK

If this is set, Python ignores case in `import` statements. This only works on Windows and macOS.

PYTHONDONTWRITEBYTECODE

If this is set to a non-empty string, Python won't try to write `.pyc` files on the import of source modules. This is equivalent to specifying the `-B` option.

PYTHONPYCACHEPREFIX

If this is set, Python will write `.pyc` files in a mirror directory tree at this path, instead of in `__pycache__` directories within the source tree. This is equivalent to specifying the `-X pycache_prefix=PATH` option.

New in version 3.8.

PYTHONHASHSEED

If this variable is not set or set to `random`, a random value is used to seed the hashes of str and bytes objects.

If `PYTHONHASHSEED` is set to an integer value, it is used as a fixed seed for generating the hash() of the types covered by the hash randomization.

Its purpose is to allow repeatable hashing, such as for selftests for the interpreter itself, or to allow a cluster of python processes to share hash values.

The integer must be a decimal number in the range [0,4294967295]. Specifying the value 0 will disable hash randomization.

New in version 3.2.3.

PYTHONINTMAXSTRDIGITS

If this variable is set to an integer, it is used to configure the interpreter's global integer string conversion length limitation.

New in version 3.10.7.

PYTHONIOENCODING

If this is set before running the interpreter, it overrides the encoding used for stdin/stdout/stderr, in the syntax `encodingname:errorhandler`. Both the `encodingname` and the `:errorhandler` parts are optional and have the same meaning as in `str.encode()`.

For stderr, the `:errorhandler` part is ignored; the handler will always be `'backslashreplace'`.

Changed in version 3.4: The `encodingname` part is now optional.

Changed in version 3.6: On Windows, the encoding specified by this variable is ignored for interactive console buffers unless `PYTHONLEGACYWINDOWSSTDIO` is also specified. Files and pipes redirected through the standard streams are not affected.

PYTHONNOUSERSITE

If this is set, Python won't add the `user site-packages directory` to `sys.path`.

> **See also:** **PEP 370** – Per user site-packages directory

PYTHONUSERBASE

Defines the `user base directory`, which is used to compute the path of the `user site-packages directory` and Distutils installation paths for `python setup.py install --user`.

> **See also:** **PEP 370** – Per user site-packages directory

PYTHONEXECUTABLE

If this environment variable is set, `sys.argv[0]` will be set to its value instead of the value got through the C runtime. Only works on macOS.

PYTHONWARNINGS

This is equivalent to the `-W` option. If set to a comma separated string, it is equivalent to specifying `-W` multiple times, with filters later in the list taking precedence over those earlier in the list.

The simplest settings apply a particular action unconditionally to all warnings emitted by a process (even those that are otherwise ignored by default):

```
PYTHONWARNINGS=default  # Warn once per call location
PYTHONWARNINGS=error    # Convert to exceptions
PYTHONWARNINGS=always   # Warn every time
PYTHONWARNINGS=module   # Warn once per calling module
PYTHONWARNINGS=once     # Warn once per Python process
PYTHONWARNINGS=ignore   # Never warn
```

See The Warnings Filter and Describing Warning Filters for more details.

PYTHONFAULTHANDLER

If this environment variable is set to a non-empty string, `faulthandler.enable()` is called at startup: install a handler for `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` and `SIGILL` signals to dump the Python traceback. This is equivalent to `-X faulthandler` option.

New in version 3.3.

PYTHONTRACEMALLOC

If this environment variable is set to a non-empty string, start tracing Python memory allocations using the `tracemalloc` module. The value of the variable is the maximum number of frames stored in a traceback of a trace. For example, `PYTHONTRACEMALLOC=1` stores only the most recent frame. See the `tracemalloc.start()` for more information.

New in version 3.4.

PYTHONPROFILEIMPORTTIME

If this environment variable is set to a non-empty string, Python will show how long each import takes. This is exactly equivalent to setting `-X importtime` on the command line.

New in version 3.7.

PYTHONASYNCIODEBUG

If this environment variable is set to a non-empty string, enable the debug mode of the `asyncio` module.

New in version 3.4.

PYTHONMALLOC

Set the Python memory allocators and/or install debug hooks.

Set the family of memory allocators used by Python:

- `default`: use the default memory allocators.
- `malloc`: use the `malloc()` function of the C library for all domains (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: use the pymalloc allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.

Install debug hooks:

- `debug`: install debug hooks on top of the default memory allocators.
- `malloc_debug`: same as `malloc` but also install debug hooks.
- `pymalloc_debug`: same as `pymalloc` but also install debug hooks.

Changed in version 3.7: Added the `"default"` allocator.

New in version 3.6.

PYTHONMALLOCSTATS

If set to a non-empty string, Python will print statistics of the pymalloc memory allocator every time a new pymalloc object arena is created, and on shutdown.

This variable is ignored if the `PYTHONMALLOC` environment variable is used to force the `malloc()` allocator of the C library, or if Python is configured without `pymalloc` support.

Changed in version 3.6: This variable can now also be used on Python compiled in release mode. It now has no effect if set to an empty string.

PYTHONLEGACYWINDOWSFSENCODING

If set to a non-empty string, the default filesystem encoding and error handler mode will revert to their pre-3.6 values of ‘mbcs’ and ‘replace’, respectively. Otherwise, the new defaults ‘utf-8’ and ‘surrogatepass’ are used.

This may also be enabled at runtime with `sys._enablelegacywindowsfsencoding()`.

Availability: Windows.

New in version 3.6: See **PEP 529** for more details.

PYTHONLEGACYWINDOWSSTDIO

If set to a non-empty string, does not use the new console reader and writer. This means that Unicode characters will be encoded according to the active console code page, rather than using utf-8.

This variable is ignored if the standard streams are redirected (to files or pipes) rather than referring to console buffers.

Availability: Windows.

New in version 3.6.

PYTHONCOERCECLOCALE

If set to the value `0`, causes the main Python command line application to skip coercing the legacy ASCII-based C and POSIX locales to a more capable UTF-8 based alternative.

If this variable is *not* set (or is set to a value other than `0`), the `LC_ALL` locale override environment variable is also not set, and the current locale reported for the `LC_CTYPE` category is either the default `C` locale, or else the explicitly ASCII-based `POSIX` locale, then the Python CLI will attempt to configure the following locales for the `LC_CTYPE` category in the order listed before loading the interpreter runtime:

- `C.UTF-8`
- `C.utf8`
- `UTF-8`

If setting one of these locale categories succeeds, then the `LC_CTYPE` environment variable will also be set accordingly in the current process environment before the Python runtime is initialized. This ensures that in addition to being seen by both the interpreter itself and other locale-aware components running in the same process (such as the GNU `readline` library), the updated setting is also seen in subprocesses (regardless of whether or not those processes are running a Python interpreter), as well as in operations that query the environment rather than the current C locale (such as Python's own `locale.getdefaultlocale()`).

Configuring one of these locales (either explicitly or via the above implicit locale coercion) automatically enables the `surrogateescape` error handler for `sys.stdin` and `sys.stdout` (`sys.stderr` continues to use `backslashreplace` as it does in any other locale). This stream handling behavior can be overridden using `PYTHONIOENCODING` as usual.

For debugging purposes, setting `PYTHONCOERCECLOCALE=warn` will cause Python to emit warning messages on `stderr` if either the locale coercion activates, or else if a locale that *would* have triggered coercion is still active when the Python runtime is initialized.

Also note that even when locale coercion is disabled, or when it fails to find a suitable target locale, `PYTHONUTF8` will still activate by default in legacy ASCII-based locales. Both features must be disabled in order to force the interpreter to use `ASCII` instead of `UTF-8` for system interfaces.

Availability: *nix.

New in version 3.7: See **PEP 538** for more details.

PYTHONDEVMODE

If this environment variable is set to a non-empty string, enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default.

New in version 3.7.

PYTHONUTF8

If set to `1`, enable the Python UTF-8 Mode.

If set to `0`, disable the Python UTF-8 Mode.

Setting any other non-empty string causes an error during interpreter initialisation.

New in version 3.7.

PYTHONWARNDEFAULTENCODING

If this environment variable is set to a non-empty string, issue a `EncodingWarning` when the locale-specific default encoding is used.

See Opt-in EncodingWarning for details.

New in version 3.10.

1.2.1. Debug-mode variables

PYTHONTHREADDEBUG

If set, Python will print threading debug info into stdout.

Need a debug build of Python.

Deprecated since version 3.10, will be removed in version 3.12.

PYTHONDUMPREFS

If set, Python will dump objects and reference counts still alive after shutting down the interpreter.

Need Python configured with the `--with-trace-refs` build option.