Python Setup and Usage

Versión 3.7.17

Guido van Rossum
and the Python development team

junio 28, 2023

Python Software Foundation
Email: docs@python.org

Índice general

This part of the documentation is devoted to general information on the setup of the Python environment on different platforms, the invocation of the interpreter and things that make working with Python easier.

CAPÍTULO 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 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 | - ] [args]
```

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).

-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.

Nota: 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
```

Ver también:

`runpy.run_module()` Equivalent functionality directly available to Python code

PEP 338 – Executing modules as scripts

Distinto en la versión 3.1: Supply the package name to run a `__main__` submodule.

Distinto en la versión 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`.

<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.

Ver también:

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 rlcompleter-config).

Ver también:

tut-invoking

Distinto en la versión 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 3.7.0b2+
```

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

```
Python 3.7.0b2+ (3.7:0c076caaa8, Sep 22 2018, 12:04:24)
[GCC 6.2.0 20161005]
```

Nuevo en la versión 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`).

Distinto en la versión 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 pyc-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.

Nuevo en la versión 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*.

Distinto en la versión 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**).

Distinto en la versión 3.5: Modify `.pyc` filenames according to **PEP 488**.

-q

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

Nuevo en la versión 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, bytes and datetime 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.

Distinto en la versión 3.7: The option is no longer ignored.

Nuevo en la versión 3.2.3.

-s

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

Ver también:

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*.

Distinto en la versión 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. See also *PYTHONVERBOSE*.

-W `arg`

Warning control. Python's warning machinery by default prints warning messages to `sys.stderr`. A typical warning message has the following form:

```
file:line: category: message
```

By default, each warning is printed once for each source line where it occurs. This option controls how often warnings are printed.

Multiple *-W* options may 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.

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 (e.g. `-Wi`, `-Wd`, `-Wa`, `-We`) and the interpreter will resolve them to the appropriate action name.

See warning-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 showalloccount` to output the total count of allocated objects for each type when the program finishes. This only works when Python was built with `COUNT_ALLOCS` defined.
- `-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 CPython's «development mode», introducing additional runtime checks which are too expensive to be enabled by default. It should not be more verbose than the default if the code is correct: new warnings are only emitted when an issue is detected. Effect of the developer mode:
  - Add `default` warning filter, as *-W* `default`.
  - Install debug hooks on memory allocators: see the `PyMem_SetupDebugHooks()` C function.
  - Enable the `faulthandler` module to dump the Python traceback on a crash.
  - Enable asyncio debug mode.
  - Set the `dev_mode` attribute of `sys.flags` to `True`.
- `-X utf8` enables UTF-8 mode for operating system interfaces, overriding the default locale-aware mode. `-X utf8=0` explicitly disables UTF-8 mode (even when it would otherwise activate automatically). See *PYTHONUTF8* for more details.

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

Distinto en la versión 3.2: The *-X* option was added.

Nuevo en la versión 3.3: The `-X faulthandler` option.

Nuevo en la versión 3.4: The `-X showrefcount` and `-X tracemalloc` options.

Nuevo en la versión 3.6: The `-X showalloccount` option.

Nuevo en la versión 3.7: The `-X importtime`, `-X dev` and `-X utf8` options.

Nuevo en la versión 3.7.14: The `-X int_max_str_digits` 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/pythonversion` and `exec_prefix/lib/pythonversion`, 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/pythonversion` (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`.

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.

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.

Nuevo en la versión 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 OS X.

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.

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

Nuevo en la versión 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.

Nuevo en la versión 3.7.14.

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'`.

Distinto en la versión 3.4: The `encodingname` part is now optional.

Distinto en la versión 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`.

Ver también:

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`.

Ver también:

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 Mac OS X.

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 warning-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.

Nuevo en la versión 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.

Nuevo en la versión 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.

Nuevo en la versión 3.7.

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

Nuevo en la versión 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.

See the default memory allocators and the `PyMem_SetupDebugHooks()` function (install debug hooks on Python memory allocators).

Distinto en la versión 3.7: Added the `"default"` allocator.

Nuevo en la versión 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.

Distinto en la versión 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 errors 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.

Nuevo en la versión 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.

Nuevo en la versión 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.

Nuevo en la versión 3.7: See **PEP 538** for more details.

PYTHONDEVMODE

If this environment variable is set to a non-empty string, enable the CPython «development mode». See the *-X* dev option.

Nuevo en la versión 3.7.

PYTHONUTF8

If set to 1, enables the interpreter's UTF-8 mode, where UTF-8 is used as the text encoding for system interfaces, regardless of the current locale setting.

This means that:

- sys.getfilesystemencoding() returns 'UTF-8' (the locale encoding is ignored).
- locale.getpreferredencoding() returns 'UTF-8' (the locale encoding is ignored, and the function's do_setlocale parameter has no effect).
- sys.stdin, sys.stdout, and sys.stderr all use UTF-8 as their text encoding, with the surrogateescape error handler being enabled for sys.stdin and sys.stdout (sys.stderr continues to use backslashreplace as it does in the default locale-aware mode)

As a consequence of the changes in those lower level APIs, other higher level APIs also exhibit different default behaviours:

- Command line arguments, environment variables and filenames are decoded to text using the UTF-8 encoding.
- os.fsdecode() and os.fsencode() use the UTF-8 encoding.
- open(), io.open(), and codecs.open() use the UTF-8 encoding by default. However, they still use the strict error handler by default so that attempting to open a binary file in text mode is likely to raise an exception rather than producing nonsense data.

Note that the standard stream settings in UTF-8 mode can be overridden by *PYTHONIOENCODING* (just as they can be in the default locale-aware mode).

If set to `0`, the interpreter runs in its default locale-aware mode.

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

If this environment variable is not set at all, then the interpreter defaults to using the current locale settings, *unless* the current locale is identified as a legacy ASCII-based locale (as described for *PYTHONCOERCECLOCALE*), and locale coercion is either disabled or fails. In such legacy locales, the interpreter will default to enabling UTF-8 mode unless explicitly instructed not to do so.

Also available as the *-X* `utf8` option.

Nuevo en la versión 3.7: See **PEP 540** for more details.

1.2.1 Debug-mode variables

Setting these variables only has an effect in a debug build of Python, that is, if Python was configured with the `--with-pydebug` build option.

PYTHONTHREADDEBUG
If set, Python will print threading debug info.

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

CAPÍTULO 2

Using Python on Unix platforms

2.1 Getting and installing the latest version of Python

2.1.1 On Linux

Python comes preinstalled on most Linux distributions, and is available as a package on all others. However there are certain features you might want to use that are not available on your distro's package. You can easily compile the latest version of Python from source.

In the event that Python doesn't come preinstalled and isn't in the repositories as well, you can easily make packages for your own distro. Have a look at the following links:

Ver también:

https://www.debian.org/doc/manuals/maint-guide/first.en.html for Debian users

https://en.opensuse.org/Portal:Packaging for OpenSuse users

https://docs-old.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html for Fedora users

http://www.slackbook.org/html/package-management-making-packages.html for Slackware users

2.1.2 On FreeBSD and OpenBSD

- FreeBSD users, to add the package use:

```
pkg install python3
```

- OpenBSD users, to add the package use:

```
pkg_add -r python

pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/<insert your␣
↪architecture here>/python-<version>.tgz
```

For example i386 users get the 2.5.1 version of Python using:

```
pkg_add ftp://ftp.openbsd.org/pub/OpenBSD/4.2/packages/i386/python-2.5.1p2.tgz
```

2.1.3 On OpenSolaris

You can get Python from OpenCSW. Various versions of Python are available and can be installed with e.g. `pkgutil -i python27`.

2.2 Building Python

If you want to compile CPython yourself, first thing you should do is get the source. You can download either the latest release's source or just grab a fresh clone. (If you want to contribute patches, you will need a clone.)

The build process consists in the usual

```
./configure
make
make install
```

invocations. Configuration options and caveats for specific Unix platforms are extensively documented in the README.rst file in the root of the Python source tree.

Advertencia: `make install` can overwrite or masquerade the `python3` binary. `make altinstall` is therefore recommended instead of `make install` since it only installs *exec_prefix*/bin/python*version*.

2.3 Python-related paths and files

These are subject to difference depending on local installation conventions; `prefix` (`${prefix}`) and `exec_prefix` (`${exec_prefix}`) are installation-dependent and should be interpreted as for GNU software; they may be the same.

For example, on most Linux systems, the default for both is `/usr`.

File/directory	Meaning
exec_prefix/bin/python3	Recommended location of the interpreter.
prefix/lib/python*version*, *exec_prefix*/lib/python*version*	Recommended locations of the directories containing the standard modules.
prefix/include/python*version*, *exec_prefix*/include/python*version*	Recommended locations of the directories containing the include files needed for developing Python extensions and embedding the interpreter.

2.4 Miscellaneous

To easily use Python scripts on Unix, you need to make them executable, e.g. with

```
$ chmod +x script
```

and put an appropriate Shebang line at the top of the script. A good choice is usually

```
#!/usr/bin/env python3
```

which searches for the Python interpreter in the whole `PATH`. However, some Unices may not have the **`env`** command, so you may need to hardcode `/usr/bin/python3` as the interpreter path.

To use shell commands in your Python scripts, look at the `subprocess` module.

CAPÍTULO 3

Using Python on Windows

This document aims to give an overview of Windows-specific behaviour you should know about when using Python on Microsoft Windows.

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. To make Python available, the CPython team has compiled Windows installers (MSI packages) with every release for many years. These installers are primarily intended to add a per-user installation of Python, with the core interpreter and library being used by a single user. The installer is also able to install for all users of a single machine, and a separate ZIP file is available for application-local distributions.

As specified in **PEP 11**, a Python release only supports a Windows platform while Microsoft considers the platform under extended support. This means that Python 3.7 supports Windows Vista and newer. If you require Windows XP support then please install Python 3.4.

There are a number of different installers available for Windows, each with certain benefits and downsides.

The full installer contains all components and is the best option for developers using Python for any kind of project.

The Microsoft Store package is a simple installation of Python that is suitable for running scripts and packages, and using IDLE or other development environments. It requires Windows 10, but can be safely installed without corrupting other programs. It also provides many convenient commands for launching Python and its tools.

The nuget.org packages are lightweight installations intended for continuous integration systems. It can be used to build Python packages or run scripts, but is not updateable and has no user interface tools.

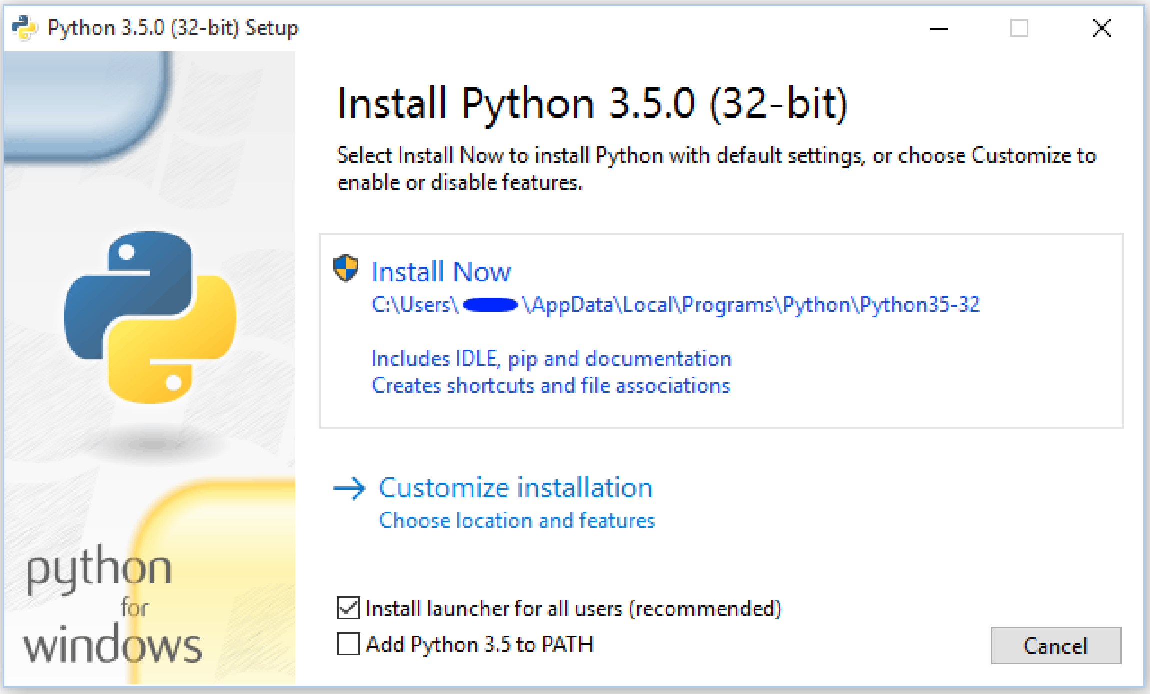
The embeddable package is a minimal package of Python suitable for embedding into a larger application.

3.1 The full installer

3.1.1 Installation steps

Four Python 3.7 installers are available for download - two each for the 32-bit and 64-bit versions of the interpreter. The *web installer* is a small initial download, and it will automatically download the required components as necessary. The *offline installer* includes the components necessary for a default installation and only requires an internet connection for optional features. See *Installing Without Downloading* for other ways to avoid downloading during installation.

After starting the installer, one of two options may be selected:



If you select «Install Now»:

- You will *not* need to be an administrator (unless a system update for the C Runtime Library is required or you install the *Python Launcher for Windows* for all users)
- Python will be installed into your user directory
- The *Python Launcher for Windows* will be installed according to the option at the bottom of the first page
- The standard library, test suite, launcher and pip will be installed
- If selected, the install directory will be added to your `PATH`
- Shortcuts will only be visible for the current user

Selecting «Customize installation» will allow you to select the features to install, the installation location and other options or post-install actions. To install debugging symbols or binaries, you will need to use this option.

To perform an all-users installation, you should select «Customize installation». In this case:

- You may be required to provide administrative credentials or approval
- Python will be installed into the Program Files directory
- The *Python Launcher for Windows* will be installed into the Windows directory
- Optional features may be selected during installation
- The standard library can be pre-compiled to bytecode
- If selected, the install directory will be added to the system `PATH`
- Shortcuts are available for all users

3.1.2 Removing the MAX_PATH Limitation

Windows historically has limited path lengths to 260 characters. This meant that paths longer than this would not resolve and errors would result.

In the latest versions of Windows, this limitation can be expanded to approximately 32,000 characters. Your administrator will need to activate the «Enable Win32 long paths» group policy, or set the registry value `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem@LongPathsEnabled` to `1`.

This allows the `open()` function, the `os` module and most other path functionality to accept and return paths longer than 260 characters.

After changing the above option, no further configuration is required.

Distinto en la versión 3.6: Support for long paths was enabled in Python.

3.1.3 Installing Without UI

All of the options available in the installer UI can also be specified from the command line, allowing scripted installers to replicate an installation on many machines without user interaction. These options may also be set without suppressing the UI in order to change some of the defaults.

To completely hide the installer UI and install Python silently, pass the `/quiet` option. To skip past the user interaction but still display progress and errors, pass the `/passive` option. The `/uninstall` option may be passed to immediately begin removing Python - no prompt will be displayed.

All other options are passed as `name=value`, where the value is usually `0` to disable a feature, `1` to enable a feature, or a path. The full list of available options is shown below.

Name	Description	Default
InstallAllUsers	Perform a system-wide installation.	0
TargetDir	The installation directory	Selected based on InstallAllUsers
DefaultAllUsersTargetDir	The default installation directory for all-user installs	`%ProgramFiles%\Python X.Y` or `%ProgramFiles(x86)%\Python X.Y`
DefaultJustForMeTargetDir	The default install directory for just-for-me installs	`%LocalAppData%\Programs\PythonXY` or `%LocalAppData%\Programs\PythonXY-32` or `%LocalAppData%\Programs\PythonXY-64`
DefaultCustomTargetDir	The default custom install directory displayed in the UI	(empty)
AssociateFiles	Create file associations if the launcher is also installed.	1
CompileAll	Compile all `.py` files to `.pyc`.	0
PrependPath	Add install and Scripts directories to `PATH` and `.PY` to `PATHEXT`	0
Shortcuts	Create shortcuts for the interpreter, documentation and IDLE if installed.	1
Include_doc	Install Python manual	1
Include_debug	Install debug binaries	0
Include_dev	Install developer headers and libraries	1
Include_exe	Install `python.exe` and related files	1
Include_launcher	Install *Python Launcher for Windows*.	1
InstallLauncherAllUsers	Installs *Python Launcher for Windows* for all users.	1
Include_lib	Install standard library and extension modules	1
Include_pip	Install bundled pip and setuptools	1
Include_symbols	Install debugging symbols (*.pdb)	0
Include_tcltk	Install Tcl/Tk support and IDLE	1
Include_test	Install standard library test suite	1
Include_tools	Install utility scripts	1
LauncherOnly	Only installs the launcher. This will override most other options.	0
SimpleInstall	Disable most install UI	0
SimpleInstallDescription	A custom message to display when the simplified install UI is used.	(empty)

For example, to silently install a default, system-wide Python installation, you could use the following command (from an elevated command prompt):

```
python-3.7.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

To allow users to easily install a personal copy of Python without the test suite, you could provide a shortcut with the following command. This will display a simplified initial page and disallow customization:

```
python-3.7.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
    SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(Note that omitting the launcher also omits file associations, and is only recommended for per-user installs when there is also a system-wide installation that included the launcher.)

The options listed above can also be provided in a file named `unattend.xml` alongside the executable. This file specifies a list of options and values. When a value is provided as an attribute, it will be converted to a number if possible. Values provided as element text are always left as strings. This example file sets the same options as the previous example:

```
<Options>
    <Option Name="InstallAllUsers" Value="no" />
    <Option Name="Include_launcher" Value="0" />
    <Option Name="Include_test" Value="no" />
    <Option Name="SimpleInstall" Value="yes" />
    <Option Name="SimpleInstallDescription">Just for me, no test suite</Option>
</Options>
```

3.1.4 Installing Without Downloading

As some features of Python are not included in the initial installer download, selecting those features may require an internet connection. To avoid this need, all possible components may be downloaded on-demand to create a complete *layout* that will no longer require an internet connection regardless of the selected features. Note that this download may be bigger than required, but where a large number of installations are going to be performed it is very useful to have a locally cached copy.

Execute the following command from Command Prompt to download all possible required files. Remember to substitute `python-3.7.0.exe` for the actual name of your installer, and to create layouts in their own directories to avoid collisions between files with the same name.

```
python-3.7.0.exe /layout [optional target directory]
```

You may also specify the `/quiet` option to hide the progress display.

3.1.5 Modifying an install

Once Python has been installed, you can add or remove features through the Programs and Features tool that is part of Windows. Select the Python entry and choose «Uninstall/Change» to open the installer in maintenance mode.

«Modify» allows you to add or remove features by modifying the checkboxes - unchanged checkboxes will not install or remove anything. Some options cannot be changed in this mode, such as the install directory; to modify these, you will need to remove and then reinstall Python completely.

«Repair» will verify all the files that should be installed using the current settings and replace any that have been removed or modified.

«Uninstall» will remove Python entirely, with the exception of the *Python Launcher for Windows*, which has its own entry in Programs and Features.

3.2 The Microsoft Store package

Nuevo en la versión 3.7.2.

Nota: The Microsoft Store package is currently considered unstable while its interactions with other tools and other copies of Python are evaluated. While Python itself is stable, this installation method may change its behavior and capabilities during Python 3.7 releases.

The Microsoft Store package is an easily installable Python interpreter that is intended mainly for interactive use, for example, by students.

To install the package, ensure you have the latest Windows 10 updates and search the Microsoft Store app for «Python 3.7». Ensure that the app you select is published by the Python Software Foundation, and install it.

Advertencia: Python will always be available for free on the Microsoft Store. If you are asked to pay for it, you have not selected the correct package.

After installation, Python may be launched by finding it in Start. Alternatively, it will be available from any Command Prompt or PowerShell session by typing `python`. Further, pip and IDLE may be used by typing `pip` or `idle`. IDLE can also be found in Start.

All three commands are also available with version number suffixes, for example, as `python3.exe` and `python3.x.exe` as well as `python.exe` (where `3.x` is the specific version you want to launch, such as 3.7).

Virtual environments can be created with `python -m venv` and activated and used as normal.

If you have installed another version of Python and added it to your `PATH` variable, it will be available as `python.exe` rather than the one from the Microsoft Store. To access the new installation, use `python3.exe` or `python3.x.exe`.

To remove Python, open Settings and use Apps and Features, or else find Python in Start and right-click to select Uninstall. Uninstalling will remove all packages you installed directly into this Python installation, but will not remove any virtual environments

3.2.1 Known Issues

Currently, the `py.exe` launcher cannot be used to start Python when it has been installed from the Microsoft Store.

Because of restrictions on Microsoft Store apps, Python scripts may not have full write access to shared locations such as `TEMP` and the registry. Instead, it will write to a private copy. If your scripts must modify the shared locations, you will need to install the full installer.

3.3 The nuget.org packages

Nuevo en la versión 3.5.2.

The nuget.org package is a reduced size Python environment intended for use on continuous integration and build systems that do not have a system-wide install of Python. While nuget is «the package manager for .NET», it also works perfectly fine for packages containing build-time tools.

Visit nuget.org for the most up-to-date information on using nuget. What follows is a summary that is sufficient for Python developers.

The `nuget.exe` command line tool may be downloaded directly from `https://aka.ms/nugetclidl`, for example, using curl or PowerShell. With the tool, the latest version of Python for 64-bit or 32-bit machines is installed using:

```
nuget.exe install python -ExcludeVersion -OutputDirectory .
nuget.exe install pythonx86 -ExcludeVersion -OutputDirectory .
```

To select a particular version, add a `-Version 3.x.y`. The output directory may be changed from `.`, and the package will be installed into a subdirectory. By default, the subdirectory is named the same as the package, and without the `-ExcludeVersion` option this name will include the specific version installed. Inside the subdirectory is a `tools` directory that contains the Python installation:

```
# Without -ExcludeVersion
> .\python.3.5.2\tools\python.exe -V
Python 3.5.2

# With -ExcludeVersion
> .\python\tools\python.exe -V
Python 3.5.2
```

In general, nuget packages are not upgradeable, and newer versions should be installed side-by-side and referenced using the full path. Alternatively, delete the package directory manually and install it again. Many CI systems will do this automatically if they do not preserve files between builds.

Alongside the `tools` directory is a `build\native` directory. This contains a MSBuild properties file `python.props` that can be used in a C++ project to reference the Python install. Including the settings will automatically use the headers and import libraries in your build.

The package information pages on nuget.org are www.nuget.org/packages/python for the 64-bit version and www.nuget.org/packages/pythonx86 for the 32-bit version.

3.4 The embeddable package

Nuevo en la versión 3.5.

The embedded distribution is a ZIP file containing a minimal Python environment. It is intended for acting as part of another application, rather than being directly accessed by end-users.

When extracted, the embedded distribution is (almost) fully isolated from the user's system, including environment variables, system registry settings, and installed packages. The standard library is included as pre-compiled and optimized `.pyc` files in a ZIP, and `python3.dll`, `python37.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependants, such as Idle), pip and the Python documentation are not included.

Nota: The embedded distribution does not include the Microsoft C Runtime and it is the responsibility of the application installer to provide this. The runtime may have already been installed on a user's system previously or automatically via Windows Update, and can be detected by finding `ucrtbase.dll` in the system directory.

Third-party packages should be installed by the application installer alongside the embedded distribution. Using pip to manage dependencies as for a regular Python installation is not supported with this distribution, though with some care it may be possible to include and use pip for automatic updates. In general, third-party packages should be treated as part of the application («vendoring») so that the developer can ensure compatibility with newer versions before providing updates to users.

The two recommended use cases for this distribution are described below.

3.4.1 Python Application

An application written in Python does not necessarily require users to be aware of that fact. The embedded distribution may be used in this case to include a private version of Python in an install package. Depending on how transparent it should be (or conversely, how professional it should appear), there are two options.

Using a specialized executable as a launcher requires some coding, but provides the most transparent experience for users. With a customized launcher, there are no obvious indications that the program is running on Python: icons can be customized, company and version information can be specified, and file associations behave properly. In most cases, a custom launcher should simply be able to call `Py_Main` with a hard-coded command line.

The simpler approach is to provide a batch file or generated shortcut that directly calls the `python.exe` or `pythonw.exe` with the required command-line arguments. In this case, the application will appear to be Python and not its actual name, and users may have trouble distinguishing it from other running Python processes or file associations.

With the latter approach, packages should be installed as directories alongside the Python executable to ensure they are available on the path. With the specialized launcher, packages can be located in other locations as there is an opportunity to specify the search path before launching the application.

3.4.2 Embedding Python

Applications written in native code often require some form of scripting language, and the embedded Python distribution can be used for this purpose. In general, the majority of the application is in native code, and some part will either invoke `python.exe` or directly use `python3.dll`. For either case, extracting the embedded distribution to a subdirectory of the application installation is sufficient to provide a loadable Python interpreter.

As with the application use, packages can be installed to any location as there is an opportunity to specify search paths before initializing the interpreter. Otherwise, there is no fundamental differences between using the embedded distribution and a regular installation.

3.5 Alternative bundles

Besides the standard CPython distribution, there are modified packages including additional functionality. The following is a list of popular versions and their key features:

ActivePython Installer with multi-platform compatibility, documentation, PyWin32

Anaconda Popular scientific modules (such as numpy, scipy and pandas) and the `conda` package manager.

Canopy A «comprehensive Python analysis environment» with editors and other development tools.

WinPython Windows-specific distribution with prebuilt scientific packages and tools for building packages.

Note that these packages may not include the latest versions of Python or other libraries, and are not maintained or supported by the core Python team.

3.6 Configuring Python

To run Python conveniently from a command prompt, you might consider changing some default environment variables in Windows. While the installer provides an option to configure the PATH and PATHEXT variables for you, this is only reliable for a single, system-wide installation. If you regularly use multiple versions of Python, consider using the *Python Launcher for Windows*.

3.6.1 Excursus: Setting environment variables

Windows allows environment variables to be configured permanently at both the User level and the System level, or temporarily in a command prompt.

To temporarily set environment variables, open Command Prompt and use the **set** command:

```
C:\>set PATH=C:\Program Files\Python 3.7;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

These changes will apply to any further commands executed in that console, and will be inherited by any applications started from the console.

Including the variable name within percent signs will expand to the existing value, allowing you to add your new value at either the start or the end. Modifying `PATH` by adding the directory containing **python.exe** to the start is a common way to ensure the correct version of Python is launched.

To permanently modify the default environment variables, click Start and search for "edit environment variables", or open System properties, *Advanced system settings* and click the *Environment Variables* button. In this dialog, you can add or modify User and System variables. To change System variables, you need non-restricted access to your machine (i.e. Administrator rights).

Nota: Windows will concatenate User variables *after* System variables, which may cause unexpected results when modifying `PATH`.

The *PYTHONPATH* variable is used by all versions of Python 2 and Python 3, so you should not permanently configure this variable unless it only includes code that is compatible with all of your installed Python versions.

Ver también:

https://www.microsoft.com/en-us/wdsi/help/folder-variables Environment variables in Windows NT

https://technet.microsoft.com/en-us/library/cc754250.aspx The SET command, for temporarily modifying environment variables

https://technet.microsoft.com/en-us/library/cc755104.aspx The SETX command, for permanently modifying environment variables

https://support.microsoft.com/en-us/help/310519/how-to-manage-environment-variables-in-windows-xp How To Manage Environment Variables in Windows XP

https://www.chem.gla.ac.uk/~louis/software/faq/q1.html Setting Environment variables, Louis J. Farrugia

3.6.2 Finding the Python executable

Distinto en la versión 3.5.

Besides using the automatically created start menu entry for the Python interpreter, you might want to start Python in the command prompt. The installer has an option to set that up for you.

On the first page of the installer, an option labelled «Add Python to PATH» may be selected to have the installer add the install location into the `PATH`. The location of the `Scripts\` folder is also added. This allows you to type **python** to run the interpreter, and **pip** for the package installer. Thus, you can also execute your scripts with command line options, see *Command line* documentation.

If you don't enable this option at install time, you can always re-run the installer, select Modify, and enable it. Alternatively, you can manually modify the `PATH` using the directions in *Excursus: Setting environment variables*. You need to set your `PATH` environment variable to include the directory of your Python installation, delimited by a semicolon from other entries. An example variable could look like this (assuming the first two entries already existed):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.7
```

3.7 UTF-8 mode

Nuevo en la versión 3.7.

Windows still uses legacy encodings for the system encoding (the ANSI Code Page). Python uses it for the default encoding of text files (e.g. `locale.getpreferredencoding()`).

This may cause issues because UTF-8 is widely used on the internet and most Unix systems, including WSL (Windows Subsystem for Linux).

You can use UTF-8 mode to change the default text encoding to UTF-8. You can enable UTF-8 mode via the `-X utf8` command line option, or the `PYTHONUTF8=1` environment variable. See *PYTHONUTF8* for enabling UTF-8 mode, and *Excursus: Setting environment variables* for how to modify environment variables.

When UTF-8 mode is enabled:

- `locale.getpreferredencoding()` returns `'UTF-8'` instead of the system encoding. This function is used for the default text encoding in many places, including `open()`, `Popen`, `Path.read_text()`, etc.
- `sys.stdin`, `sys.stdout`, and `sys.stderr` all use UTF-8 as their text encoding.
- You can still use the system encoding via the «mbcs» codec.

Note that adding `PYTHONUTF8=1` to the default environment variables will affect all Python 3.7+ applications on your system. If you have any Python 3.7+ applications which rely on the legacy system encoding, it is recommended to set the environment variable temporarily or use the `-X utf8` command line option.

Nota: Even when UTF-8 mode is disabled, Python uses UTF-8 by default on Windows for:

- Console I/O including standard I/O (see **PEP 528** for details).
- The filesystem encoding (see **PEP 529** for details).

3.8 Python Launcher for Windows

Nuevo en la versión 3.3.

The Python launcher for Windows is a utility which aids in locating and executing of different Python versions. It allows scripts (or the command-line) to indicate a preference for a specific Python version, and will locate and execute that version.

Unlike the `PATH` variable, the launcher will correctly select the most appropriate version of Python. It will prefer per-user installations over system-wide ones, and orders by language version rather than using the most recently installed version.

The launcher was originally specified in **PEP 397**.

3.8.1 Getting started

From the command-line

Distinto en la versión 3.6.

System-wide installations of Python 3.3 and later will put the launcher on your `PATH`. The launcher is compatible with all available versions of Python, so it does not matter which version is installed. To check that the launcher is available, execute the following command in Command Prompt:

```
py
```

You should find that the latest version of Python you have installed is started - it can be exited as normal, and any additional command-line arguments specified will be sent directly to Python.

If you have multiple versions of Python installed (e.g., 2.7 and 3.7) you will have noticed that Python 3.7 was started - to launch Python 2.7, try the command:

```
py -2.7
```

If you want the latest version of Python 2.x you have installed, try the command:

```
py -2
```

You should find the latest version of Python 2.x starts.

If you see the following error, you do not have the launcher installed:

```
'py' is not recognized as an internal or external command,
operable program or batch file.
```

Per-user installations of Python do not add the launcher to `PATH` unless the option was selected on installation.

Virtual environments

Nuevo en la versión 3.5.

If the launcher is run with no explicit Python version specification, and a virtual environment (created with the standard library `venv` module or the external `virtualenv` tool) active, the launcher will run the virtual environment's interpreter rather than the global one. To run the global interpreter, either deactivate the virtual environment, or explicitly specify the global Python version.

From a script

Let's create a test Python script - create a file called `hello.py` with the following contents

```
#! python
import sys
sys.stdout.write("hello from Python %s\n" % (sys.version,))
```

From the directory in which hello.py lives, execute the command:

```
py hello.py
```

You should notice the version number of your latest Python 2.x installation is printed. Now try changing the first line to be:

```
#! python3
```

Re-executing the command should now print the latest Python 3.x information. As with the above command-line examples, you can specify a more explicit version qualifier. Assuming you have Python 2.6 installed, try changing the first line to `#! python2.6` and you should find the 2.6 version information printed.

Note that unlike interactive use, a bare «python» will use the latest version of Python 2.x that you have installed. This is for backward compatibility and for compatibility with Unix, where the command `python` typically refers to Python 2.

From file associations

The launcher should have been associated with Python files (i.e. `.py`, `.pyw`, `.pyc` files) when it was installed. This means that when you double-click on one of these files from Windows explorer the launcher will be used, and therefore you can use the same facilities described above to have the script specify the version which should be used.

The key benefit of this is that a single launcher can support multiple Python versions at the same time depending on the contents of the first line.

3.8.2 Shebang Lines

If the first line of a script file starts with `#!`, it is known as a «shebang» line. Linux and other Unix like operating systems have native support for such lines and they are commonly used on such systems to indicate how a script should be executed. This launcher allows the same facilities to be used with Python scripts on Windows and the examples above demonstrate their use.

To allow shebang lines in Python scripts to be portable between Unix and Windows, this launcher supports a number of "virtual" commands to specify which interpreter to use. The supported virtual commands are:

- `/usr/bin/env python`
- `/usr/bin/python`
- `/usr/local/bin/python`
- `python`

For example, if the first line of your script starts with

```
#! /usr/bin/python
```

The default Python will be located and used. As many Python scripts written to work on Unix will already have this line, you should find these scripts can be used by the launcher without modification. If you are writing a new script on Windows which you hope will be useful on Unix, you should use one of the shebang lines starting with `/usr`.

Any of the above virtual commands can be suffixed with an explicit version (either just the major version, or the major and minor version). Furthermore the 32-bit version can be requested by adding «-32» after the minor version. I.e. `/usr/bin/python2.7-32` will request usage of the 32-bit python 2.7.

Nuevo en la versión 3.7: Beginning with python launcher 3.7 it is possible to request 64-bit version by the «-64» suffix. Furthermore it is possible to specify a major and architecture without minor (i.e. `/usr/bin/python3-64`).

The `/usr/bin/env` form of shebang line has one further special property. Before looking for installed Python interpreters, this form will search the executable `PATH` for a Python executable. This corresponds to the behaviour of the Unix `env` program, which performs a `PATH` search.

3.8.3 Arguments in shebang lines

The shebang lines can also specify additional options to be passed to the Python interpreter. For example, if you have a shebang line:

```
#! /usr/bin/python -v
```

Then Python will be started with the `-v` option

3.8.4 Customization

Customization via INI files

Two .ini files will be searched by the launcher - `py.ini` in the current user's «application data» directory (i.e. the directory returned by calling the Windows function `SHGetFolderPath` with `CSIDL_LOCAL_APPDATA`) and `py.ini` in the same directory as the launcher. The same .ini files are used for both the "console" version of the launcher (i.e. py.exe) and for the "windows" version (i.e. pyw.exe).

Customization specified in the «application directory» will have precedence over the one next to the executable, so a user, who may not have write access to the .ini file next to the launcher, can override commands in that global .ini file.

Customizing default Python versions

In some cases, a version qualifier can be included in a command to dictate which version of Python will be used by the command. A version qualifier starts with a major version number and can optionally be followed by a period (".") and a minor version specifier. Furthermore it is possible to specifiy if a 32 or 64 bit implementation shall be requested by adding «-32» or «-64».

For example, a shebang line of `#!python` has no version qualifier, while `#!python3` has a version qualifier which specifies only a major version.

If no version qualifiers are found in a command, the environment variable `PY_PYTHON` can be set to specify the default version qualifier. If it is not set, the default is «3». The variable can specify any value that may be passed on the command line, such as «3», «3.7», «3.7-32» or «3.7-64». (Note that the «-64» option is only available with the launcher included with Python 3.7 or newer.)

If no minor version qualifiers are found, the environment variable `PY_PYTHON{major}` (where `{major}` is the current major version qualifier as determined above) can be set to specify the full version. If no such option is found, the launcher will enumerate the installed Python versions and use the latest minor release found for the major version, which is likely, although not guaranteed, to be the most recently installed version in that family.

On 64-bit Windows with both 32-bit and 64-bit implementations of the same (major.minor) Python version installed, the 64-bit version will always be preferred. This will be true for both 32-bit and 64-bit implementations of the launcher - a 32-bit launcher will prefer to execute a 64-bit Python installation of the specified version if available. This is so the behavior of the launcher can be predicted knowing only what versions are installed on the PC and without regard to the order in which they were installed (i.e., without knowing whether a 32 or 64-bit version of Python and corresponding launcher was installed last). As noted above, an optional «-32» or «-64» suffix can be used on a version specifier to change this behaviour.

Examples:

- If no relevant options are set, the commands `python` and `python2` will use the latest Python 2.x version installed and the command `python3` will use the latest Python 3.x installed.
- The commands `python3.1` and `python2.7` will not consult any options at all as the versions are fully specified.
- If `PY_PYTHON=3`, the commands `python` and `python3` will both use the latest installed Python 3 version.
- If `PY_PYTHON=3.1-32`, the command `python` will use the 32-bit implementation of 3.1 whereas the command `python3` will use the latest installed Python (PY_PYTHON was not considered at all as a major version was specified.)
- If `PY_PYTHON=3` and `PY_PYTHON3=3.1`, the commands `python` and `python3` will both use specifically 3.1

In addition to environment variables, the same settings can be configured in the .INI file used by the launcher. The section in the INI file is called `[defaults]` and the key name will be the same as the environment variables without the leading `PY_` prefix (and note that the key names in the INI file are case insensitive.) The contents of an environment variable will override things specified in the INI file.

For example:

- Setting PY_PYTHON=3.1 is equivalent to the INI file containing:

```
[defaults]
python=3.1
```

- Setting PY_PYTHON=3 and PY_PYTHON3=3.1 is equivalent to the INI file containing:

```
[defaults]
python=3
python3=3.1
```

3.8.5 Diagnostics

If an environment variable PYLAUNCH_DEBUG is set (to any value), the launcher will print diagnostic information to stderr (i.e. to the console). While this information manages to be simultaneously verbose *and* terse, it should allow you to see what versions of Python were located, why a particular version was chosen and the exact command-line used to execute the target Python.

3.9 Finding modules

Python usually stores its library (and thereby your site-packages folder) in the installation directory. So, if you had installed Python to C:\Python\, the default library would reside in C:\Python\Lib\ and third-party modules should be stored in C:\Python\Lib\site-packages\.

To completely override sys.path, create a ._pth file with the same name as the DLL (python37._pth) or the executable (python._pth) and specify one line for each path to add to sys.path. The file based on the DLL name overrides the one based on the executable, which allows paths to be restricted for any program loading the runtime if desired.

When the file exists, all registry and environment variables are ignored, isolated mode is enabled, and site is not imported unless one line in the file specifies import site. Blank paths and lines starting with # are ignored. Each path may be absolute or relative to the location of the file. Import statements other than to site are not permitted, and arbitrary code cannot be specified.

Note that .pth files (without leading underscore) will be processed normally by the site module when import site has been specified.

When no ._pth file is found, this is how sys.path is populated on Windows:

- An empty entry is added at the start, which corresponds to the current directory.
- If the environment variable *PYTHONPATH* exists, as described in *Environment variables*, its entries are added next. Note that on Windows, paths in this variable must be separated by semicolons, to distinguish them from the colon used in drive identifiers (C:\ etc.).
- Additional «application paths» can be added in the registry as subkeys of \SOFTWARE\Python\PythonCore{version}\PythonPath under both the HKEY_CURRENT_USER and HKEY_LOCAL_MACHINE hives. Subkeys which have semicolon-delimited path strings as their default value will cause each path to be added to sys.path. (Note that all known installers only use HKLM, so HKCU is typically empty.)
- If the environment variable *PYTHONHOME* is set, it is assumed as «Python Home». Otherwise, the path of the main Python executable is used to locate a «landmark file» (either Lib\os.py or pythonXY.zip) to deduce the «Python Home». If a Python home is found, the relevant sub-directories added to sys.path (Lib, plat-win, etc) are based on that folder. Otherwise, the core Python path is constructed from the PythonPath stored in the registry.
- If the Python Home cannot be located, no *PYTHONPATH* is specified in the environment, and no registry entries can be found, a default path with relative entries is used (e.g. .\Lib;.\plat-win, etc).

If a pyvenv.cfg file is found alongside the main executable or in the directory one level above the executable, the following variations apply:

- If home is an absolute path and *PYTHONHOME* is not set, this path is used instead of the path to the main executable when deducing the home location.

The end result of all this is:

- When running python.exe, or any other .exe in the main Python directory (either an installed version, or directly from the PCbuild directory), the core path is deduced, and the core paths in the registry are ignored. Other «application paths» in the registry are always read.
- When Python is hosted in another .exe (different directory, embedded via COM, etc), the «Python Home» will not be deduced, so the core path from the registry is used. Other «application paths» in the registry are always read.
- If Python can't find its home and there are no registry value (frozen .exe, some very strange installation setup) you get a path with some default, but relative, paths.

For those who want to bundle Python into their application or distribution, the following advice will prevent conflicts with other installations:

- Include a ._pth file alongside your executable containing the directories to include. This will ignore paths listed in the registry and environment variables, and also ignore site unless import site is listed.
- If you are loading python3.dll or python37.dll in your own executable, explicitly call Py_SetPath() or (at least) Py_SetProgramName() before Py_Initialize().
- Clear and/or overwrite *PYTHONPATH* and set *PYTHONHOME* before launching python.exe from your application.
- If you cannot use the previous suggestions (for example, you are a distribution that allows people to run python.exe directly), ensure that the landmark file (Lib\os.py) exists in your install directory. (Note that it will not be detected inside a ZIP file, but a correctly named ZIP file will be detected instead.)

These will ensure that the files in a system-wide installation will not take precedence over the copy of the standard library bundled with your application. Otherwise, your users may experience problems using your application. Note that the first suggestion is the best, as the others may still be susceptible to non-standard paths in the registry and user site-packages.

Distinto en la versión 3.6:

- Adds ._pth file support and removes applocal option from pyvenv.cfg.
- Adds pythonXX.zip as a potential landmark when directly adjacent to the executable.

Obsoleto desde la versión 3.6: Modules specified in the registry under Modules (not PythonPath) may be imported by importlib.machinery.WindowsRegistryFinder. This finder is enabled on Windows in 3.6.0 and earlier, but may need to be explicitly added to sys.meta_path in the future.

3.10 Additional modules

Even though Python aims to be portable among all platforms, there are features that are unique to Windows. A couple of modules, both in the standard library and external, and snippets exist to use these features.

The Windows-specific standard modules are documented in mswin-specific-services.

3.10.1 PyWin32

The PyWin32 module by Mark Hammond is a collection of modules for advanced Windows-specific support. This includes utilities for:

- Component Object Model (COM)
- Win32 API calls
- Registry
- Event log
- Microsoft Foundation Classes (MFC) user interfaces

PythonWin is a sample MFC application shipped with PyWin32. It is an embeddable IDE with a built-in debugger.

Ver también:

Win32 How Do I…? by Tim Golden

Python and COM by David and Paul Boddie

3.10.2 cx_Freeze

cx_Freeze is a `distutils` extension (see extending-distutils) which wraps Python scripts into executable Windows programs (`*.exe` files). When you have done this, you can distribute your application without requiring your users to install Python.

3.10.3 WConio

Since Python's advanced terminal handling layer, `curses`, is restricted to Unix-like systems, there is a library exclusive to Windows as well: Windows Console I/O for Python.

WConio is a wrapper for Turbo-C's `CONIO.H`, used to create text user interfaces.

3.11 Compiling Python on Windows

If you want to compile CPython yourself, first thing you should do is get the source. You can download either the latest release's source or just grab a fresh checkout.

The source tree contains a build solution and project files for Microsoft Visual Studio 2015, which is the compiler used to build the official Python releases. These files are in the `PCbuild` directory.

Check `PCbuild/readme.txt` for general information on the build process.

For extension modules, consult building-on-windows.

Ver también:

Python + Windows + distutils + SWIG + gcc MinGW or «Creating Python extensions in C/C++ with SWIG and compiling them with MinGW gcc under Windows» or «Installing Python extension with distutils and without Microsoft Visual C++» by Sébastien Sauvage, 2003

MingW – Python extensions

3.12 Other Platforms

With ongoing development of Python, some platforms that used to be supported earlier are no longer supported (due to the lack of users or developers). Check **PEP 11** for details on all unsupported platforms.

- Windows CE is still supported.
- The Cygwin installer offers to install the Python interpreter as well (cf. Cygwin package source, Maintainer releases)

See Python for Windows for detailed information about platforms with pre-compiled installers.

CAPÍTULO 4

Using Python on a Macintosh

Author Bob Savage <bobsavage@mac.com>

Python on a Macintosh running Mac OS X is in principle very similar to Python on any other Unix platform, but there are a number of additional features such as the IDE and the Package Manager that are worth pointing out.

4.1 Getting and Installing MacPython

Mac OS X 10.8 comes with Python 2.7 pre-installed by Apple. If you wish, you are invited to install the most recent version of Python 3 from the Python website (https://www.python.org). A current «universal binary» build of Python, which runs natively on the Mac's new Intel and legacy PPC CPU's, is available there.

What you get after installing is a number of things:

- A `Python 3.7` folder in your `Applications` folder. In here you find IDLE, the development environment that is a standard part of official Python distributions; and PythonLauncher, which handles double-clicking Python scripts from the Finder.
- A framework `/Library/Frameworks/Python.framework`, which includes the Python executable and libraries. The installer adds this location to your shell path. To uninstall MacPython, you can simply remove these three things. A symlink to the Python executable is placed in /usr/local/bin/.

The Apple-provided build of Python is installed in `/System/Library/Frameworks/Python.framework` and `/usr/bin/python`, respectively. You should never modify or delete these, as they are Apple-controlled and are used by Apple- or third-party software. Remember that if you choose to install a newer Python version from python.org, you will have two different but functional Python installations on your computer, so it will be important that your paths and usages are consistent with what you want to do.

IDLE includes a help menu that allows you to access Python documentation. If you are completely new to Python you should start reading the tutorial introduction in that document.

If you are familiar with Python on other Unix platforms you should read the section on running Python scripts from the Unix shell.

4.1.1 How to run a Python script

Your best way to get started with Python on Mac OS X is through the IDLE integrated development environment, see section *The IDE* and use the Help menu when the IDE is running.

If you want to run Python scripts from the Terminal window command line or from the Finder you first need an editor to create your script. Mac OS X comes with a number of standard Unix command line editors, **vim** and **emacs** among them. If you want a more Mac-like editor, **BBEdit** or **TextWrangler** from Bare Bones Software (see http://www.barebones.com/products/bbedit/index.html) are good choices, as is **TextMate** (see https://macromates.com/). Other editors include **Gvim** (http://macvim-dev.github.io/macvim/) and **Aquamacs** (http://aquamacs.org/).

To run your script from the Terminal window you must make sure that `/usr/local/bin` is in your shell search path.

To run your script from the Finder you have two options:

- Drag it to **PythonLauncher**
- Select **PythonLauncher** as the default application to open your script (or any .py script) through the finder Info window and double-click it. **PythonLauncher** has various preferences to control how your script is launched. Option-dragging allows you to change these for one invocation, or use its Preferences menu to change things globally.

4.1.2 Running scripts with a GUI

With older versions of Python, there is one Mac OS X quirk that you need to be aware of: programs that talk to the Aqua window manager (in other words, anything that has a GUI) need to be run in a special way. Use **pythonw** instead of **python** to start such scripts.

With Python 3.7, you can use either **python** or **pythonw**.

4.1.3 Configuration

Python on OS X honors all standard Unix environment variables such as *PYTHONPATH*, but setting these variables for programs started from the Finder is non-standard as the Finder does not read your `.profile` or `.cshrc` at startup. You need to create a file `~/.MacOSX/environment.plist`. See Apple's Technical Document QA1067 for details.

For more information on installation Python packages in MacPython, see section *Installing Additional Python Packages*.

4.2 The IDE

MacPython ships with the standard IDLE development environment. A good introduction to using IDLE can be found at http://www.hashcollision.org/hkn/python/idle_intro/index.html.

4.3 Installing Additional Python Packages

There are several methods to install additional Python packages:

- Packages can be installed via the standard Python distutils mode (`python setup.py install`).
- Many packages can also be installed via the **`setuptools`** extension or **`pip`** wrapper, see https://pip.pypa.io/.

4.4 GUI Programming on the Mac

There are several options for building GUI applications on the Mac with Python.

PyObjC is a Python binding to Apple's Objective-C/Cocoa framework, which is the foundation of most modern Mac development. Information on PyObjC is available from https://pypi.org/project/pyobjc/.

The standard Python GUI toolkit is `tkinter`, based on the cross-platform Tk toolkit (https://www.tcl.tk). An Aqua-native version of Tk is bundled with OS X by Apple, and the latest version can be downloaded and installed from https://www.activestate.com; it can also be built from source.

wxPython is another popular cross-platform GUI toolkit that runs natively on Mac OS X. Packages and documentation are available from https://www.wxpython.org.

PyQt is another popular cross-platform GUI toolkit that runs natively on Mac OS X. More information can be found at https://riverbankcomputing.com/software/pyqt/intro.

4.5 Distributing Python Applications on the Mac

The standard tool for deploying standalone Python applications on the Mac is **`py2app`**. More information on installing and using py2app can be found at http://undefined.org/python/#py2app.

4.6 Other Resources

The MacPython mailing list is an excellent support resource for Python users and developers on the Mac:

https://www.python.org/community/sigs/current/pythonmac-sig/

Another useful resource is the MacPython wiki:

https://wiki.python.org/moin/MacPython

CAPÍTULO 5

Editors and IDEs

There are a number of IDEs that support Python programming language. Many editors and IDEs provide syntax highlighting, debugging tools, and **PEP 8** checks.

Please go to Python Editors and Integrated Development Environments for a comprehensive list.

APÉNDICE A

Glosario

>>> El prompt en el shell interactivo de Python por omisión. Frecuentemente vistos en ejemplos de código que pueden ser ejecutados interactivamente en el intérprete.

... The default Python prompt of the interactive shell when entering the code for an indented code block, when within a pair of matching left and right delimiters (parentheses, square brackets, curly braces or triple quotes), or after specifying a decorator.

2to3 Una herramienta que intenta convertir código de Python 2.x a Python 3.x arreglando la mayoría de las incompatibilidades que pueden ser detectadas analizando el código y recorriendo el árbol de análisis sintáctico.

2to3 está disponible en la biblioteca estándar como `lib2to3`; un punto de entrada independiente es provisto como `Tools/scripts/2to3`. Vea 2to3-reference.

clase base abstracta Las clases base abstractas (ABC, por sus siglas en inglés *Abstract Base Class*) complementan al *duck-typing* brindando un forma de definir interfaces con técnicas como `hasattr()` que serían confusas o sutilmente erróneas (por ejemplo con magic methods). Las ABC introduce subclases virtuales, las cuales son clases que no heredan desde una clase pero aún así son reconocidas por `isinstance()` y `issubclass()`; vea la documentación del módulo `abc`. Python viene con muchas ABC incorporadas para las estructuras de datos(en el módulo `collections.abc`), números (en el módulo `numbers`), flujos de datos (en el módulo `io`), buscadores y cargadores de importaciones (en el módulo `importlib.abc`). Puede crear sus propios ABCs con el módulo `abc`.

anotación Una etiqueta asociada a una variable, atributo de clase, parámetro de función o valor de retorno, usado por convención como un *type hint*.

Las anotaciones de variables no pueden ser accedidas en tiempo de ejecución, pero las anotaciones de variables globales, atributos de clase, y funciones son almacenadas en el atributo especial `__annotations__` de módulos, clases y funciones, respectivamente.

Vea *variable annotation*, *function annotation*, **PEP 484** y **PEP 526**, los cuales describen esta funcionalidad.

argumento Un valor pasado a una *function* (o *method*) cuando se llama a la función. Hay dos clases de argumentos:

- *argumento nombrado*: es un argumento precedido por un identificador (por ejemplo, `nombre=`) en una llamada a una función o pasado como valor en un diccionario precedido por `**`. Por ejemplo `3` y `5` son argumentos nombrados en las llamadas a `complex()`:

```
complex(real=3, imag=5)
complex(**{'real': 3, 'imag': 5})
```

- *argumento posicional* son aquellos que no son nombrados. Los argumentos posicionales deben aparecer al principio de una lista de argumentos o ser pasados como elementos de un *iterable* precedido por `*`. Por ejemplo, `3` y `5` son argumentos posicionales en las siguientes llamadas:

```
complex(3, 5)
complex(*(3, 5))
```

Los argumentos son asignados a las variables locales en el cuerpo de la función. Vea en la sección calls las reglas que rigen estas asignaciones. Sintácticamente, cualquier expresión puede ser usada para representar un argumento; el valor evaluado es asignado a la variable local.

Vea también el *parameter* en el glosario, la pregunta frecuente la diferencia entre argumentos y parámetros, y **PEP 362**.

administrador asincrónico de contexto Un objeto que controla el entorno visible en un sentencia `async with` al definir los métodos `__aenter__()` `__aexit__()`. Introducido por **PEP 492**.

generador asincrónico Una función que retorna un *asynchronous generator iterator*. Es similar a una función corrutina definida con `async def` excepto que contiene expresiones `yield` para producir series de variables usadas en un ciclo `async for`.

Usualmente se refiere a una función generadora asincrónica, pero puede referirse a un *iterador generador asincrónico* en ciertos contextos. En aquellos casos en los que el significado no está claro, usar los términos completos evita la ambigüedad.

Una función generadora asincrónica puede contener expresiones `await` así como sentencias `async for`, y `async with`.

iterador generador asincrónico Un objeto creado por una función *asynchronous generator*.

Este es un *asynchronous iterator* el cual cuando es llamado usa el método `__anext__()` retornando un objeto aguardable el cual ejecutará el cuerpo de la función generadora asincrónica hasta la siguiente expresión `yield`.

Cada `yield` suspende temporalmente el procesamiento, recordando el estado local de ejecución (incluyendo a las variables locales y las sentencias *try* pendientes). Cuando el *iterador del generador asincrónico* vuelve efectivamente con otro aguardable retornado por el método `__anext__()`, retoma donde lo dejó. Vea **PEP 492** y **PEP 525**.

iterable asincrónico Un objeto, que puede ser usado en una sentencia `async for`. Debe retornar un *asynchronous iterator* de su método `__aiter__()`. Introducido por **PEP 492**.

iterador asincrónico Un objeto que implementa los métodos meth:*__aiter__* y `__anext__()`. `__anext__` debe retornar un objeto *awaitable*. `async for` resuelve los esperables retornados por un método de iterador asincrónico `__anext__()` hasta que lanza una excepción `StopAsyncIteration`. Introducido por **PEP 492**.

atributo Un valor asociado a un objeto que es referencias por el nombre usado expresiones de punto. Por ejemplo, si un objeto *o* tiene un atributo *a* sería referenciado como *o.a*.

aguardable Es un objeto que puede ser usado en una expresión `await`. Puede ser una *coroutine* o un objeto con un método `__await__()`. Vea también pep:*492*.

BDFL Sigla de Benevolent Dictator For Life, Benevolente dictador vitalicio, es decir Guido van Rossum, el creador de Python.

archivo binario Un *file object* capaz de leer y escribir *objetos tipo binarios*. Ejemplos de archivos binarios son los abiertos en modo binario (`'rb'`, `'wb'` o `'rb+'`), `sys.stdin.buffer`, `sys.stdout.buffer`, e instancias de `io.BytesIO` y de `gzip.GzipFile`.

Vea también *text file* para un objeto archivo capaz de leer y escribir objetos `str`.

objetos tipo binarios Un objeto que soporta bufferobjects y puede exportar un buffer C-*contiguous*. Esto incluye todas los objetos `bytes`, `bytearray`, y `array.array`, así como muchos objetos comunes `memoryview`. Los objetos tipo binarios pueden ser usados para varias operaciones que usan datos binarios; éstas incluyen compresión, salvar a archivos binarios, y enviarlos a través de un socket.

Algunas operaciones necesitan que los datos binarios sean mutables. La documentación frecuentemente se refiere a éstos como «objetos tipo binario de lectura y escritura». Ejemplos de objetos de buffer mutables incluyen a `bytearray` y `memoryview` de la `bytearray`. Otras operaciones que requieren datos binarios almacenados en objetos inmutables («objetos tipo binario de sólo lectura»); ejemplos de éstos incluyen `bytes` y `memoryview` del objeto `bytes`.

bytecode El código fuente Python es compilado en bytecode, la representación interna de un programa python en el intérprete CPython. El bytecode también es guardado en caché en los archivos *.pyc* de tal forma que ejecutar el mismo archivo es más fácil la segunda vez (la recompilación desde el código fuente a bytecode puede ser evitada). Este «lenguaje intermedio» deberá corren en una *virtual machine* que ejecute el código de máquina correspondiente a cada bytecode. Note que los bytecodes no tienen como requisito trabajar en las diversas máquina virtuales de Python, ni de ser estable entre versiones Python.

Una lista de las instrucciones en bytecode está disponible en la documentación de el módulo dis.

clase Una plantilla para crear objetos definidos por el usuario. Las definiciones de clase normalmente contienen definiciones de métodos que operan una instancia de la clase.

variable de clase Una variable definida en una clase y prevista para ser modificada sólo a nivel de clase (es decir, no en una instancia de la clase).

coerción La conversión implícita de una instancia de un tipo en otra durante una operación que involucra dos argumentos del mismo tipo. Por ejemplo, `int(3.15)` convierte el número de punto flotante al entero `3`, pero en `3 + 4.5`, cada argumento es de un tipo diferente (uno entero, otro flotante), y ambos deben ser convertidos al mismo tipo antes de que puedan ser sumados o emitiría un `TypeError`. Sin coerción, todos los argumentos, incluso de tipos compatibles, deberían ser normalizados al mismo tipo por el programador, por ejemplo `float(3)+4.5` en lugar de `3+4.5`.

número complejo Una extensión del sistema familiar de número reales en el cual los números son expresados como la suma de una parte real y una parte imaginaria. Los números imaginarios son múltiplos de la unidad imaginaria (la raíz cuadrada de `-1`), usualmente escrita como `i` en matemáticas o `j` en ingeniería. Python tiene soporte incorporado para números complejos, los cuales son escritos con la notación mencionada al final.; la parte imaginaria es escrita con un sufijo `j`, por ejemplo, `3+1j`. Para tener acceso a los equivalentes complejos del módulo `math` module, use :mod:'cmath. El uso de números complejos es matemática bastante avanzada. Si no le parecen necesarios, puede ignorarlos sin inconvenientes.

administrador de contextos Un objeto que controla el entorno en la sentencia `with` definiendo `__enter__()` y `__exit__()` methods. Vea **PEP 343**.

context variable A variable which can have different values depending on its context. This is similar to Thread-Local Storage in which each execution thread may have a different value for a variable. However, with context variables, there may be several contexts in one execution thread and the main usage for context variables is to keep track of variables in concurrent asynchronous tasks. See `contextvars`.

contiguo Un buffer es considerado contiguo con precisión si es *C-contíguo* o *Fortran contiguo*. Los buffers cero dimensionales con C y Fortran contiguos. En los arreglos unidimensionales, los ítems deben ser dispuestos en memoria uno siguiente al otro, ordenados por índices que comienzan en cero. En arreglos unidimensionales C-contíguos, el último índice varía más velozmente en el orden de las direcciones de memoria. Sin embargo, en arreglos Fortran contiguos, el primer índice vería más rápidamente.

corrutina Coroutines are a more generalized form of subroutines. Subroutines are entered at one point and exited at another point. Coroutines can be entered, exited, and resumed at many different points. They can be implemented with the `async def` statement. See also **PEP 492**.

función corrutina Un función que retorna un objeto *coroutine* . Una función corrutina puede ser definida con la sentencia `async def`, y puede contener las palabras claves `await`, `async for`, y `async with`. Las mismas son introducidas en **PEP 492**.

CPython La implementación canónica del lenguaje de programación Python, como se distribuye en python.org. El término «CPython» es usado cuando es necesario distinguir esta implementación de otras como Jython o IronPython.

decorador Una función que retorna otra función, usualmente aplicada como una función de transformación empleando la sintaxis `@envoltorio`. Ejemplos comunes de decoradores son `classmethod()` y func:*staticmethod*.

La sintaxis del decorador es meramente azúcar sintáctico, las definiciones de las siguientes dos funciones son semánticamente equivalentes:

```
def f(...):
    ...
f = staticmethod(f)

@staticmethod
def f(...):
    ...
```

El mismo concepto existe para clases, pero son menos usadas. Vea la documentación de function definitions y class definitions para mayor detalle sobre decoradores.

descriptor Cualquier objeto que define los métodos `__get__()`, `__set__()`, o `__delete__()`. Cuando un atributo de clase es un descriptor, su conducta enlazada especial es disparada durante la búsqueda del atributo. Normalmente, usando *a.b* para consultar, establecer o borrar un atributo busca el objeto llamado *b* en el diccionario de clase de *a*, pero si *b* es un descriptor, el respectivo método descriptor es llamado. Entender descriptores es clave para lograr una comprensión profunda de Python porque son la base de muchas de las capacidades incluyendo funciones, métodos, propiedades, métodos de clase, métodos estáticos, y referencia a súper clases.

Para más información sobre métodos descriptores, vea descriptors.

diccionario Un arreglo asociativo, con claves arbitrarias que son asociadas a valores. Las claves pueden ser cualquier objeto con los métodos `__hash__()` y `__eq__()` . Son llamadas hash en Perl.

vista de diccionario Los objetos retornados por los métodos `dict.keys()`, `dict.values()`, y `dict.items()` son llamados vistas de diccionarios. Proveen una vista dinámica de las entradas de un diccionario, lo que significa que cuando el diccionario cambia, la vista refleja éstos cambios. Para forzar a la vista de diccionario a convertirse en una lista completa, use `list(dictview)`. Vea dict-views.

docstring Una cadena de caracteres literal que aparece como la primera expresión en una clase, función o módulo. Aunque es ignorada cuando se ejecuta, es reconocida por el compilador y puesta en el atributo `__doc__` de la clase, función o módulo comprendida. Como está disponible mediante introspección, es el lugar canónico para ubicar la documentación del objeto.

tipado de pato Un estilo de programación que no revisa el tipo del objeto para determinar si tiene la interfaz correcta; en vez de ello, el método o atributo es simplemente llamado o usado («Si se ve como un pato y grazna como un pato, debe ser un pato»). Enfatizando las interfaces en vez de hacerlo con los tipos específicos, un código bien diseñado pues tener mayor flexibilidad permitiendo la sustitución polimórfica. El tipado de pato *duck-typing* evita usar pruebas llamando a `type()` o `isinstance()`. (Nota: si embargo, el tipado de pato puede ser complementado con *abstract base classes*. En su lugar, generalmente emplea `hasattr()` tests o *EAFP*.

EAFP Del inglés «Easier to ask for forgiveness than permission», es más fácil pedir perdón que pedir permiso. Este estilo de codificación común en Python asume la existencia de claves o atributos válidos y atrapa las excepciones si esta suposición resulta falsa. Este estilo rápido y limpio está caracterizado por muchas sentencias `try` y `except`. Esta técnica contrasta con estilo *LBYL* usual en otros lenguajes como C.

expresión Una construcción sintáctica que puede ser evaluada, hasta dar un valor. En otras palabras, una expresión es una acumulación de elementos de expresión tales como literales, nombres, accesos a atributos, operadores o llamadas a funciones, todos ellos retornando valor. A diferencia de otros lenguajes, no toda la sintaxis del lenguaje son expresiones. También hay *statement*s que no pueden ser usadas como expresiones, como la `while`. Las asignaciones también son sentencias, no expresiones.

módulo de extensión Un módulo escrito en C o C++, usando la API para C de Python para interactuar con el núcleo y el código del usuario.

f-string Son llamadas «f-strings» las cadenas literales que usan el prefijo `'f'` o `'F'`, que es una abreviatura para cadenas literales formateadas. Vea también **PEP 498**.

objeto archivo Un objeto que expone una API orientada a archivos (con métodos como `read()` o `write()`) al objeto subyacente. Dependiendo de la forma en la que fue creado, un objeto archivo, puede mediar el acceso a un archivo real en el disco u otro tipo de dispositivo de almacenamiento o de comunicación (por ejemplo, entrada/salida estándar, buffer de memoria, sockets, pipes, etc.). Los objetos archivo son también denominados *objetos tipo archivo* o *flujos*.

Existen tres categorías de objetos archivo: crudos *raw* *archivos binarios*, con buffer *archivos binarios* y *archivos de texto*. Sus interfaces son definidas en el módulo `io`. La forma canónica de crear objetos archivo es usando la función `open()`.

objetos tipo archivo Un sinónimo de *file object*.

buscador Un objeto que trata de encontrar el *loader* para el módulo que está siendo importado.

Desde la versión 3.3 de Python, existen dos tipos de buscadores: *meta buscadores de ruta* para usar con `sys.meta_path`, y *buscadores de entradas de rutas* para usar con `sys.path_hooks`.

Vea **PEP 302**, **PEP 420** y **PEP 451** para mayores detalles.

división entera Una división matemática que se redondea hacia el entero menor más cercano. El operador de la división entera es `//`. Por ejemplo, la expresión `11 // 4` evalúa `2` a diferencia del `2.75` retornado por la verdadera división de números flotantes. Note que `(-11) // 4` es `-3` porque es `-2.75` redondeado *para abajo*. Ver **PEP 238**.

función Una serie de sentencias que retornan un valor al que las llama. También se le puede pasar cero o más *argumentos* los cuales pueden ser usados en la ejecución de la misma. Vea también *parameter*, *method*, y la sección function.

anotación de función Una *annotation* del parámetro de una función o un valor de retorno.

Las anotaciones de funciones son usadas frecuentemente para type hint's , por ejemplo, se espera que una función tome dos argumentos de clase :class:'int y también se espera que devuelva dos valores `int`:

```
def sum_two_numbers(a: int, b: int) -> int:
   return a + b
```

La sintaxis de las anotaciones de funciones son explicadas en la sección function.

Vea *variable annotation* y **PEP 484**, que describen esta funcionalidad.

__future__ Un pseudo-módulo que los programadores pueden usar para habilitar nuevas capacidades del lenguaje que no son compatibles con el intérprete actual.

Al importar el módulo `__future__` y evaluar sus variables, puede verse cuándo las nuevas capacidades fueron agregadas por primera vez al lenguaje y cuando se quedaron establecidas por defecto:

```
>>> import __future__
>>> __future__.division
_Feature((2, 2, 0, 'alpha', 2), (3, 0, 0, 'alpha', 0), 8192)
```

recolección de basura El proceso de liberar la memoria de lo que ya no está en uso. Python realiza recolección de basura (*garbage collection*) llevando la cuenta de las referencias, y el recogedor de basura cíclico es capaz de detectar y romper las referencias cíclicas. El recogedor de basura puede ser controlado mediante el módulo `gc` .

generador Una función que retorna un *generator iterator*. Luce como una función normal excepto que contiene la expresión `yield` para producir series de valores utilizables en un bucle for o que pueden ser obtenidas una por una con la función `next()`.

Usualmente se refiere a una función generadora, pero puede referirse a un *iterador generador* en ciertos contextos. En aquellos casos en los que el significado no está claro, usar los términos completos evita la ambigüedad.

iterador generador Un objeto creado por una función *generator*.

Cada `yield` suspende temporalmente el procesamiento, recordando el estado de ejecución local (incluyendo las variables locales y las sentencias try pendientes). Cuando el «iterador generado» vuelve, retoma donde ha dejado, a diferencia de lo que ocurre con las funciones que comienzan nuevamente con cada invocación.

expresión generadora Una expresión que retorna un iterador. Luce como una expresión normal seguida por la cláusula `for` definiendo así una variable de bucle, un rango y una cláusula opcional `if`. La expresión combinada genera valores para la función contenedora:

```
>>> sum(i*i for i in range(10))         # sum of squares 0, 1, 4, ... 81
285
```

función genérica Una función compuesta de muchas funciones que implementan la misma operación para diferentes tipos. Qué implementación deberá ser usada durante la llamada a la misma es determinado por el algoritmo de despacho.

Vea también la entrada de glosario *single dispatch*, el decorador `functools.singledispatch()`, y **PEP 443**.

GIL Vea *global interpreter lock*.

bloqueo global del intérprete Mecanismo empleado por el intérprete *CPython* para asegurar que sólo un hilo ejecute el *bytecode* Python por vez. Esto simplifica la implementación de CPython haciendo que el modelo de objetos (incluyendo algunos críticos como `dict`) están implícitamente a salvo de acceso concurrente. Bloqueando el intérprete completo se simplifica hacerlo multi-hilos, a costa de mucho del paralelismo ofrecido por las máquinas con múltiples procesadores.

Sin embargo, algunos módulos de extensión, tanto estándar como de terceros, están diseñados para liberar el GIL cuando se realizan tareas computacionalmente intensivas como la compresión o el hashing. Además, el GIL siempre es liberado cuando se hace entrada/salida.

Esfuerzos previos hechos para crear un intérprete «sin hilos» (uno que bloquee los datos compartidos con una granularidad mucho más fina) no han sido exitosos debido a que el rendimiento sufrió para el caso más común de un solo procesador. Se cree que superar este problema de rendimiento haría la implementación mucho más compleja y por tanto, más costosa de mantener.

hash-based pyc Un archivo cache de bytecode que usa el hash en vez de usar el tiempo de la última modificación del archivo fuente correspondiente para determinar su validez. Vea pyc-invalidation.

hashable Un objeto es *hashable* si tiene un valor de hash que nunca cambiará durante su tiempo de vida (necesita un método `__hash__()`), y puede ser comparado con otro objeto (necesita el método `__eq__()`). Los objetos hashables que se comparan iguales deben tener el mismo número hash.

La hashabilidad hace a un objeto empleable como clave de un diccionario y miembro de un set, porque éstas estructuras de datos usan los valores de hash internamente.

Most of Python's immutable built-in objects are hashable; mutable containers (such as lists or dictionaries) are not; immutable containers (such as tuples and frozensets) are only hashable if their elements are hashable. Objects which are instances of user-defined classes are hashable by default. They all compare unequal (except with themselves), and their hash value is derived from their `id()`.

IDLE El entorno integrado de desarrollo de Python, o «Integrated Development Environment for Python». IDLE es un editor básico y un entorno de intérprete que se incluye con la distribución estándar de Python.

inmutable Un objeto con un valor fijo. Los objetos inmutables son números, cadenas y tuplas. Éstos objetos no pueden ser alterados. Un nuevo objeto debe ser creado si un valor diferente ha de ser guardado. Juegan un rol importante en lugares donde es necesario un valor de hash constante, por ejemplo como claves de un diccionario.

ruta de importación Una lista de las ubicaciones (o *entradas de ruta*) que son revisadas por *path based finder* al importar módulos. Durante la importación, ésta lista de localizaciones usualmente viene de `sys.path`, pero para los subpaquetes también puede incluir al atributo `__path__` del paquete padre.

importar El proceso mediante el cual el código Python dentro de un módulo se hace alcanzable desde otro código Python en otro módulo.

importador Un objeto que buscan y lee un módulo; un objeto que es tanto *finder* como *loader*.

interactivo Python tiene un intérprete interactivo, lo que significa que puede ingresar sentencias y expresiones en el prompt del intérprete, ejecutarlos de inmediato y ver sus resultados. Sólo ejecute `python` sin argumentos (podría seleccionarlo desde el menú principal de su computadora). Es una forma muy potente de probar nuevas ideas o inspeccionar módulos y paquetes (recuerde `help(x)`).

interpretado Python es un lenguaje interpretado, a diferencia de uno compilado, a pesar de que la distinción puede ser difusa debido al compilador a bytecode. Esto significa que los archivos fuente pueden ser corridos directamente, sin crear explícitamente un ejecutable que es corrido luego. Los lenguajes interpretados típicamente tienen ciclos de desarrollo y depuración más cortos que los compilados, sin embargo sus programas suelen correr más lentamente. Vea también *interactive*.

apagado del intérprete Cuando se le solicita apagarse, el intérprete Python ingresa a un fase especial en la cual gradualmente libera todos los recursos reservados, como módulos y varias estructuras internas críticas. También hace varias llamadas al *recolector de basura*. Esto puede disparar la ejecución de código de destructores definidos por el usuario o «weakref callbacks». El código ejecutado durante la fase de apagado puede encontrar varias excepciones debido a que los recursos que necesita pueden no funcionar más (ejemplos comunes son los módulos de bibliotecas o los artefactos de advertencias «warnings machinery»)

La principal razón para el apagado del intérpreter es que el módulo `__main__` o el script que estaba corriendo termine su ejecución.

iterable Un objeto capaz de retornar sus miembros uno por vez. Ejemplos de iterables son todos los tipos de secuencias (como `list`, `str`, y `tuple`) y algunos de tipos no secuenciales, como `dict`, *objeto archivo*, y objetos de cualquier clase que defina con los métodos `__iter__()` o con un método `__getitem__()` que implementen la semántica de *Sequence*.

Los iterables pueden ser usados en el bucle `for` y en muchos otros sitios donde una secuencia es necesaria (`zip()`, `map()`, …). Cuando un objeto iterable es pasado como argumento a la función incorporada `iter()`, retorna un iterador para el objeto. Este iterador pasa así el conjunto de valores. Cuando se usan iterables, normalmente no es necesario llamar a la función `iter()` o tratar con los objetos iteradores usted mismo. La sentencia `for` lo hace automáticamente por usted, creando un variable temporal sin nombre para mantener el iterador mientras dura el bucle. Vea también *iterator*, *sequence*, y *generator*.

iterador Un objeto que representa un flujo de datos. Llamadas repetidas al método `__next__()` del iterador (o al pasar la función incorporada `next()`) retorna ítems sucesivos del flujo. Cuando no hay más datos disponibles, una excepción `StopIteration` es disparada. En este momento, el objeto iterador está exhausto y cualquier llamada posterior al método `__next__()` sólo dispara otra vez `StopIteration`. Los iteradores necesitan tener un método:meth:*__iter__* que retorna el objeto iterador mismo así cada iterador es también un iterable y puede ser usado en casi todos los lugares donde los iterables son aceptados. Una excepción importante es el código que intenta múltiples pases de iteración. Un objeto contenedor (como la `list`) produce un nuevo iterador cada vez que las pasa a una función `iter()` o la usa en un blucle `for`. Intentar ésto con un iterador simplemente retornaría el mismo objeto iterador exhausto usado en previas iteraciones, haciéndolo aparecer como un contenedor vacío.

Puede encontrar más información en typeiter.

función clave Una función clave o una función de colación es un invocable que retorna un valor usado para el ordenamiento o clasificación. Por ejemplo, `locale.strxfrm()` es usada para producir claves de ordenamiento que se adaptan a las convenciones específicas de ordenamiento de un locale.

Cierta cantidad de herramientas de Python aceptan funciones clave para controlar como los elementos son ordenados o agrupados. Incluyendo a `min()`, `max()`, `sorted()`, `list.sort()`, `heapq.merge()`, `heapq.nsmallest()`, `heapq.nlargest()`, y `itertools.groupby()`.

Hay varias formas de crear una función clave. Por ejemplo, el método `str.lower()` puede servir como función clave para ordenamientos que no distingan mayúsculas de minúsculas. Como alternativa, una función clave puede ser realizada con una expresión `lambda` como `lambda r: (r[0], r[2])`. También, el módulo `operator` provee tres constructores de funciones clave: `attrgetter()`, `itemgetter()`, y `methodcaller()`. Vea en Sorting HOW TO ejemplos de cómo crear y usar funciones clave.

argumento nombrado Vea *argument*.

lambda Una función anónima de una línea consistente en un sola *expression* que es evaluada cuando la función es llamada. La sintaxis para crear una función lambda es `lambda [parameters]: expression`

LBYL Del inglés «Look before you leap», «mira antes de saltar». Es un estilo de codificación que prueba explícitamente las condiciones previas antes de hacer llamadas o búsquedas. Este estilo contrasta con la manera *EAFP* y está caracterizado por la presencia de muchas sentencias `if`.

En entornos multi-hilos, el método LBYL tiene el riesgo de introducir condiciones de carrera entre los hilos que están «mirando» y los que están «saltando». Por ejemplo, el código, *if key in mapping: return mapping[key]‘* puede fallar si otro hilo remueve *key* de *mapping* después del test, pero antes de retornar el valor. Este problema puede ser resuelto usando bloqueos o empleando el método EAFP.

lista Es una *sequence* Python incorporada. A pesar de su nombre es más similar a un arreglo en otros lenguajes que a una lista enlazada porque el acceso a los elementos es O(1).

comprensión de listas Una forma compacta de procesar todos o parte de los elementos en una secuencia y retornar una lista como resultado. `result = ['{:#04x}'.format(x) for x in range(256) if x % 2 == 0]` genera una lista de cadenas conteniendo números hexadecimales (0x..) entre 0 y 255. La cláusula `if` es opcional. Si es omitida, todos los elementos en `range(256)` son procesados.

cargador Un objeto que carga un módulo. Debe definir el método llamado `load_module()`. Un cargador es normalmente retornados por un *finder*. Vea **PEP 302** para detalles y `importlib.abc.Loader` para una *abstract base class*.

método mágico Una manera informal de llamar a un *special method*.

mapeado Un objeto contenedor que permite recupero de claves arbitrarias y que implementa los métodos especificados en la `Mapping` o `MutableMapping` abstract base classes. Por ejemplo, `dict`, `collections.defaultdict`, `collections.OrderedDict` y `collections.Counter`.

meta buscadores de ruta Un *finder* retornado por una búsqueda de `sys.meta_path`. Los meta buscadores de ruta están relacionados a *buscadores de entradas de rutas*, pero son algo diferente.

Vea en `importlib.abc.MetaPathFinder` los métodos que los meta buscadores de ruta implementan.

metaclase La clase de una clase. Las definiciones de clases crean nombres de clase, un diccionario de clase, y una lista de clases base. Las metaclases son responsables de tomar estos tres argumentos y crear la clase. La mayoría de los objetos de un lenguaje de programación orientado a objetos provienen de una implementación por defecto. Lo que hace a Python especial que es posible crear metaclases a medida. La mayoría de los usuario nunca necesitarán esta herramienta, pero cuando la necesidad surge, las metaclases pueden brindar soluciones poderosas y elegantes. Han sido usadas para loggear acceso de atributos, agregar seguridad a hilos, rastrear la creación de objetos, implementar singletons, y muchas otras tareas.

Más información hallará en metaclasses.

método Una función que es definida dentro del cuerpo de una clase. Si es llamada como un atributo de una instancia de otra clase, el método tomará el objeto instanciado como su primer *argument* (el cual es usualmente denominado *self*). Vea *function* y *nested scope*.

orden de resolución de métodos Orden de resolución de métodos es el orden en el cual una clase base es buscada por un miembro durante la búsqueda. Mire en The Python 2.3 Method Resolution Order los detalles del algoritmo usado por el intérprete Python desde la versión 2.3.

módulo Un objeto que sirve como unidad de organización del código Python. Los módulos tienen espacios de nombres conteniendo objetos Python arbitrarios. Los módulos son cargados en Python por el proceso de *importing*.

Vea también *package*.

especificador de módulo Un espacio de nombres que contiene la información relacionada a la importación usada al leer un módulo. Una instancia de `importlib.machinery.ModuleSpec`.

MRO Vea *method resolution order*.

mutable Los objetos mutables pueden cambiar su valor pero mantener su `id()`. Vea también *immutable*.

tupla nombrada The term «named tuple» applies to any type or class that inherits from tuple and whose indexable elements are also accessible using named attributes. The type or class may have other features as well.

Several built-in types are named tuples, including the values returned by `time.localtime()` and `os.stat()`. Another example is `sys.float_info`:

```
>>> sys.float_info[1]                   # indexed access
1024
>>> sys.float_info.max_exp              # named field access
```

(continué en la próxima página)

(proviene de la página anterior)

```
1024
>>> isinstance(sys.float_info, tuple)    # kind of tuple
True
```

Some named tuples are built-in types (such as the above examples). Alternatively, a named tuple can be created from a regular class definition that inherits from `tuple` and that defines named fields. Such a class can be written by hand or it can be created with the factory function `collections.namedtuple()`. The latter technique also adds some extra methods that may not be found in hand-written or built-in named tuples.

espacio de nombres El lugar donde la variable es almacenada. Los espacios de nombres son implementados como diccionarios. Hay espacio de nombre local, global, e incorporado así como espacios de nombres anidados en objetos (en métodos). Los espacios de nombres soportan modularidad previniendo conflictos de nombramiento. Por ejemplo, las funciones `builtins.open` y `os.open()` se distinguen por su espacio de nombres. Los espacios de nombres también ayuda a la legibilidad y mantenibilidad dejando claro qué módulo implementa una función. Por ejemplo, escribiendo `random.seed()` o `itertools.islice()` queda claro que éstas funciones están implementadas en los módulos `random` y `itertools`, respectivamente.

paquete de espacios de nombres Un **PEP 420** *package* que sirve sólo para contener subpaquetes. Los paquetes de espacios de nombres pueden no tener representación física, y específicamente se diferencian de los *regular package* porque no tienen un archivo `__init__.py`.

Vea también *module*.

alcances anidados La habilidad de referirse a una variable dentro de una definición encerrada. Por ejemplo, una función definida dentro de otra función puede referir a variables en la función externa. Note que los alcances anidados por defecto sólo funcionan para referencia y no para asignación. Las variables locales leen y escriben sólo en el alcance más interno. De manera semejante, las variables globales pueden leer y escribir en el espacio de nombres global. Con `nonlocal` se puede escribir en alcances exteriores.

clase de nuevo estilo Vieja denominación usada para el estilo de clases ahora empleado en todos los objetos de clase. En versiones más tempranas de Python, sólo las nuevas clases podían usar capacidades nuevas y versátiles de Python como `__slots__`, descriptores, propiedades, `__getattribute__()`, métodos de clase y métodos estáticos.

objeto Cualquier dato con estado (atributo o valor) y comportamiento definido (métodos). También es la más básica clase base para cualquier *new-style class*.

paquete Un *module* Python que puede contener submódulos o recursivamente, subpaquetes. Técnicamente, un paquete es un módulo Python con un atributo `__path__`.

Vea también *regular package* y *namespace package*.

parámetro Una entidad nombrada en una definición de una *function* (o método) que especifica un *argument* (o en algunos casos, varios argumentos) que la función puede aceptar. Existen cinco tipos de argumentos:

- *posicional o nombrado*: especifica un argumento que puede ser pasado tanto como *posicional* o como *nombrado*. Este es el tipo por defecto de parámetro, como *foo* y *bar* en el siguiente ejemplo:

```
def func(foo, bar=None): ...
```

- *sólo posicional*: especifica un argumento que puede ser pasado sólo por posición. Python no tiene una sintaxis específica para los parámetros que son sólo por posición. Sin embargo, algunas funciones tienen parámetros sólo por posición (por ejemplo `abs()`).

- *sólo nombrado*: especifica un argumento que sólo puede ser pasado por nombre. Los parámetros sólo por nombre pueden ser definidos incluyendo un parámetro posicional de una sola variable o un mero `*`` antes de ellos en la lista de parámetros en la definición de la función, como *kw_only1* y *kw_only2* en el ejemplo siguiente:

```
def func(arg, *, kw_only1, kw_only2): ...
```

- *variable posicional*: especifica una secuencia arbitraria de argumentos posicionales que pueden ser brindados (además de cualquier argumento posicional aceptado por otros parámetros). Este parámetro puede ser definido anteponiendo al nombre del parámetro *, como a *args* en el siguiente ejemplo:

```
def func(*args, **kwargs): ...
```

- *variable nombrado*: especifica que arbitrariamente muchos argumentos nombrados pueden ser brindados (además de cualquier argumento nombrado ya aceptado por cualquier otro parámetro). Este parámetro puede ser definido anteponiendo al nombre del parámetro con **, como *kwargs* en el ejemplo más arriba.

Los parámetros puede especificar tanto argumentos opcionales como requeridos, así como valores por defecto para algunos argumentos opcionales.

Vea también el glosario de *argument*, la pregunta respondida en la diferencia entre argumentos y parámetros, la clase `inspect.Parameter`, la sección function , y **PEP 362**.

entrada de ruta Una ubicación única en el *import path* que el *path based finder* consulta para encontrar los módulos a importar.

buscador de entradas de ruta Un *finder* retornado por un invocable en `sys.path_hooks` (esto es, un *path entry hook*) que sabe cómo localizar módulos dada una *path entry*.

Vea en `importlib.abc.PathEntryFinder` los métodos que los buscadores de entradas de paths implementan.

gancho a entrada de ruta Un invocable en la lista `sys.path_hook` que retorna un *path entry finder* si éste sabe cómo encontrar módulos en un *path entry* específico.

buscador basado en ruta Uno de los *meta buscadores de ruta* por defecto que busca un *import path* para los módulos.

objeto tipo ruta Un objeto que representa una ruta del sistema de archivos. Un objeto tipo ruta puede ser tanto una `str` como un `bytes` representando una ruta, o un objeto que implementa el protocolo `os.PathLike`. Un objeto que soporta el protocolo `os.PathLike` puede ser convertido a ruta del sistema de archivo de clase `str` o `bytes` usando la función `os.fspath()`; `os.fsdecode()` `os.fsencode()` pueden emplearse para garantizar que retorne respectivamente `str` o `bytes`. Introducido por **PEP 519**.

PEP Propuesta de mejora de Python, del inglés «Python Enhancement Proposal». Un PEP es un documento de diseño que brinda información a la comunidad Python, o describe una nueva capacidad para Python, sus procesos o entorno. Los PEPs deberían dar una especificación técnica concisa y una fundamentación para las capacidades propuestas.

Los PEPs tienen como propósito ser los mecanismos primarios para proponer nuevas y mayores capacidad, para recoger la opinión de la comunidad sobre un tema, y para documentar las decisiones de diseño que se han hecho en Python. El autor del PEP es el responsable de lograr consenso con la comunidad y documentar las opiniones disidentes.

Vea **PEP 1**.

porción Un conjunto de archivos en un único directorio (posiblemente guardo en un archivo comprimido zip) que contribuye a un espacio de nombres de paquete, como está definido en **PEP 420**.

argumento posicional Vea *argument*.

API provisoria Una API provisoria es aquella que deliberadamente fue excluida de las garantías de compatibilidad hacia atrás de la biblioteca estándar. Aunque no se esperan cambios fundamentales en dichas interfaces, como están marcadas como provisionales, los cambios incompatibles hacia atrás (incluso remover la misma interfaz) podrían ocurrir si los desarrolladores principales lo estiman. Estos cambios no se hacen gratuitamente – solo ocurrirán si fallas fundamentales y serias son descubiertas que no fueron vistas antes de la inclusión de la API.

Incluso para APIs provisorias, los cambios incompatibles hacia atrás son vistos como una «solución de último recurso» - se intentará todo para encontrar una solución compatible hacia atrás para los problemas identificados.

Este proceso permite que la biblioteca estándar continúe evolucionando con el tiempo, sin bloquearse por errores de diseño problemáticos por períodos extensos de tiempo. Vea :pep'241' para más detalles.

paquete provisorio Vea *provisional API*.

Python 3000 Apodo para la fecha de lanzamiento de Python 3.x (acuñada en un tiempo cuando llegar a la versión 3 era algo distante en el futuro.) También se lo abrevió como «Py3k».

Pythónico Una idea o pieza de código que sigue ajustadamente la convenciones idiomáticas comunes del lenguaje Python, en vez de implementar código usando conceptos comunes a otros lenguajes. Por ejemplo, una convención común en Python es hacer bucles sobre todos los elementos de un iterable con la sentencia `for`. Muchos otros lenguajes no tienen este tipo de construcción, así que los que no están familiarizados con Python podrían usar contadores numéricos:

```
for i in range(len(food)):
    print(food[i])
```

En contraste, un método Pythónico más limpio:

```
for piece in food:
    print(piece)
```

nombre calificado Un nombre con puntos mostrando la ruta desde el alcance global del módulo a la clase, función o método definido en dicho módulo, como se define en **PEP 3155**. Para las funciones o clases de más alto nivel, el nombre calificado es el igual al nombre del objeto:

```
>>> class C:
...     class D:
...         def meth(self):
...             pass
...
>>> C.__qualname__
'C'
>>> C.D.__qualname__
'C.D'
>>> C.D.meth.__qualname__
'C.D.meth'
```

Cuando es usado para referirse a los módulos, *nombre completamente calificado* significa la ruta con puntos completo al módulo, incluyendo cualquier paquete padre, por ejemplo, *email.mime.text*‘:

```
>>> import email.mime.text
>>> email.mime.text.__name__
'email.mime.text'
```

contador de referencias El número de referencias a un objeto. Cuando el contador de referencias de un objeto cae hasta cero, éste es desalojable. En conteo de referencias no suele ser visible en el código de Python, pero es un elemento clave para la implementación de *CPython*. El módulo `sys` define la `getrefcount()` que los programadores pueden emplear para retornar el conteo de referencias de un objeto en particular.

paquete regular Un *package* tradicional, como aquellos con un directorio conteniendo el archivo `__init__.py`.

Vea también *namespace package*.

__slots__ Es una declaración dentro de una clase que ahorra memoria pre declarando espacio para las atributos de la instancia y eliminando diccionarios de la instancia. Aunque es popular, esta técnica es algo dificultosa de lograr correctamente y es mejor reservarla para los casos raros en los que existen grandes cantidades de instancias en aplicaciones con uso crítico de memoria.

secuencia Un *iterable* que logra un acceso eficiente a los elementos usando índices enteros a través del método especial `__getitem__()` y que define un método `__len__()` que devuelve la longitud de la secuencia. Algunas de las secuencias incorporadas son `list`, `str`, `tuple`, y `bytes`. Observe que `dict` también soporta `__getitem__()` y `__len__()`, pero es considerada un mapeo más que una secuencia porque las búsquedas son por claves arbitraria *immutable* y no por enteros.

La clase base abstracta `collections.abc.Sequence` define una interfaz mucho más rica que va más allá de sólo `__getitem__()` y `__len__()`, agregando `count()`, `index()`, `__contains__()`, y `__reversed__()`. Los tipos que implementan esta interfaz expandida pueden ser registrados explícitamente usando `register()`.

despacho único Una forma de despacho de una *generic function* donde la implementación es elegida a partir del tipo de un sólo argumento.

rebanada Un objeto que contiene una porción de una *sequence*. Una rebanada es creada usando la notación de suscripto, `[]` con dos puntos entre los números cuando se ponen varios, como en `nombre_variable[1:3:5]`. La notación con corchete (suscrito) usa internamente objetos `slice`.

método especial Un método que es llamado implícitamente por Python cuando ejecuta ciertas operaciones en un tipo, como la adición. Estos métodos tienen nombres que comienzan y terminan con doble barra baja. Los métodos especiales están documentados en specialnames.

sentencia Una sentencia es parte de un conjunto (un «bloque» de código). Una sentencia tanto es una *expression* como alguna de las varias sintaxis usando una palabra clave, como `if`, `while` o `for`.

codificación de texto Un códec que codifica las cadenas Unicode a bytes.

archivo de texto Un *file object* capaz de leer y escribir objetos `str`. Frecuentemente, un archivo de texto también accede a un flujo de datos binario y maneja automáticamente el *text encoding*. Ejemplos de archivos de texto que son abiertos en modo texto (`'r'` o `'w'`), `sys.stdin`, `sys.stdout`, y las instancias de `io.StringIO`.

Vea también *binary file* por objeto de archivos capaces de leer y escribir *objeto tipo binario*.

cadena con triple comilla Una cadena que está enmarcada por tres instancias de comillas (») o apostrofes (“). Aunque no brindan ninguna funcionalidad que no está disponible usando cadenas con comillas simple, son útiles por varias razones. Permiten incluir comillas simples o dobles sin escapar dentro de las cadenas y pueden abarcar múltiples líneas sin el uso de caracteres de continuación, haciéndolas particularmente útiles para escribir docstrings.

tipo El tipo de un objeto Python determina qué tipo de objeto es; cada objeto tiene un tipo. El tipo de un objeto puede ser accedido por su atributo `__class__` o puede ser conseguido usando `type(obj)`.

alias de tipos Un sinónimo para un tipo, creado al asignar un tipo a un identificador.

Los alias de tipos son útiles para simplificar los *indicadores de tipo*. Por ejemplo:

```
from typing import List, Tuple

def remove_gray_shades(
        colors: List[Tuple[int, int, int]]) -> List[Tuple[int, int, int]]:
    pass
```

podría ser más legible así:

```
from typing import List, Tuple

Color = Tuple[int, int, int]

def remove_gray_shades(colors: List[Color]) -> List[Color]:
    pass
```

Vea `typing` y **PEP 484**, que describen esta funcionalidad.

indicador de tipo Una *annotation* que especifica el tipo esperado para una variable, un atributo de clase, un parámetro para una función o un valor de retorno.

Los indicadores de tipo son opcionales y no son obligados por Python pero son útiles para las herramientas de análisis de tipos estático, y ayuda a las IDE en el completado del código y la refactorización.

Los indicadores de tipo de las variables globales, atributos de clase, y funciones, no de variables locales, pueden ser accedidos usando `typing.get_type_hints()`.

Vea `typing` y **PEP 484**, que describen esta funcionalidad.

saltos de líneas universales Una manera de interpretar flujos de texto en la cual son reconocidos como finales de línea todas siguientes formas: la convención de Unix para fin de línea `'\n'`, la convención de Windows `'\r\n'`, y la vieja convención de Macintosh `'\r'`. Vea **PEP 278** y **PEP 3116**, además de:func:*bytes.splitlines* para usos adicionales.

anotación de variable Una *annotation* de una variable o un atributo de clase.

Cuando se anota una variable o un atributo de clase, la asignación es opcional:

```
class C:
    field: 'annotation'
```

Las anotaciones de variables son frecuentemente usadas para *type hints*: por ejemplo, se espera que esta variable tenga valores de clase `int`:

```
count: int = 0
```

La sintaxis de la anotación de variables está explicada en la sección annassign.

Vea *function annotation*, **PEP 484** y **PEP 526**, los cuales describen esta funcionalidad.

entorno virtual Un entorno cooperativamente aislado de ejecución que permite a los usuarios de Python y a las aplicaciones instalar y actualizar paquetes de distribución de Python sin interferir con el comportamiento de otras aplicaciones de Python en el mismo sistema.

Vea también `venv`.

máquina virtual Una computadora definida enteramente por software. La máquina virtual de Python ejecuta el *bytecode* generado por el compilador de bytecode.

Zen de Python Un listado de los principios de diseño y la filosofía de Python que son útiles para entender y usar el lenguaje. El listado puede encontrarse ingresando «`import this`» en la consola interactiva.

APÉNDICE B

Acerca de estos documentos

Estos documentos son generados por reStructuredText desarrollado por Sphinx, un procesador de documentos específicamente escrito para la documentación de Python.

El desarrollo de la documentación y su cadena de herramientas es un esfuerzo enteramente voluntario, al igual que Python. Si tu quieres contribuir, por favor revisa la página reporting-bugs para más información de cómo hacerlo. Los nuevos voluntarios son siempre bienvenidos!

Agradecemos a:

- Fred L. Drake, Jr., el creador original de la documentación del conjunto de herramientas de Python y escritor de gran parte del contenido;
- el proyecto Docutils para creación de reStructuredText y el juego de Utilidades de Documentación;
- Fredrik Lundh por su proyecto Referencia Alternativa de Python para la cual Sphinx tuvo muchas ideas.

B.1 Contribuidores de la documentación de Python

Muchas personas han contribuido para el lenguaje de Python, la librería estándar de Python, y la documentación de Python. Revisa Misc/ACKS la distribución de Python para una lista parcial de contribuidores.

Es solamente con la aportación y contribuciones de la comunidad de Python que Python tiene tan fantástica documentación – Muchas gracias!

APÉNDICE C

History and License

C.1 History of the software

Python was created in the early 1990s by Guido van Rossum at Stichting Mathematisch Centrum (CWI, see https://www.cwi.nl/) in the Netherlands as a successor of a language called ABC. Guido remains Python's principal author, although it includes many contributions from others.

In 1995, Guido continued his work on Python at the Corporation for National Research Initiatives (CNRI, see https://www.cnri.reston.va.us/) in Reston, Virginia where he released several versions of the software.

In May 2000, Guido and the Python core development team moved to BeOpen.com to form the BeOpen PythonLabs team. In October of the same year, the PythonLabs team moved to Digital Creations (now Zope Corporation; see https://www.zope.org/). In 2001, the Python Software Foundation (PSF, see https://www.python.org/psf/) was formed, a non-profit organization created specifically to own Python-related Intellectual Property. Zope Corporation is a sponsoring member of the PSF.

All Python releases are Open Source (see https://opensource.org/ for the Open Source Definition). Historically, most, but not all, Python releases have also been GPL-compatible; the table below summarizes the various releases.

Release	Derived from	Year	Owner	GPL compatible?
0.9.0 thru 1.2	n/a	1991-1995	CWI	yes
1.3 thru 1.5.2	1.2	1995-1999	CNRI	yes
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	no
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	yes
2.1.1	2.1+2.0.1	2001	PSF	yes
2.1.2	2.1.1	2002	PSF	yes
2.1.3	2.1.2	2002	PSF	yes
2.2 and above	2.1.1	2001-now	PSF	yes

Nota: GPL-compatible doesn't mean that we're distributing Python under the GPL. All Python licenses, unlike the GPL, let you distribute a modified version without making your changes open source. The GPL-compatible licenses make it possible to combine Python with other software that is released under the GPL; the others don't.

Thanks to the many outside volunteers who have worked under Guido's direction to make these releases possible.

C.2 Terms and conditions for accessing or otherwise using Python

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.7.17

```
1. This LICENSE AGREEMENT is between the Python Software Foundation␣
↪("PSF"), and
   the Individual or Organization ("Licensee") accessing and otherwise␣
↪using Python
   3.7.17 software in source or binary form and its associated␣
↪documentation.

2. Subject to the terms and conditions of this License Agreement, PSF␣
↪hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to␣
↪reproduce,
   analyze, test, perform and/or display publicly, prepare derivative␣
↪works,
   distribute, and otherwise use Python 3.7.17 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's␣
↪notice of
   copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All␣
↪Rights
   Reserved" are retained in Python 3.7.17 alone or in any derivative␣
↪version
   prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 3.7.17 or any part thereof, and wants to make the
   derivative work available to others as provided herein, then Licensee␣
↪hereby
   agrees to include in any such work a brief summary of the changes made␣
↪to Python
   3.7.17.

4. PSF is making Python 3.7.17 available to Licensee on an "AS IS" basis.
   PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY␣
↪OF
   EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY␣
↪REPRESENTATION OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR␣
↪THAT THE
   USE OF PYTHON 3.7.17 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.7.17
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A␣
↪RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.7.17, OR ANY␣
↪DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material␣
↪breach of
   its terms and conditions.
```

```
7. Nothing in this License Agreement shall be deemed to create any␣
↪relationship
   of agency, partnership, or joint venture between PSF and Licensee. ␣
↪This License
   Agreement does not grant permission to use PSF trademarks or trade name␣
↪in a
   trademark sense to endorse or promote products or services of Licensee,␣
↪or any
   third party.

8. By copying, installing or otherwise using Python 3.7.17, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at
   160 Saratoga Avenue, Santa Clara, CA 95051, and the Individual or Organization
   ("Licensee") accessing and otherwise using this software in source or binary
   form and its associated documentation ("the Software").

2. Subject to the terms and conditions of this BeOpen Python License Agreement,
   BeOpen hereby grants Licensee a non-exclusive, royalty-free, world-wide license
   to reproduce, analyze, test, perform and/or display publicly, prepare derivative
   works, distribute, and otherwise use the Software alone or in any derivative
   version, provided, however, that the BeOpen Python License is retained in the
   Software, alone or in any derivative version prepared by Licensee.

3. BeOpen is making the Software available to Licensee on an "AS IS" basis.
   BEOPEN MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, BEOPEN MAKES NO AND DISCLAIMS ANY REPRESENTATION OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE
   USE OF THE SOFTWARE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

4. BEOPEN SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF THE SOFTWARE FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF USING,
   MODIFYING OR DISTRIBUTING THE SOFTWARE, OR ANY DERIVATIVE THEREOF, EVEN IF
   ADVISED OF THE POSSIBILITY THEREOF.

5. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

6. This License Agreement shall be governed by and interpreted in all respects
   by the law of the State of California, excluding conflict of law provisions.
   Nothing in this License Agreement shall be deemed to create any relationship of
   agency, partnership, or joint venture between BeOpen and Licensee.  This License
   Agreement does not grant permission to use BeOpen trademarks or trade names in a
   trademark sense to endorse or promote products or services of Licensee, or any
   third party.  As an exception, the "BeOpen Python" logos available at
   http://www.pythonlabs.com/logos.html may be used according to the permissions
   granted on that web page.

7. By copying, installing or otherwise using the software, Licensee agrees to be
   bound by the terms and conditions of this License Agreement.
```

C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

```
1. This LICENSE AGREEMENT is between the Corporation for National Research
   Initiatives, having an office at 1895 Preston White Drive, Reston, VA 20191
   ("CNRI"), and the Individual or Organization ("Licensee") accessing and
   otherwise using Python 1.6.1 software in source or binary form and its
   associated documentation.

2. Subject to the terms and conditions of this License Agreement, CNRI hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 1.6.1 alone or in any derivative version,
   provided, however, that CNRI's License Agreement and CNRI's notice of copyright,
   i.e., "Copyright © 1995-2001 Corporation for National Research Initiatives; All
   Rights Reserved" are retained in Python 1.6.1 alone or in any derivative version
   prepared by Licensee.  Alternately, in lieu of CNRI's License Agreement,
   Licensee may substitute the following text (omitting the quotes): "Python 1.6.1
   is made available subject to the terms and conditions in CNRI's License
   Agreement.  This Agreement together with Python 1.6.1 may be located on the
   Internet using the following unique, persistent identifier (known as a handle):
   1895.22/1013.  This Agreement may also be obtained from a proxy server on the
   Internet using the following URL: http://hdl.handle.net/1895.22/1013."

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 1.6.1 or any part thereof, and wants to make the derivative
   work available to others as provided herein, then Licensee hereby agrees to
   include in any such work a brief summary of the changes made to Python 1.6.1.

4. CNRI is making Python 1.6.1 available to Licensee on an "AS IS" basis.  CNRI
   MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF EXAMPLE,
   BUT NOT LIMITATION, CNRI MAKES NO AND DISCLAIMS ANY REPRESENTATION OR WARRANTY
   OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF
   PYTHON 1.6.1 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.

5. CNRI SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 1.6.1 FOR
   ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 1.6.1, OR ANY DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of
   its terms and conditions.

7. This License Agreement shall be governed by the federal intellectual property
   law of the United States, including without limitation the federal copyright
   law, and, to the extent such U.S. federal law does not apply, by the law of the
   Commonwealth of Virginia, excluding Virginia's conflict of law provisions.
   Notwithstanding the foregoing, with regard to derivative works based on Python
   1.6.1 that incorporate non-separable material that was previously distributed
   under the GNU General Public License (GPL), the law of the Commonwealth of
   Virginia shall govern this License Agreement only as to issues arising under or
   with respect to Paragraphs 4, 5, and 7 of this License Agreement.  Nothing in
   this License Agreement shall be deemed to create any relationship of agency,
   partnership, or joint venture between CNRI and Licensee.  This License Agreement
   does not grant permission to use CNRI trademarks or trade name in a trademark
   sense to endorse or promote products or services of Licensee, or any third
   party.

8. By clicking on the "ACCEPT" button where indicated, or by copying, installing
   or otherwise using Python 1.6.1, Licensee agrees to be bound by the terms and
   conditions of this License Agreement.
```

C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

C.3 Licenses and Acknowledgements for Incorporated Software

This section is an incomplete, but growing list of licenses and acknowledgements for third-party software incorporated in the Python distribution.

C.3.1 Mersenne Twister

The `_random` module includes code based on a download from http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html. The following are the verbatim comments from the original code:

```
A C-program for MT19937, with initialization improved 2002/1/26.
Coded by Takuji Nishimura and Makoto Matsumoto.

Before using, initialize the state by using init_genrand(seed)
or init_by_array(init_key, key_length).

Copyright (C) 1997 - 2002, Makoto Matsumoto and Takuji Nishimura,
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

 1. Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.

 2. Redistributions in binary form must reproduce the above copyright
    notice, this list of conditions and the following disclaimer in the
    documentation and/or other materials provided with the distribution.

 3. The names of its contributors may not be used to endorse or promote
    products derived from this software without specific prior written
    permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
```

(continué en la próxima página)

(proviene de la página anterior)

```
A PARTICULAR PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.


Any feedback is very welcome.
http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/emt.html
email: m-mat @ math.sci.hiroshima-u.ac.jp (remove space)
```

C.3.2 Sockets

The `socket` module uses the functions, `getaddrinfo()`, and `getnameinfo()`, which are coded in separate source files from the WIDE Project, http://www.wide.ad.jp/.

```
Copyright (C) 1995, 1996, 1997, and 1998 WIDE Project.
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. Neither the name of the project nor the names of its contributors
   may be used to endorse or promote products derived from this software
   without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE PROJECT AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE PROJECT OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.3 Asynchronous socket services

The `asynchat` and `asyncore` modules contain the following notice:

```
Copyright 1996 by Sam Rushing

                        All Rights Reserved

Permission to use, copy, modify, and distribute this software and
its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
```

(continué en la próxima página)

(proviene de la página anterior)

```
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of Sam
Rushing not be used in advertising or publicity pertaining to
distribution of the software without specific, written prior
permission.

SAM RUSHING DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE,
INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN
NO EVENT SHALL SAM RUSHING BE LIABLE FOR ANY SPECIAL, INDIRECT OR
CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS
OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT,
NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN
CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.
```

C.3.4 Cookie management

The `http.cookies` module contains the following notice:

```
Copyright 2000 by Timothy O'Malley <timo@alum.mit.edu>

               All Rights Reserved

Permission to use, copy, modify, and distribute this software
and its documentation for any purpose and without fee is hereby
granted, provided that the above copyright notice appear in all
copies and that both that copyright notice and this permission
notice appear in supporting documentation, and that the name of
Timothy O'Malley  not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

Timothy O'Malley DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS
SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY
AND FITNESS, IN NO EVENT SHALL Timothy O'Malley BE LIABLE FOR
ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3.5 Execution tracing

The `trace` module contains the following notice:

```
portions copyright 2001, Autonomous Zones Industries, Inc., all rights...
err...  reserved and offered to the public under the terms of the
Python 2.2 license.
Author: Zooko O'Whielacronx
http://zooko.com/
mailto:zooko@zooko.com

Copyright 2000, Mojam Media, Inc., all rights reserved.
Author: Skip Montanaro

Copyright 1999, Bioreason, Inc., all rights reserved.
Author: Andrew Dalke

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
```

(continué en la próxima página)

(proviene de la página anterior)

```
Author: Skip Montanaro

Copyright 1991-1995, Stichting Mathematisch Centrum, all rights reserved.


Permission to use, copy, modify, and distribute this Python software and
its associated documentation for any purpose without fee is hereby
granted, provided that the above copyright notice appears in all copies,
and that both that copyright notice and this permission notice appear in
supporting documentation, and that the name of neither Automatrix,
Bioreason or Mojam Media be used in advertising or publicity pertaining to
distribution of the software without specific, written prior permission.
```

C.3.6 UUencode and UUdecode functions

The `uu` module contains the following notice:

```
Copyright 1994 by Lance Ellinghouse
Cathedral City, California Republic, United States of America.
                       All Rights Reserved
Permission to use, copy, modify, and distribute this software and its
documentation for any purpose and without fee is hereby granted,
provided that the above copyright notice appear in all copies and that
both that copyright notice and this permission notice appear in
supporting documentation, and that the name of Lance Ellinghouse
not be used in advertising or publicity pertaining to distribution
of the software without specific, written prior permission.
LANCE ELLINGHOUSE DISCLAIMS ALL WARRANTIES WITH REGARD TO
THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS, IN NO EVENT SHALL LANCE ELLINGHOUSE CENTRUM BE LIABLE
FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES
WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN
ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT
OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Modified by Jack Jansen, CWI, July 1995:
- Use binascii module to do the actual line-by-line conversion
  between ascii and binary. This results in a 1000-fold speedup. The C
  version is still 5 times faster, though.
- Arguments more compliant with Python standard
```

C.3.7 XML Remote Procedure Calls

The `xmlrpc.client` module contains the following notice:

```
    The XML-RPC client interface is

Copyright (c) 1999-2002 by Secret Labs AB
Copyright (c) 1999-2002 by Fredrik Lundh

By obtaining, using, and/or copying this software and/or its
associated documentation, you agree that you have read, understood,
and will comply with the following terms and conditions:

Permission to use, copy, modify, and distribute this software and
its associated documentation for any purpose and without fee is
hereby granted, provided that the above copyright notice appears in
all copies, and that both that copyright notice and this permission
```

(continué en la próxima página)

(proviene de la página anterior)

```
notice appear in supporting documentation, and that the name of
Secret Labs AB or the author not be used in advertising or publicity
pertaining to distribution of the software without specific, written
prior permission.

SECRET LABS AB AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD
TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANT-
ABILITY AND FITNESS.  IN NO EVENT SHALL SECRET LABS AB OR THE AUTHOR
BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

C.3.8 test_epoll

The `test_epoll` module contains the following notice:

```
Copyright (c) 2001-2006 Twisted Matrix Laboratories.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be
included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE
LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION
OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION
WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

C.3.9 Select kqueue

The `select` module contains the following notice for the kqueue interface:

```
Copyright (c) 2000 Doug White, 2006 James Knight, 2007 Christian Heimes
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS ``AS IS'' AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
```

(continué en la próxima página)

(proviene de la página anterior)

```
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.10 SipHash24

The file `Python/pyhash.c` contains Marek Majkowski" implementation of Dan Bernstein's SipHash24 algorithm. It contains the following note:

```
<MIT License>
Copyright (c) 2013  Marek Majkowski <marek@popcount.org>

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in
all copies or substantial portions of the Software.
</MIT License>

Original location:
   https://github.com/majek/csiphash/

Solution inspired by code from:
   Samuel Neves (supercop/crypto_auth/siphash24/little)
   djb (supercop/crypto_auth/siphash24/little2)
   Jean-Philippe Aumasson (https://131002.net/siphash/siphash24.c)
```

C.3.11 strtod and dtoa

The file `Python/dtoa.c`, which supplies C functions dtoa and strtod for conversion of C doubles to and from strings, is derived from the file of the same name by David M. Gay, currently available from http://www.netlib.org/fp/. The original file, as retrieved on March 16, 2009, contains the following copyright and licensing notice:

```
/****************************************************************
 *
 * The author of this software is David M. Gay.
 *
 * Copyright (c) 1991, 2000, 2001 by Lucent Technologies.
 *
 * Permission to use, copy, modify, and distribute this software for any
 * purpose without fee is hereby granted, provided that this entire notice
 * is included in all copies of any software which is or includes a copy
 * or modification of this software and in all copies of the supporting
 * documentation for such software.
 *
 * THIS SOFTWARE IS BEING PROVIDED "AS IS", WITHOUT ANY EXPRESS OR IMPLIED
 * WARRANTY.  IN PARTICULAR, NEITHER THE AUTHOR NOR LUCENT MAKES ANY
 * REPRESENTATION OR WARRANTY OF ANY KIND CONCERNING THE MERCHANTABILITY
```

(continué en la próxima página)

(proviene de la página anterior)

```
* OF THIS SOFTWARE OR ITS FITNESS FOR ANY PARTICULAR PURPOSE.
*
****************************************************************/
```

C.3.12 OpenSSL

The modules `hashlib`, `posix`, `ssl`, `crypt` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and Mac OS X installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here:

```
LICENSE ISSUES
==============

The OpenSSL toolkit stays under a dual license, i.e. both the conditions of
the OpenSSL License and the original SSLeay license apply to the toolkit.
See below for the actual license texts. Actually both licenses are BSD-style
Open Source licenses. In case of any license issues related to OpenSSL
please contact openssl-core@openssl.org.

OpenSSL License
---------------

  /* ====================================================================
   * Copyright (c) 1998-2008 The OpenSSL Project.  All rights reserved.
   *
   * Redistribution and use in source and binary forms, with or without
   * modification, are permitted provided that the following conditions
   * are met:
   *
   * 1. Redistributions of source code must retain the above copyright
   *    notice, this list of conditions and the following disclaimer.
   *
   * 2. Redistributions in binary form must reproduce the above copyright
   *    notice, this list of conditions and the following disclaimer in
   *    the documentation and/or other materials provided with the
   *    distribution.
   *
   * 3. All advertising materials mentioning features or use of this
   *    software must display the following acknowledgment:
   *    "This product includes software developed by the OpenSSL Project
   *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
   *
   * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
   *    endorse or promote products derived from this software without
   *    prior written permission. For written permission, please contact
   *    openssl-core@openssl.org.
   *
   * 5. Products derived from this software may not be called "OpenSSL"
   *    nor may "OpenSSL" appear in their names without prior written
   *    permission of the OpenSSL Project.
   *
   * 6. Redistributions of any form whatsoever must retain the following
   *    acknowledgment:
   *    "This product includes software developed by the OpenSSL Project
   *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
   *
   * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT ``AS IS'' AND ANY
   * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
   * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
```

(continué en la próxima página)

(proviene de la página anterior)

```
    * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
    * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
    * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
    * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
    * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
    * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
    * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
    * OF THE POSSIBILITY OF SUCH DAMAGE.
    * ====================================================================
    *
    * This product includes cryptographic software written by Eric Young
    * (eay@cryptsoft.com).  This product includes software written by Tim
    * Hudson (tjh@cryptsoft.com).
    *
    */

Original SSLeay License
-----------------------

   /* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
    * All rights reserved.
    *
    * This package is an SSL implementation written
    * by Eric Young (eay@cryptsoft.com).
    * The implementation was written so as to conform with Netscapes SSL.
    *
    * This library is free for commercial and non-commercial use as long as
    * the following conditions are aheared to.  The following conditions
    * apply to all code found in this distribution, be it the RC4, RSA,
    * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
    * included with this distribution is covered by the same copyright terms
    * except that the holder is Tim Hudson (tjh@cryptsoft.com).
    *
    * Copyright remains Eric Young's, and as such any Copyright notices in
    * the code are not to be removed.
    * If this package is used in a product, Eric Young should be given attribution
    * as the author of the parts of the library used.
    * This can be in the form of a textual message at program startup or
    * in documentation (online or textual) provided with the package.
    *
    * Redistribution and use in source and binary forms, with or without
    * modification, are permitted provided that the following conditions
    * are met:
    * 1. Redistributions of source code must retain the copyright
    *    notice, this list of conditions and the following disclaimer.
    * 2. Redistributions in binary form must reproduce the above copyright
    *    notice, this list of conditions and the following disclaimer in the
    *    documentation and/or other materials provided with the distribution.
    * 3. All advertising materials mentioning features or use of this software
    *    must display the following acknowledgement:
    *    "This product includes cryptographic software written by
    *     Eric Young (eay@cryptsoft.com)"
    *    The word 'cryptographic' can be left out if the rouines from the library
    *    being used are not cryptographic related :-).
    * 4. If you include any Windows specific code (or a derivative thereof) from
    *    the apps directory (application code) you must include an acknowledgement:
    *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
    *
    * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
    * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
```

(continué en la próxima página)

(proviene de la página anterior)

```
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

C.3.13 expat

The `pyexpat` extension is built using an included copy of the expat sources unless the build is configured `--with-system-expat`:

```
Copyright (c) 1998, 1999, 2000 Thai Open Source Software Center Ltd
                               and Clark Cooper

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

C.3.14 libffi

The `_ctypes` extension is built using an included copy of the libffi sources unless the build is configured `--with-system-libffi`:

```
Copyright (c) 1996-2008  Red Hat, Inc and others.

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
``Software''), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:
```

(continué en la próxima página)

(proviene de la página anterior)

```
The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED ``AS IS'', WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
NONINFRINGEMENT.  IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY,
WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
DEALINGS IN THE SOFTWARE.
```

C.3.15 zlib

The `zlib` extension is built using an included copy of the zlib sources if the zlib version found on the system is too old to be used for the build:

```
Copyright (C) 1995-2011 Jean-loup Gailly and Mark Adler

This software is provided 'as-is', without any express or implied
warranty.  In no event will the authors be held liable for any damages
arising from the use of this software.

Permission is granted to anyone to use this software for any purpose,
including commercial applications, and to alter it and redistribute it
freely, subject to the following restrictions:

1. The origin of this software must not be misrepresented; you must not
   claim that you wrote the original software. If you use this software
   in a product, an acknowledgment in the product documentation would be
   appreciated but is not required.

2. Altered source versions must be plainly marked as such, and must not be
   misrepresented as being the original software.

3. This notice may not be removed or altered from any source distribution.

Jean-loup Gailly        Mark Adler
jloup@gzip.org          madler@alumni.caltech.edu
```

C.3.16 cfuhash

The implementation of the hash table used by the `tracemalloc` is based on the cfuhash project:

```
Copyright (c) 2005 Don Owens
All rights reserved.

This code is released under the BSD license:

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

  * Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
```

(continué en la próxima página)

(proviene de la página anterior)

```
  * Redistributions in binary form must reproduce the above
    copyright notice, this list of conditions and the following
    disclaimer in the documentation and/or other materials provided
    with the distribution.

  * Neither the name of the author nor the names of its
    contributors may be used to endorse or promote products derived
    from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS
FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR
SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

C.3.17 libmpdec

The `_decimal` module is built using an included copy of the libmpdec library unless the build is configured `--with-system-libmpdec`:

```
Copyright (c) 2008-2016 Stefan Krah. All rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:

1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR AND CONTRIBUTORS "AS IS" AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

APÉNDICE D

Copyright

See *History and License* for complete license and permissions information.

Índice

No alfabético

A

B

C