
Python Setup and Usage

Versión 3.14.0rc2

Guido van Rossum and the Python development team

septiembre 01, 2025

**Python Software Foundation
Email: docs@python.org**

1	Línea de comandos y entorno	3
1.1	Línea de comando	3
1.1.1	Opciones de interfaz	3
1.1.2	Opciones genéricas	5
1.1.3	Opciones diversas	6
1.1.4	Controlling color	11
1.2	Variables de entorno	11
1.2.1	Variables de modo de depuración	18
2	Uso de Python en plataformas Unix	21
2.1	Obteniendo e instalando la última versión de Python	21
2.1.1	En Linux	21
2.1.2	En FreeBSD y OpenBSD	22
2.2	Construyendo Python	22
2.3	Rutas y archivos relacionados con Python	22
2.4	Miscelánea	23
2.5	OpenSSL personalizado	23
3	Configurar Python	25
3.1	Requisitos de compilación	25
3.2	Archivos generados	26
3.2.1	configure script	26
3.3	Configurar opciones	26
3.3.1	Opciones generales	26
3.3.2	C compiler options	29
3.3.3	Opciones del enlazador	30
3.3.4	Options for third-party dependencies	30
3.3.5	Opciones de WebAssembly	31
3.3.6	Opciones de instalación	31
3.3.7	Opciones de desempeño	32
3.3.8	Compilación de depuración de Python	34
3.3.9	Opciones de depuración	35
3.3.10	Opciones del enlazador	36
3.3.11	Opciones de bibliotecas	36
3.3.12	Opciones de seguridad	37
3.3.13	Opciones macOS	38
3.3.14	iOS Options	39
3.3.15	Opciones de compilación cruzada	39
3.4	Sistema de compilación Python	40
3.4.1	Archivos principales del sistema de compilación	40
3.4.2	Pasos principales de compilación	40

3.4.3	Objetivos principales de Makefile	40
3.4.4	Extensiones C	41
3.5	Banderas de compilador y vinculación	42
3.5.1	Banderas del preprocesador	42
3.5.2	Banderas del compilador	42
3.5.3	Banderas de vinculación	44
4	Uso de Python en Windows	47
4.1	Python Install Manager	47
4.1.1	Installation	47
4.1.2	Basic Use	48
4.1.3	Command Help	49
4.1.4	Listing Runtimes	49
4.1.5	Installing Runtimes	50
4.1.6	Offline Installs	50
4.1.7	Uninstalling Runtimes	50
4.1.8	Configuration	51
4.1.9	Shebang lines	52
4.1.10	Advanced Installation	52
4.1.11	Administrative Configuration	54
4.1.12	Installing Free-threaded Binaries	54
4.1.13	Troubleshooting	55
4.2	El paquete incrustable	56
4.2.1	Aplicación Python	57
4.2.2	Incrustar Python	57
4.3	El paquete de nuget.org	57
4.3.1	Free-threaded packages	58
4.4	Distribuciones alternativas	58
4.5	Supported Windows versions	59
4.6	Quitar el límite de MAX_PATH	59
4.7	Modo UTF-8	59
4.8	Encontrar módulos	60
4.9	Módulos adicionales	61
4.9.1	PyWin32	61
4.9.2	cx_Freeze	61
4.10	Compilar Python en Windows	61
4.11	The full installer (deprecated)	62
4.11.1	Pasos para la instalación	62
4.11.2	Quitar el límite de MAX_PATH	63
4.11.3	Instalación sin interfaz de usuario	63
4.11.4	Instalación sin descargas	65
4.11.5	Modificar una instalación	65
4.11.6	Installing Free-threaded Binaries	66
4.12	Python Launcher for Windows (Deprecated)	66
4.12.1	Comenzar	67
4.12.2	Líneas shebang	68
4.12.3	Argumentos en líneas shebang	69
4.12.4	Personalización	70
4.12.5	Diagnóstico	71
4.12.6	Ejecución en seco	71
4.12.7	Instalación bajo demanda	71
4.12.8	Códigos de retorno	71
5	Using Python on macOS	73
5.1	Using Python for macOS from <code>python.org</code>	73
5.1.1	Installation steps	73
5.1.2	Cómo ejecutar un <i>script</i> de Python	81
5.2	Alternative Distributions	81

5.3	Instalación de paquetes adicionales de Python	81
5.4	GUI Programming	81
5.5	Advanced Topics	82
5.5.1	Installing Free-threaded Binaries	82
5.5.2	Installing using the command line	83
5.5.3	Distributing Python Applications	85
5.5.4	App Store Compliance	85
5.6	Otros recursos	85
6	Using Python on Android	87
6.1	Adding Python to an Android app	87
6.2	Building a Python package for Android	88
7	Using Python on iOS	89
7.1	Python at runtime on iOS	89
7.1.1	iOS version compatibility	89
7.1.2	Platform identification	89
7.1.3	Standard library availability	90
7.1.4	Binary extension modules	90
7.1.5	Compiler stub binaries	90
7.2	Installing Python on iOS	91
7.2.1	Tools for building iOS apps	91
7.2.2	Adding Python to an iOS project	91
7.2.3	Testing a Python package	94
7.3	App Store Compliance	94
8	Editores e IDEs	95
8.1	IDLE — Python editor and shell	95
8.2	Other Editors and IDEs	95
A	Glosario	97
B	About this documentation	117
B.1	Contributors to the Python documentation	117
C	Historia y Licencia	119
C.1	Historia del software	119
C.2	Términos y condiciones para acceder o usar Python	120
C.2.1	PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2	120
C.2.2	ACUERDO DE LICENCIA DE BEOPEN.COM PARA PYTHON 2.0	121
C.2.3	ACUERDO DE LICENCIA CNRI PARA PYTHON 1.6.1	122
C.2.4	ACUERDO DE LICENCIA CWI PARA PYTHON 0.9.0 HASTA 1.2	123
C.2.5	ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION	123
C.3	Licencias y reconocimientos para software incorporado	123
C.3.1	Mersenne Twister	123
C.3.2	Sockets	124
C.3.3	Servicios de socket asincrónicos	125
C.3.4	Gestión de cookies	125
C.3.5	Seguimiento de ejecución	126
C.3.6	funciones UUencode y UUdecode	126
C.3.7	Llamadas a procedimientos remotos XML	127
C.3.8	test_epoll	127
C.3.9	Seleccionar kqueue	128
C.3.10	SipHash24	128
C.3.11	strtod y dtoa	129
C.3.12	OpenSSL	129
C.3.13	expat	133
C.3.14	libffi	133
C.3.15	zlib	134

C.3.16	cfuhash	134
C.3.17	libmpdec	135
C.3.18	Conjunto de pruebas W3C C14N	135
C.3.19	mimalloc	136
C.3.20	asyncio	136
C.3.21	Global Unbounded Sequences (GUS)	137
C.3.22	Zstandard bindings	137
D	Derechos de autor	139
	Índice	141

Esta parte de la documentación está dedicada a información general sobre la configuración del entorno Python en diferentes plataformas, la invocación del intérprete y cosas que facilitan el trabajo con Python.

Línea de comandos y entorno

El intérprete de CPython analiza la línea de comandos y el entorno en busca de varias configuraciones.

Los esquemas de línea de comandos de otras implementaciones pueden diferir. Véase [implementations](#) para obtener más recursos.

1.1 Línea de comando

Al invocar Python, puede especificar cualquiera de estas opciones:

```
python [-bBdEhiIOpQrSuvVWx?] [-c command | -m module-name | script | - ] [args]
```

El caso de uso más común es, por supuesto, una simple invocación de un script:

```
python myscript.py
```

1.1.1 Opciones de interfaz

La interfaz del intérprete es similar a la del shell UNIX, pero proporciona algunos métodos adicionales de invocación:

- 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. For more on interactive mode, see [tut-interac](#).
- Cuando se llama con un argumento de nombre de archivo o con un archivo como entrada estándar, lee y ejecuta un script de ese archivo.
- Cuando se llama con un argumento de nombre de directorio, lee y ejecuta un script con el nombre adecuado desde ese directorio.
- Cuando se llama con `-c comando`, ejecuta las instrucciones de Python dadas como *command*. Aquí *comando* puede contener varias instrucciones separadas por nuevas líneas. ¡El espacio en blanco principal es significativo en las instrucciones de Python!
- Cuando se llama con `-m module-name`, el módulo dado se encuentra en la ruta del módulo Python y se ejecuta como un script.

En el modo no interactivo, toda la entrada se analiza antes de ejecutarse.

Una opción de interfaz termina la lista de opciones consumidas por el intérprete, todos los argumentos consecutivos terminarán en `sys.argv` – tenga en cuenta que el primer elemento, subíndice cero (`sys.argv[0]`), es una cadena que refleja el origen del programa.

-c <command>

Ejecute el código de Python en *comando*. *comando* puede ser una o más sentencias separadas por nuevas líneas, con espacio en blanco inicial significativo como en el código normal del módulo.

Si se proporciona esta opción, el primer elemento de `sys.argv` será `"-c"` y el directorio actual se agregará al inicio de `sys.path` (permitiendo que los módulos de ese directorio se importen como módulos de nivel superior).

Lanza un auditing event `cpython.run_command` con el argumento `command`.

Distinto en la versión 3.14: *command* is automatically dedented before execution.

-m <module-name>

Busque `sys.path` para el módulo con nombre y ejecute su contenido como el módulo `__main__`.

Dado que el argumento es un nombre *módulo*, no debe dar una extensión de archivo (`.py`). El nombre del módulo debe ser un nombre de módulo Python absoluto válido, pero es posible que la implementación no siempre lo aplique (por ejemplo, puede permitirle usar un nombre que incluya un guión).

También se permiten los nombres de paquetes (incluidos los paquetes de espacio de nombres). Cuando se proporciona un nombre de paquete en lugar de un módulo normal, el intérprete ejecutará `<pkg>.__main__` como módulo principal. Este comportamiento es deliberadamente similar al manejo de directorios y archivos zip que se pasan al intérprete como argumento del script.

Nota

Esta opción no se puede utilizar con módulos integrados y módulos de extensión escritos en C, ya que no tienen archivos de módulo Python. Sin embargo, todavía se puede utilizar para módulos precompilados, incluso si el archivo de origen original no está disponible.

Si se da esta opción, el primer elemento de `sys.argv` será la ruta de acceso completa al archivo de módulo (mientras se encuentra el archivo de módulo, el primer elemento se establecerá en `"-m"`). Al igual que con la opción `-c`, el directorio actual se agregará al inicio de `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.

Muchos módulos de biblioteca estándar contienen código que se invoca en su ejecución como script. Un ejemplo es el módulo `timeit`:

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

Retorna un auditing event `cpython.run_module` con el argumento `nombre-módulo`.

Ver también

runpy.run_module()

Funcionalidad equivalente directamente disponible para el código Python

PEP 338 – Ejecución de módulos como scripts

Distinto en la versión 3.1: Proporcione el nombre del paquete para ejecutar un submódulo `__main__`.

Distinto en la versión 3.4: paquetes de espacio de nombres también son compatibles

-

Leer comandos de entrada estándar (`sys.stdin`). Si la entrada estándar es un terminal, `-i` está implícita.

Si se da esta opción, el primer elemento de `sys.argv` será "-" y el directorio actual se agregará al inicio de `sys.path`.

Lanza un evento auditing event `cpython.run_stdin` sin argumentos.

<script>

Ejecute el código Python contenido en *script*, que debe ser una ruta de acceso del sistema de archivos (absoluta o relativa) que haga referencia a un archivo Python, un directorio que contenga un archivo `__main__.py` o un archivo zip que contenga un archivo `__main__.py`.

Si se proporciona esta opción, el primer elemento de `sys.argv` será el nombre del script como se indica en la línea de comandos.

Si el nombre del script hace referencia directamente a un archivo Python, el directorio que contiene ese archivo se agrega al inicio de `sys.path`, y el archivo se ejecuta como el módulo `__main__`.

Si el nombre del script hace referencia a un directorio o zipfile, el nombre del script se agrega al inicio de `sys.path` y el archivo `__main__.py` en esa ubicación se ejecuta como el módulo `__main__`.

`-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.

Lanza un auditing event `cpython.run_file` con el argumento `filename`.

Ver también

`runpy.run_path()`

Funcionalidad equivalente directamente disponible para el código Python

Si no se da ninguna opción de interfaz, `-i` está implícita, `sys.argv[0]` es una cadena vacía ("") y el directorio actual se agregará al inicio de `sys.path`. Además, la finalización de tabulación y la edición del historial se habilitan automáticamente, si están disponibles en su plataforma (consulte `rlcompleter-config`).

Ver también

tut-invoking

Distinto en la versión 3.4: Habilitación automática de la finalización de pestañas y edición del historial.

1.1.2 Opciones genéricas

`-?`

`-h`

`--help`

Imprime una breve descripción de todas las opciones de la línea de comandos.

`--help-env`

Imprima una breve descripción de las variables de entorno específicas de Python y salga.

Added in version 3.11.

`--help-xoptions`

Imprima una descripción de las opciones `-X` específicas de la implementación y salga.

Added in version 3.11.

--help-all

Imprima información completa de uso y salga.

Added in version 3.11.

-v

--version

Imprima el número de versión de Python y salga. Ejemplo de salida podría ser:

```
Python 3.8.0b2+
```

Cuando se le dé dos veces, imprima más información sobre la compilación, como:

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

Added in version 3.6: La opción -vv.

1.1.3 Opciones diversas

-b

Issue a warning when converting `bytes` or `bytearray` to `str` without specifying encoding or 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 also comparisons of `bytes` with `int`.

-B

Si se da, Python no intentará escribir archivos `.pyc` en la importación de módulos de origen. Véase también `PYTHONDONTWRITEBYTECODE`.

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

Controle el comportamiento de validación de los archivos `.pyc` basados en hash. Véase `pyc-invalidation`. Cuando se establece en `default`, los archivos de caché de código de bytes basados en hash marcados y desmarcados se validan según su semántica predeterminada. Cuando se establece en `always`, todos los archivos basados en hash `.pyc`, ya estén marcados o desmarcados, se validan con su archivo de origen correspondiente. Cuando se establece en `never`, los archivos basados en hash `.pyc` no se validan con sus archivos de origen correspondientes.

Esta opción no afecta a la semántica de los archivos `.pyc` basados en la marca de tiempo.

-d

Activa la salida de depuración (solo para expertos, dependiendo de las opciones de compilación). Véase también la variable de ambiente `PYTHONDEBUG`.

Esta opción necesita una *compilación de depuración de Python*, de lo contrario será ignorado.

-E

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

Véase también las opciones `-P` e `-I` (aisladas).

-i

Enter interactive mode after execution.

Using the `-i` option will enter interactive mode in any of the following circumstances:

- When a script is passed as first argument
- When the `-c` option is used
- When the `-m` option is used

Interactive mode will start even when `sys.stdin` does not appear to be a terminal. The `PYTHONSTARTUP` file is not read.

Esto puede ser útil para inspeccionar variables globales o un seguimiento de pila cuando un script lanza una excepción. Véase también `PYTHONINSPECT`.

-I

Ejecute Python en modo aislado. Esto también implica las opciones `-E`, `-P` y `-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.

Added in version 3.4.

-O

Quite las instrucciones `assert` y cualquier código condicionado al valor de `__debug__`. Aumente el nombre de archivo para los archivos compilados (*bytecode*) agregando `.opt-1` antes de la extensión `.pyc` (consulte **PEP 488**). Véase también `PYTHONOPTIMIZE`.

Distinto en la versión 3.5: Modifique los nombres de archivo `.pyc` según **PEP 488**.

-OO

Haga `-O` y también deseche las docstrings. Aumente el nombre de archivo para los archivos compilados (*bytecode*) agregando `.opt-2` antes de la extensión `.pyc` (consulte **PEP 488**).

Distinto en la versión 3.5: Modifique los nombres de archivo `.pyc` según **PEP 488**.

-P

No anteponga una ruta potencialmente insegura a `sys.path`:

- `python -m module` command line: No anteponga el directorio de trabajo actual.
- `python script.py` command line: No anteponga el directorio del script. Si es un enlace simbólico, resuelva los enlaces simbólicos.
- `python -c code` y `python (REPL)` command lines: No anteponga una cadena vacía, lo que significa el directorio de trabajo actual.

Consulte también la variable de entorno `PYTHONSAFEPATH` y las opciones `-E` y `-I` (aisladas).

Added in version 3.11.

-q

No muestres los mensajes de copyright y versión incluso en modo interactivo.

Added in version 3.2.

-R

Active la aleatorización de hash. Esta opción solo tiene efecto si la variable de entorno `PYTHONHASHSEED` está establecida en 0, ya que la aleatorización de hash está habilitada de forma predeterminada.

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

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

`PYTHONHASHSEED` le permite establecer un valor fijo para el secreto de inicialización hash.


Added in version 3.2.3.

Distinto en la versión 3.7: La opción ya no se omite.

-s

No agregue el `user site-packages` directory a `sys.path`.

See also [PYTHONNOUSERSITE](#).

 **Ver también**

PEP 370 – Directorio de paquetes de sitio por usuario

-S

Deshabilite la importación del módulo `site` y las manipulaciones dependientes del sitio de `sys.path` que conlleva. También deshabilite estas manipulaciones si `site` se importa explícitamente más tarde (llame a `site.main()` si desea que se activen).

-u

Forzar que las corrientes `stdout` y `stderr` no estén en búfer. Esta opción no tiene ningún efecto en la secuencia `stdin`.

Véase también [PYTHONUNBUFFERED](#).

Distinto en la versión 3.7: La capa de texto de las secuencias `stdout` y `stderr` ahora no está en búfer.

-v

Imprime un mensaje cada vez que se inicialice un módulo, mostrando el lugar (nombre de archivo o módulo integrado) desde el que se carga. Cuando se le pasa dos veces (`-vv`), imprime un mensaje para cada archivo que se comprueba al buscar un módulo. También proporciona información sobre la limpieza del módulo en la salida.

Distinto en la versión 3.10: El módulo `site` informa las rutas específicas del sitio y los archivos `.pth` que se están procesando.

Véase también [PYTHONVERBOSE](#).

-W arg

Control de advertencia. La maquinaria de advertencia de Python por defecto imprime mensajes de advertencia en `sys.stderr`.

La configuración más sencilla aplica una acción determinada incondicionalmente a todas las advertencias emitidas por un proceso (incluso aquellas que de otro modo se ignoran de forma predeterminada):

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

Los nombres de acción se pueden abreviar como se desee y el intérprete los resolverá con el nombre de acción adecuado. Por ejemplo, `-Wi` es lo mismo que `-Wignore`.

La forma completa del argumento es:

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

Los campos vacíos cuadran con todos los valores; los campos vacíos finales pueden omitirse. Por ejemplo, `-W ignore::DeprecationWarning` ignora todas las advertencias de `DeprecationWarning`.

El campo `action` es como se explicó anteriormente, pero solo se aplica a las advertencias que coinciden con los campos restantes.

El campo `message` debe coincidir con el mensaje de advertencia completo; esta coincidencia no distingue entre mayúsculas y minúsculas.

El campo *category* coincide con la categoría de advertencia (por ej. `DeprecationWarning`). Debe ser un nombre de clase; la coincidencia prueba si la categoría de advertencia real del mensaje es una subclase de la categoría de advertencia especificada.

El campo *module* coincide con el nombre de dominio completo del módulo; esta coincidencia distingue entre mayúsculas y minúsculas.

El campo *lineno* coincide con el número de línea, donde cero corresponde a todos los números de línea y, por lo tanto, es equivalente a número de línea omitido.

Se pueden dar varias opciones *-W*; cuando una advertencia coincide con más de una opción, se realiza la acción para la última opción de coincidencia. Las opciones inválidas *-W* son ignoradas (aunque se imprime un mensaje de advertencia sobre opciones no válidas cuando se emite la primera advertencia).

Las advertencias también se pueden controlar utilizando la variable de entorno `PYTHONWARNINGS` y desde un programa Python utilizando el módulo `warnings`. Por ejemplo, la función `warnings.filterwarnings()` puede ser usada con una expresión regular en el mensaje de advertencia.

Consulte `warning-filter` y `describing-warning-filters` para obtener más detalles.

-x

Omita la primera línea de la fuente, permitiendo el uso de formas que no sean de Unix de `#!cmd`. Esto está destinado a un hackeo específico de DOS solamente.

-X

Reservado para varias opciones específicas de la implementación. CPython define actualmente los siguientes valores posibles:

- `-X faulthandler` para habilitar `faulthandler`. Véase `PYTHONFAULTHANDLER` para obtener más información.

Added in version 3.3.

- `-X showrefcount` para generar el recuento total de referencias y el número de bloques de memoria utilizados cuando finalice el programa o después de cada instrucción en el intérprete interactivo. Esto sólo funciona en *compilaciones de depuración*.

Added in version 3.4.

- `-X tracemalloc` para iniciar el seguimiento de las asignaciones de memoria de Python mediante el módulo `tracemalloc`. De forma predeterminada, solo el marco más reciente se almacena en un seguimiento de un seguimiento. Utilice `-X tracemalloc=NFRAME` para iniciar el seguimiento con un límite de rastreo de marcos *NFRAME*. Consulte el `tracemalloc.start()` y `PYTHONTRACEMALLOC` para obtener más información.

Added in version 3.4.

- `-X int_max_str_digits` configura la limitación de longitud de conversión de cadena de tipo entero. Véase también `PYTHONINTMAXSTRDIGITS`.

Added in version 3.11.

- `-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 `python -X importtime -c 'import asyncio'`.

`-X importtime=2` enables additional output that indicates when an imported module has already been loaded. In such cases, the string `cached` will be printed in both time columns.

See also `PYTHONPROFILEIMPORTTIME`.

Added in version 3.7.

Distinto en la versión 3.14: Added `-X importtime=2` to also trace imports of loaded modules, and reserved values other than 1 and 2 for future use.

- `-X dev`: enable Python Development Mode, introducing additional runtime checks that are too expensive to be enabled by default. See also `PYTHONDEVMODE`.

Added in version 3.7.

- `-X utf8` habilita el modo Python UTF-8. `-X utf8=0` desactiva explícitamente el modo Python UTF-8 (incluso cuando al contrario se activaría automáticamente). Véase [PYTHONUTF8](#).

Added in version 3.7.

- `-X pycache_prefix=PATH` permite escribir archivos `.pyc` en un árbol paralelo enraizado en el directorio dado en lugar de en el árbol de código. Véase también [PYTHONPYCACHEPREFIX](#).

Added in version 3.8.

- `-X warn_default_encoding` emite un `EncodingWarning` cuando se usa la codificación predeterminada específica de la configuración regional para abrir archivos. Vea también [PYTHONWARNDEFAULTENCODING](#).

Added in version 3.10.

- `-X no_debug_ranges` deshabilita la inclusión de tablas que asignan información de ubicación adicional (línea final, desplazamiento de columna inicial y desplazamiento de columna final) a cada instrucción en los objetos de código. Esto es útil cuando se desean objetos de código más pequeños y archivos `pyc`, además de suprimir los indicadores de ubicación visuales adicionales cuando el intérprete muestra rastreos. Véase también [PYTHONNODEBUGRANGES](#).

Added in version 3.11.

- `-X frozen_modules` determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` if this is an installed Python (the normal case). If it's under development (running from the source tree) then the default is `off`. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`. See also [PYTHON_FROZEN_MODULES](#).

Added in version 3.11.

- `-X perf` habilita el soporte para el perfilador `perf` de Linux. Cuando se proporciona esta opción, el perfilador `perf` podrá informar llamadas de Python. Esta opción sólo está disponible en algunas plataformas y no hará nada si no es compatible con el sistema actual. El valor predeterminado es «off». Consulte también [PYTHONPERFSUPPORT](#) y `perf_profiling`.

Added in version 3.12.

- `-X perf_jit` enables support for the Linux `perf` profiler with DWARF support. When this option is provided, the `perf` profiler will be able to report Python calls using DWARF information. This option is only available on some platforms and will do nothing if is not supported on the current system. The default value is «off». See also [PYTHON_PERF_JIT_SUPPORT](#) and `perf_profiling`.

Added in version 3.13.

- `-X disable_remote_debug` disables the remote debugging support as described in [PEP 768](#). This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

This option is only available on some platforms and will do nothing if is not supported on the current system. See also [PYTHON_DISABLE_REMOTE_DEBUG](#) and [PEP 768](#).

Added in version 3.14.

- `-X cpu_count=n` overrides `os.cpu_count()`, `os.process_cpu_count()`, and `multiprocessing.cpu_count()`. `n` must be greater than or equal to 1. This option may be useful for users who need to limit CPU resources of a container system. See also [PYTHON_CPU_COUNT](#). If `n` is default, nothing is overridden.

Added in version 3.13.

- `-X presite=package.module` specifies a module that should be imported before the `site` module is executed and before the `__main__` module exists. Therefore, the imported module isn't `__main__`. This can be used to execute code early during Python initialization. Python needs to be *built in debug mode* for this option to exist. See also [PYTHON_PRESITE](#).

Added in version 3.13.

- `-X gil=0, 1` forces the GIL to be disabled or enabled, respectively. Setting to 0 is only available in builds configured with `--disable-gil`. See also `PYTHON_GIL` and `whatsnew313-free-threaded-cpython`.

Added in version 3.13.

- `-X thread_inherit_context=0, 1` causes `Thread` to, by default, use a copy of context of the caller of `Thread.start()` when starting. Otherwise, threads will start with an empty context. If unset, the value of this option defaults to 1 on free-threaded builds and to 0 otherwise. See also `PYTHON_THREAD_INHERIT_CONTEXT`.

Added in version 3.14.

- `-X context_aware_warnings=0, 1` causes the `warnings.catch_warnings` context manager to use a `ContextVar` to store warnings filter state. If unset, the value of this option defaults to 1 on free-threaded builds and to 0 otherwise. See also `PYTHON_CONTEXT_AWARE_WARNINGS`.

Added in version 3.14.

- `-X tlbcc=0, 1` enables (1, the default) or disables (0) thread-local bytecode in builds configured with `--disable-gil`. When disabled, this also disables the specializing interpreter. See also `PYTHON_TLBC`.

Added in version 3.14.

También permite pasar valores arbitrarios y recuperarlos a través del diccionario `sys._xoptions`.

Added in version 3.2.

Distinto en la versión 3.9: Removed the `-X showalloccount` option.

Distinto en la versión 3.10: Removed the `-X oldparser` option.

Removed in version 3.14: `-J` is no longer reserved for use by `Jython`, and now has no special meaning.

1.1.4 Controlling color

The Python interpreter is configured by default to use colors to highlight output in certain situations such as when displaying tracebacks. This behavior can be controlled by setting different environment variables.

Setting the environment variable `TERM` to `dumb` will disable color.

If the `FORCE_COLOR` environment variable is set, then color will be enabled regardless of the value of `TERM`. This is useful on CI systems which aren't terminals but can still display ANSI escape sequences.

If the `NO_COLOR` environment variable is set, Python will disable all color in the output. This takes precedence over `FORCE_COLOR`.

All these environment variables are used also by other tools to control color output. To control the color output only in the Python interpreter, the `PYTHON_COLORS` environment variable can be used. This variable takes precedence over `NO_COLOR`, which in turn takes precedence over `FORCE_COLOR`.

1.2 Variables de entorno

Estas variables de entorno influyen en el comportamiento de Python, se procesan antes de que los modificadores de línea de comandos distintos de `-E` o `-I`. Es habitual que los modificadores de línea de comandos anulen variables de entorno donde hay un conflicto.

PYTHONHOME

Cambie la ubicación de las bibliotecas estándar de Python. De forma predeterminada, las bibliotecas se buscan en `prefix/lib/pythonversion` y `exec_prefix/lib/pythonversion`, donde `prefix` y `exec_prefix` son directorios dependientes de la instalación, ambos predeterminados `/usr/local`.

Cuando `PYTHONHOME` se establece en un único directorio, su valor reemplaza tanto al `prefix` como a `exec_prefix`. Para especificar valores diferentes para estos, establezca `PYTHONHOME` en `prefix:exec_prefix`.

PYTHONPATH

Aumente la ruta de búsqueda predeterminada para los archivos de módulo. El formato es el mismo que el de shell `PATH`: uno o más nombres de ruta de directorio separados por `os.pathsep` (por ejemplo, dos puntos en Unix o punto y coma en Windows). Los directorios inexistentes se omiten silenciosamente.

Además de los directorios normales, las entradas individuales `PYTHONPATH` pueden referirse a archivos zip que contienen módulos Python puros (ya sea en forma de origen o compilado). Los módulos de extensión no se pueden importar desde zipfiles.

La ruta de búsqueda predeterminada depende de la instalación, pero generalmente comienza con `prefix/lib/pythonversion` (consulte `PYTHONHOME` arriba). Es *always* anexo a `PYTHONPATH`.

Se insertará un directorio adicional en la ruta de búsqueda delante de `PYTHONPATH` como se describió anteriormente en *Opciones de interfaz*. La ruta de búsqueda se puede manipular desde un programa Python como la variable `sys.path`.

PYTHONSAFEPATH

Si se establece en una cadena no vacía, no anteponga una ruta potencialmente insegura a `sys.path`: véase la opción `-u` para más información.

Added in version 3.11.

PYTHONPLATLIBDIR

Si se establece en una cadena no vacía, anula el valor `sys.platlibdir`.

Added in version 3.9.

PYTHONSTARTUP

Si este es el nombre de un archivo legible, los comandos de Python de ese archivo se ejecutan antes de que el primer mensaje se muestre en modo interactivo. El archivo se ejecuta en el mismo espacio de nombres donde se ejecutan comandos interactivos para que los objetos definidos o importados en él se puedan usar sin calificación en la sesión interactiva. También puede cambiar las solicitudes `sys.ps1` y `sys.ps2` y el enlace `sys.__interactivehook__` en este archivo.

Lanza auditing event `cpython.run_startup` con el nombre de archivo como argumento cuando se llama al inicio.

PYTHONOPTIMIZE

Si se establece en una cadena no vacía, equivale a especificar la opción `-O`. Si se establece en un entero, es equivalente a especificar `-O` varias veces.

PYTHONBREAKPOINT

Si se establece, nombra un nombre que se puede llamar mediante la notación de trayecto de puntos. El módulo que contiene el invocable se importará y, a continuación, el invocable se ejecutará por la implementación predeterminada de `sys.breakpointhook()` que a su vez se llama por incorporado `breakpoint()`. Si no se establece o se establece en la cadena vacía, es equivalente al valor «`pdb.set_trace`». Establecer esto en la cadena «0» hace que la implementación predeterminada de `sys.breakpointhook()` no haga nada más que retornar inmediatamente.

Added in version 3.7.

PYTHONDEBUG

Si se establece en una cadena no vacía, equivale a especificar la opción `-d`. Si se establece en un entero, equivale a especificar `-d` varias veces.

Esta variable de entorno necesita una *compilación de depuración de Python*, de lo contrario será ignorada.

PYTHONINSPECT

Si se establece en una cadena no vacía, equivale a especificar la opción `-i`.

Esta variable también se puede modificar mediante código Python mediante `os.environ` para forzar el modo de inspección en la terminación del programa.

Lanza un evento auditing event `cpython.run_stdin` sin argumentos.

Distinto en la versión 3.12.5: (also 3.11.10, 3.10.15, 3.9.20, and 3.8.20) Emits audit events.

Distinto en la versión 3.13: Uses PyREPL if possible, in which case `PYTHONSTARTUP` is also executed. Emits audit events.

PYTHONUNBUFFERED

Si se establece en una cadena no vacía, equivale a especificar la opción `-u`.

PYTHONVERBOSE

Si se establece en una cadena no vacía, equivale a especificar la opción `-v`. Si se establece en un entero, equivale a especificar `-v` varias veces.

PYTHONCASEOK

Si se establece, Python omite mayúsculas y minúsculas en las instrucciones `import`. Esto sólo funciona en Windows y macOS.

PYTHONDONTWRITEBYTECODE

Si se establece en una cadena no vacía, Python no intentará escribir archivos `.pyc` en la importación de módulos de origen. Esto equivale a especificar la opción `-B`.

PYTHONPYCACHEPREFIX

Si se establece, Python escribirá archivos `.pyc` en un árbol de directorios reflejado en esta ruta de acceso, en lugar de en directorios `__pycache__` dentro del árbol de origen. Esto equivale a especificar la opción `-X pycache_prefix=PATH`.

Added in version 3.8.

PYTHONHASHSEED

Si esta variable no se establece o se establece en `random`, se utiliza un valor aleatorio para sembrar los hashes de los objetos `str` y `bytes`.

Si `PYTHONHASHSEED` se establece en un valor entero, se utiliza como una semilla fija para generar el `hash()` de los tipos cubiertos por la aleatorización `hash`.

Su propósito es permitir el `hash` repetible, como para las autocomprobaciones para el propio intérprete, o permitir que un grupo de procesos python comparta valores `hash`.

El entero debe ser un número decimal en el intervalo `[0,4294967295]`. Especificar el valor 0 deshabilitará la aleatorización de `hash`.

Added in version 3.2.3.

PYTHONINTMAXSTRDIGITS

Si esta variable se establece en un número entero, se utiliza para configurar la limitación de longitud de conversión de cadena entera.

Added in version 3.11.

PYTHONIOENCODING

Si se establece antes de ejecutar el intérprete, invalida la codificación utilizada para `stdin/stdout/stderr`, en la sintaxis `encodingname:errorhandler`. Tanto las partes `encodingname` como `:errorhandler` son opcionales y tienen el mismo significado que en `str.encode()`.


Para `stderr`, se omite la parte `:errorhandler`; el manejador siempre será `'backslashreplace'`.

Distinto en la versión 3.4: La parte `encodingname` ahora es opcional.


Distinto en la versión 3.6: En Windows, la codificación especificada por esta variable se omite para los búferes de consola interactivos a menos que también se especifique `PYTHONLEGACYWINDOWSSTDIO`. Los archivos y canalizaciones redirigidos a través de las corrientes estándar no se ven afectados.

PYTHONNOUSERSITE

Si se establece, Python no agregará `user site-packages` directory a `sys.path`.

 **Ver también****PEP 370** – Directorio de paquetes de sitio por usuario**PYTHONUSERBASE**

Define el `user base directory`, que se utiliza para calcular la ruta de acceso de `user site-packages directory` y `installation paths` para `python -m pip install --user`.

 **Ver también****PEP 370** – Directorio de paquetes de sitio por usuario**PYTHONEXECUTABLE**

Si se establece esta variable de entorno, `sys.argv[0]` se establecerá en su valor en lugar del valor conseguido a través del tiempo de ejecución de C. Sólo funciona en macOS.

PYTHONWARNINGS

Esto es equivalente a la opción `-W`. Si se establece en una cadena separada por comas, es equivalente a especificar `-W` varias veces, con filtros más adelante en la lista que tienen prioridad sobre los anteriores de la lista.

La configuración más sencilla aplica una acción determinada incondicionalmente a todas las advertencias emitidas por un proceso (incluso aquellas que de otro modo se ignoran de forma predeterminada):

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

Consulte `warning-filter` y `describing-warning-filters` para obtener más detalles.

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.

Added in version 3.3.

PYTHONTRACEMALLOC

Si esta variable de entorno se establece en una cadena no vacía, comience a trazar las asignaciones de memoria de Python mediante el módulo `tracemalloc`. El valor de la variable es el número máximo de marcos almacenados en un rastreo de un seguimiento. Por ejemplo, `PYTHONTRACEMALLOC=1` almacena sólo el marco más reciente. Consulte la función `tracemalloc.start()` para obtener más información. Esto equivale a configurar la opción `-X tracemalloc`.

Added in version 3.4.

PYTHONPROFILEIMPORTTIME

If this environment variable is set to 1, Python will show how long each import takes. If set to 2, Python will include output for imported modules that have already been loaded. This is equivalent to setting the `-X importtime` option.

Added in version 3.7.

Distinto en la versión 3.14: Added `PYTHONPROFILEIMPORTTIME=2` to also trace imports of loaded modules.

PYTHONASYNCIODEBUG

Si esta variable de entorno se establece en una cadena no vacía, habilite el modo debug mode del módulo `asyncio`.

Added in version 3.4.

PYTHONMALLOC

Establezca los asignadores de memoria de Python y/o instale enlaces de depuración.

Establezca la familia de asignadores de memoria utilizados por Python:

- `default`: utilice default memory allocators.
- `malloc`: utilice la función `malloc()` de la biblioteca C para todos los dominios (`PYMEM_DOMAIN_RAW`, `PYMEM_DOMAIN_MEM`, `PYMEM_DOMAIN_OBJ`).
- `pymalloc`: utilice el `pymalloc` allocator para `PYMEM_DOMAIN_MEM` y dominios `PYMEM_DOMAIN_OBJ` y utilice la función `malloc()` para el dominio `PYMEM_DOMAIN_RAW`.
- `mimalloc`: use the `mimalloc` allocator for `PYMEM_DOMAIN_MEM` and `PYMEM_DOMAIN_OBJ` domains and use the `malloc()` function for the `PYMEM_DOMAIN_RAW` domain.

Instalar enlaces de depuración:

- `debug`: instale los enlaces de depuración encima de default memory allocators.
- `malloc_debug`: igual que `malloc` pero también instalar ganchos de depuración.
- `pymalloc_debug`: igual que `pymalloc` pero también instalar enlaces de depuración.
- `mimalloc_debug`: same as `mimalloc` but also install debug hooks.

Added in version 3.6.

Distinto en la versión 3.7: Se ha añadido el asignador "predeterminado".

PYTHONMALLOCSTATS

Si se establece en una cadena no vacía, Python imprimirá estadísticas de `pymalloc` memory allocator cada vez que se crea una nueva arena de objetos `pymalloc` y al apagarse.

Esta variable se omite si la variable de entorno `PYTHONMALLOC` se utiliza para forzar el asignador `malloc()` de la biblioteca C, o si Python está configurado sin compatibilidad con `pymalloc`.

Distinto en la versión 3.6: Esta variable ahora también se puede utilizar en Python compilado en modo de versión. Ahora no tiene ningún efecto si se establece en una cadena vacía.

PYTHONLEGACYWINDOWSFSENCODING

Si se establece en una cadena no vacía, el *filesystem encoding and error handler* predeterminado volverá a sus valores pre-3.6 de *mbcs* y *replace*, respectivamente. De lo contrario, se utilizan los nuevos valores predeterminados "utf-8" y "surrogatepass".

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

Availability: Windows.

Added in version 3.6: Consulte **PEP 529** para obtener más detalles.

PYTHONLEGACYWINDOWSSSTDIO

Si se establece en una cadena no vacía, no utiliza el nuevo lector y escritor de consola. Esto significa que los caracteres Unicode se codificarán de acuerdo con la página de códigos de la consola activa, en lugar de usar utf-8.

Esta variable se omite si se redirigen las secuencias estándar (a archivos o canalizaciones) en lugar de hacer referencia a búferes de consola.

Availability: Windows.

Added in version 3.6.

PYTHONCOERCECLOCALE

Si se establece en el valor 0, hace que la aplicación principal de línea de comandos de Python omita la coerción de las configuraciones regionales C y POSIX basadas en ASCII heredadas a una alternativa basada en UTF-8 más capaz.

Si esta variable es *no* establecida (o se establece en un valor distinto de 0), tampoco se establece la variable de entorno de invalidación local `LC_ALL`, y la configuración local actual notificada para la categoría `LC_CTYPE` es la configuración local C predeterminada, o bien la configuración local basada explícitamente en ASCII POSIX, entonces la CLI de Python intentará configurar las siguientes configuraciones locales para la categoría `LC_CTYPE` en el orden indicado antes de cargar el tiempo de ejecución del intérprete:

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

Si la configuración de una de estas categorías de configuración local se realiza correctamente, la variable de entorno `LC_CTYPE` también se establecerá en consecuencia en el entorno de proceso actual antes de que se inicialice el tiempo de ejecución de Python. Esto garantiza que, además de ser visto tanto por el propio intérprete como por otros componentes compatibles con la configuración local que se ejecutan en el mismo proceso (como la biblioteca GNU `readline`), la configuración actualizada también se ve en los subprocessos (independientemente de si esos procesos están ejecutando o no un intérprete de Python), así como en las operaciones que consultan el entorno en lugar de la configuración regional de C actual (como la propia de Python `locale.getdefaultlocale()`).

La configuración de una de estas configuraciones regionales (ya sea explícitamente o a través de la coerción de configuración regional implícita anterior) habilita automáticamente el `surrogateescape` error handler para `sys.stdin` y `sys.stdout` (`sys.stderr` continúa utilizando `backslashreplace` como lo hace en cualquier otra configuración local). Este comportamiento de control de secuencias se puede invalidar mediante `PYTHONIOENCODING` como de costumbre.

Para fines de depuración, establecer `PYTHONCOERCECLOCALE-warn` hará que Python emita mensajes de advertencia en `stderr` si se activa la coerción de configuración regional, o si una configuración regional que *would* ha activado la coerción sigue activa cuando se inicializa el tiempo de ejecución de Python.

También tenga en cuenta que incluso cuando la coerción de configuración regional está desactivada, o cuando no puede encontrar una configuración regional de destino adecuada, `PYTHONUTF8` se activará de forma predeterminada en las configuraciones regionales heredadas basadas en ASCII. Ambas funciones deben estar deshabilitadas para obligar al intérprete a usar ASCII en lugar de UTF-8 para las interfaces del sistema.

Availability: Unix.

Added in version 3.7: Consulte [PEP 538](#) para obtener más detalles.

PYTHONDEVMODE

Si esta variable de entorno se establece en una cadena no vacía, habilite Python Development Mode, introduciendo comprobaciones de tiempo de ejecución adicionales que son demasiado caras para habilitarse de forma predeterminada. Esto equivale a configurar la opción `-X dev`.

Added in version 3.7.

PYTHONUTF8

Si se establece en 1, habilita el modo Python UTF-8.

Si se establece en 0, deshabilita el modo Python UTF-8.

Establecer cualquier otra cadena no vacía produce un error durante la inicialización del intérprete.

Added in version 3.7.

PYTHONWARNDEFAULTENCODING

Si esta variable de entorno se establece como una cadena de caracteres no vacía, emite un `EncodingWarning` cuando se utilice la codificación predeterminada específica de la configuración regional.

Ver `io-encoding-warning` para más detalles.

Added in version 3.10.

PYTHONNODEBUGRANGES

Si se establece esta variable, deshabilita la inclusión de tablas que asignan información de ubicación adicional (línea final, desplazamiento de columna inicial y desplazamiento de columna final) a cada instrucción en los objetos de código. Esto es útil cuando se desean objetos de código más pequeños y archivos pyc, además de suprimir los indicadores de ubicación visuales adicionales cuando el intérprete muestra rastreos.

Added in version 3.11.

PYTHONPERFSUPPORT

Si esta variable se establece en un valor distinto de cero, habilita la compatibilidad con el perfilador `perf` de Linux para que pueda detectar las llamadas de Python.

Si se establece en 0, deshabilite la compatibilidad con el generador de perfiles `perf` de Linux.

Consulte también la opción de línea de comandos `-X perf` y `perf_profiling`.

Added in version 3.12.

PYTHON_PERF_JIT_SUPPORT

If this variable is set to a nonzero value, it enables support for the Linux `perf` profiler so Python calls can be detected by it using DWARF information.

Si se establece en 0, deshabilite la compatibilidad con el generador de perfiles `perf` de Linux.

See also the `-X perf_jit` command-line option and `perf_profiling`.

Added in version 3.13.

PYTHON_DISABLE_REMOTE_DEBUG

If this variable is set to a non-empty string, it disables the remote debugging feature described in [PEP 768](#). This includes both the functionality to schedule code for execution in another process and the functionality to receive code for execution in the current process.

See also the `-X disable_remote_debug` command-line option.

Added in version 3.14.

PYTHON_CPU_COUNT

If this variable is set to a positive integer, it overrides the return values of `os.cpu_count()` and `os.process_cpu_count()`.

See also the `-X cpu_count` command-line option.

Added in version 3.13.

PYTHON_FROZEN_MODULES

If this variable is set to `on` or `off`, it determines whether or not frozen modules are ignored by the import machinery. A value of `on` means they get imported and `off` means they are ignored. The default is `on` for non-debug builds (the normal case) and `off` for debug builds. Note that the `importlib_bootstrap` and `importlib_bootstrap_external` frozen modules are always used, even if this flag is set to `off`.

See also the `-X frozen_modules` command-line option.

Added in version 3.13.

PYTHON_COLORS

If this variable is set to 1, the interpreter will colorize various kinds of output. Setting it to 0 deactivates this behavior. See also [Controlling color](#).

Added in version 3.13.

PYTHON_BASIC_REPL

If this variable is set to any value, the interpreter will not attempt to load the Python-based [REPL](#) that requires `curses` and `readline`, and will instead use the traditional parser-based [REPL](#).

Added in version 3.13.

PYTHON_HISTORY

This environment variable can be used to set the location of a `.python_history` file (by default, it is `.python_history` in the user's home directory).

Added in version 3.13.

PYTHON_GIL

If this variable is set to 1, the global interpreter lock (GIL) will be forced on. Setting it to 0 forces the GIL off (needs Python configured with the `--disable-gil` build option).

See also the `-X gil` command-line option, which takes precedence over this variable, and `whatsnew313-free-threaded-cpython`.

Added in version 3.13.

PYTHON_THREAD_INHERIT_CONTEXT

If this variable is set to 1 then `Thread` will, by default, use a copy of context of the caller of `Thread.start()` when starting. Otherwise, new threads will start with an empty context. If unset, this variable defaults to 1 on free-threaded builds and to 0 otherwise. See also `-X thread_inherit_context`.

Added in version 3.14.

PYTHON_CONTEXT_AWARE_WARNINGS

If set to 1 then the `warnings.catch_warnings` context manager will use a `ContextVar` to store warnings filter state. If unset, this variable defaults to 1 on free-threaded builds and to 0 otherwise. See `-X context_aware_warnings`.

Added in version 3.14.

PYTHON_JIT

On builds where experimental just-in-time compilation is available, this variable can force the JIT to be disabled (0) or enabled (1) at interpreter startup.

Added in version 3.13.

PYTHON_TLBC

If set to 1 enables thread-local bytecode. If set to 0 thread-local bytecode and the specializing interpreter are disabled. Only applies to builds configured with `--disable-gil`.

See also the `-X tlbc` command-line option.

Added in version 3.14.

1.2.1 Variables de modo de depuración

PYTHONDUMPPREFS

Si se establece, Python volcará objetos y recuentos de referencias aún vivos después de apagar el intérprete.

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

PYTHONDUMPPREFSFILE

If set, Python will dump objects and reference counts still alive after shutting down the interpreter into a file under the path given as the value to this environment variable.

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

Added in version 3.11.

PYTHON_PRESITE

If this variable is set to a module, that module will be imported early in the interpreter lifecycle, before the `site` module is executed, and before the `__main__` module is created. Therefore, the imported module is not treated as `__main__`.

This can be used to execute code early during Python initialization.

To import a submodule, use `package.module` as the value, like in an import statement.

See also the `-X presite` command-line option, which takes precedence over this variable.

Needs Python configured with the `--with-pydebug` build option.

Added in version 3.13.

Uso de Python en plataformas Unix

2.1 Obteniendo e instalando la última versión de Python

2.1.1 En 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 compile the latest version of Python from source.

In the event that the latest version of Python doesn't come preinstalled and isn't in the repositories as well, you can 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>

para usuarios de Debian

<https://en.opensuse.org/Portal:Packaging>

para los usuarios de OpenSuse

https://docs.fedoraproject.org/en-US/package-maintainers/Packaging_Tutorial_GNU_Hello/

para los usuarios de Fedora

<https://slackbook.org/html/package-management-making-packages.html>

para los usuarios de Slackware

Installing IDLE

In some cases, IDLE might not be included in your Python installation.

- For Debian and Ubuntu users:

```
sudo apt update
sudo apt install idle
```

- For Fedora, RHEL, and CentOS users:

```
sudo dnf install python3-idle
```

- For SUSE and OpenSUSE users:

```
sudo zypper install python3-idle
```

- For Alpine Linux users:

```
sudo apk add python3-idle
```

2.1.2 En FreeBSD y OpenBSD

- Usuarios FreeBSD, para añadir al paquete use:

```
pkg install python3
```

- Usuarios OpenBSD, para añadir al paquete use:

```
pkg_add -r python

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

Por ejemplo, los usuarios de i386 obtienen la versión 2.5.1 de Python usando:

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

2.2 Construyendo Python

Si desea compilar CPython por sí mismo, lo primero que debería hacer es obtener la [fuente](#). Puede descargar la fuente de la última versión o simplemente conseguir un nuevo [clon](#). (Si desea contribuir con parches, necesitará un clon.)

El proceso de construcción consta de los comandos habituales:

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

Opciones de configuración y las advertencias para plataformas Unix específicas están ampliamente documentadas en el archivo [README.rst](#) en la raíz del árbol de fuentes de Python.

Advertencia

`make install` puede sobrescribir o enmascarar el binario `python3`. Por lo tanto se recomienda `make altinstall` en lugar de `make install` debido a que sólo instala `exec_prefix/bin/pythonversion`.

2.3 Rutas y archivos relacionados con Python

Estos están sujetos a diferencias según las convenciones de instalación locales; `prefix` y `exec_prefix` son dependientes de la instalación y deben interpretarse como en el software GNU; pueden ser iguales.

Por ejemplo, en la mayoría de los sistemas Linux, el valor predeterminado para ambos es `/usr`.

Archivo/directorio	Significado
<code>exec_prefix/bin/python3</code>	Ubicación recomendada del intérprete.
<code>prefix/lib/pythonversion,</code> <code>exec_prefix/lib/</code> <code>pythonversion</code>	Ubicaciones recomendadas de los directorios que contienen los módulos estándar.
<code>prefix/include/</code> <code>pythonversion,</code> <code>exec_prefix/</code> <code>include/pythonversion</code>	Ubicaciones recomendadas de los directorios que contienen los archivos de inclusión necesarios para desarrollar extensiones de Python e incrustar el intérprete.

2.4 Miscelánea

Para usar fácilmente los scripts de Python en Unix, debe hacerlos ejecutables, p. ej. con

```
$ chmod +x script
```

y coloque una línea *Shebang* adecuada en la parte superior del script. Una buena opción es usualmente

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

que busca el intérprete de Python en el conjunto `PATH`. Sin embargo, algunos Unixes puede que no tengan el comando `env`, por lo que es posible que deba codificar `/usr/bin/python3` como la ruta intérprete.

Para usar comandos de shell en sus scripts de Python, mire el módulo `subprocess`.

2.5 OpenSSL personalizado

1. Para utilizar la configuración de OpenSSL de su proveedor y el almacén de confianza del sistema, busque el directorio con el archivo `openssl.cnf` o el enlace simbólico en `/etc`. En la mayoría de las distribuciones, el archivo está en `/etc/ssl` o `/etc/pki/tls`. El directorio también debe contener un archivo `cert.pem` y / o un directorio `certs`.

```
$ find /etc/ -name openssl.cnf -printf "%h\n"
/etc/ssl
```

2. Descargue, compile e instale OpenSSL. Asegúrese de utilizar `install_sw` y no `install`. El destino `install_sw` no anula `openssl.cnf`.

```
$ curl -O https://www.openssl.org/source/openssl-VERSION.tar.gz
$ tar xzf openssl-VERSION
$ pushd openssl-VERSION
$ ./config \
  --prefix=/usr/local/custom-openssl \
  --libdir=lib \
  --openssldir=/etc/ssl
$ make -j1 depend
$ make -j8
$ make install_sw
$ popd
```

3. Construir Python con OpenSSL personalizado (consulte las opciones `configure --with-openssl` y `--with-openssl-rpath`)

```
$ pushd python-3.x.x
$ ./configure -C \
  --with-openssl=/usr/local/custom-openssl \
```

(continúe en la próxima página)

(proviene de la página anterior)

```
--with-openssl-rpath=auto \
--prefix=/usr/local/python-3.x.x
$ make -j8
$ make altinstall
```

Nota

Las versiones de parche de OpenSSL tienen una ABI compatible con versiones anteriores. No es necesario volver a compilar Python para actualizar OpenSSL. Es suficiente reemplazar la instalación personalizada de OpenSSL con una versión más nueva.

3.1 Requisitos de compilación

Features and minimum versions required to build CPython:

- Un compilador C11. Características opcionales de C11 no son necesarias.
- En Windows, se necesita Microsoft Visual Studio 2017 o posterior.
- Support for IEEE 754 floating-point numbers and floating-point Not-a-Number (NaN).
- Soporte para hilos.
- OpenSSL 1.1.1 is the minimum version and OpenSSL 3.0.16 is the recommended minimum version for the `ssl` and `hashlib` extension modules.
- SQLite 3.15.2 for the `sqlite3` extension module.
- Tcl/Tk 8.5.12 for the `tkinter` module.
- `libmpdec` 2.5.0 for the `decimal` module.
- Autoconf 2.72 and `aclocal` 1.16.5 are required to regenerate the `configure` script.

Distinto en la versión 3.1: Tcl/Tk version 8.3.1 is now required.

Distinto en la versión 3.5: On Windows, Visual Studio 2015 or later is now required. Tcl/Tk version 8.4 is now required.

Distinto en la versión 3.6: Ahora se necesitan características seleccionadas de C99, como `<stdint.h>` y funciones `static inline`.

Distinto en la versión 3.7: Ahora se necesita soporte de hilos y OpenSSL 1.0.2.

Distinto en la versión 3.10: OpenSSL 1.1.1 is now required. Require SQLite 3.7.15.

Distinto en la versión 3.11: C11 compiler, IEEE 754 and NaN support are now required. On Windows, Visual Studio 2017 or later is required. Tcl/Tk version 8.5.12 is now required for the `tkinter` module.

Distinto en la versión 3.13: Autoconf 2.71, `aclocal` 1.16.5 and SQLite 3.15.2 are now required.

Distinto en la versión 3.14: Autoconf 2.72 is now required.

Ver también [PEP 7](#) «Style Guide for C Code» y [PEP 11](#) «CPython platform support».

3.2 Archivos generados

Para reducir las dependencias de compilación, el código fuente de Python contiene varios archivos generados. Comandos para regenerar todos los archivos generados:

```
make regen-all
make regen-stdlib-module-names
make regen-limited-abi
make regen-configure
```

El archivo `Makefile.pre.in` documenta archivos generados, sus entradas y las herramientas que se usaron para regenerarlos. Busque los objetivos `make regen-*`.

3.2.1 configure script

The `make regen-configure` command regenerates the `aclocal.m4` file and the `configure` script using the `Tools/build/regen-configure.sh` shell script which uses an Ubuntu container to get the same tools versions and have a reproducible output.

The container is optional, the following command can be run locally:

```
autoreconf -ivf -Werror
```

The generated files can change depending on the exact `autoconf-archive`, `aclocal` and `pkg-config` versions.

3.3 Configurar opciones

List all `configure` script options using:

```
./configure --help
```

Consultar también `Misc/SpecialBuilds.txt` en la distribución fuente de Python.

3.3.1 Opciones generales

--enable-loadable-sqlite-extensions

Admite extensiones cargables en el módulo de extensión `_sqlite` (el valor por defecto es no) del módulo `sqlite3`.

Consultar el método `sqlite3.Connection.enable_load_extension()` del módulo `sqlite3`.

Added in version 3.6.

--disable-ipv6

Deshabilita la compatibilidad con IPv6 (habilitada de forma predeterminada si es compatible), consulte el módulo `socket`.

--enable-big-digits=[15|30]

Define el tamaño en bits de los dígitos `int` de Python: 15 o 30 bits.

Por defecto, el tamaño del dígito es 30.

Define el `PYLONG_BITS_IN_DIGIT` en 15 o 30.

Consultar `sys.int_info.bits_per_digit`.

--with-suffix=SUFFIX

Establece el sufijo ejecutable de Python en *SUFFIX*.

El sufijo predeterminado es `.exe` en Windows y macOS (ejecutable `python.exe`), `.js` en el nodo Emscripten, `.html` en el navegador Emscripten, `.wasm` en WASI y una cadena vacía en otras plataformas (ejecutable `python`).

Distinto en la versión 3.11: El sufijo predeterminado en la plataforma WASM es uno de `.js`, `.html` o `.wasm`.

--with-tzpath=<list of absolute paths separated by pathsep>

Selecciona la ruta de búsqueda de zona horaria predeterminada para `zoneinfo.TZPATH`. Consultar la Configuración en tiempo de compilación del módulo `zoneinfo`.

Por defecto: `/usr/share/zoneinfo:/usr/lib/zoneinfo:/usr/share/lib/zoneinfo:/etc/zoneinfo`.

Consultar separador de rutas `os.pathsep`.

Added in version 3.9.

--without-decimal-contextvar

Construye el módulo de extensión `_decimal` usando un contexto local de hilos en lugar de un contexto local de corutinas (predeterminado), consultar el módulo `decimal`.

Consultar `decimal.HAVE_CONTEXTVAR` y el módulo `contextvars`.

Added in version 3.9.

--with-dbmliborder=<list of backend names>

Sobrescribe el orden para verificar los de las bases datos para el módulo `dbm`

Un valor válido es una cadena de caracteres separada por dos puntos (:) con los nombres de los backends:

- `ndbm`;
- `gdbm`;
- `bdb`.

--without-c-locale-coercion

Deshabilita la coerción de configuración regional C a una configuración regional basada en UTF-8 (habilitada de forma predeterminada).

No define la macro `PY_COERCE_C_LOCALE`.

Consultar [PYTHONCOERCECLOCALE](#) y el [PEP 538](#).

--with-platlibdir=DIRNAME

Nombre del directorio de la biblioteca de Python (por defecto es `lib`).

Fedora y SuSE usan `lib64` en plataformas 64-bit.

Consultar `sys.platlibdir`.

Added in version 3.9.

--with-wheel-pkg-dir=PATH

Directorio de los paquetes *wheel* usados por el módulo `ensurepip` (ninguno por defecto)

Algunas políticas de empaquetado de distribución de Linux recomiendan no empaquetar dependencias. Por ejemplo, Fedora instala paquetes *wheel* en el directorio `/usr/share/python-wheels/` y no instala el paquete `ensurepip._bundled`.

Added in version 3.10.

--with-pkg-config=[check|yes|no]

Si configure debe usar **pkg-config** para detectar dependencias de compilación.

- `check` (predeterminado): **pkg-config** es opcional
- `yes`: **pkg-config** es obligatorio
- `no`: configure no usa **pkg-config** incluso cuando está presente

Added in version 3.11.

--enable-pystats

Turn on internal Python performance statistics gathering.

By default, statistics gathering is off. Use `python3 -X pystats` command or set `PYTHONSTATS=1` environment variable to turn on statistics gathering at Python startup.

At Python exit, dump statistics if statistics gathering was on and not cleared.

Efectos:

- Add `-X pystats` command line option.
- Add `PYTHONSTATS` environment variable.
- Define the `Py_STATS` macro.
- Add functions to the `sys` module:
 - `sys._stats_on()`: Turns on statistics gathering.
 - `sys._stats_off()`: Turns off statistics gathering.
 - `sys._stats_clear()`: Clears the statistics.
 - `sys._stats_dump()`: Dump statistics to file, and clears the statistics.

The statistics will be dumped to a arbitrary (probably unique) file in `/tmp/py_stats/` (Unix) or `C:\temp\py_stats\` (Windows). If that directory does not exist, results will be printed on stderr.

Usa `Tools/scripts/summarize_stats.py` para leer las estadísticas.

Statistics:

- Opcode:
 - Specialization: success, failure, hit, deferred, miss, deopt, failures;
 - Execution count;
 - Pair count.
- Call:
 - Inlined Python calls;
 - PyEval calls;
 - Frames pushed;
 - Frame object created;
 - Eval calls: vector, generator, legacy, function VECTORCALL, build class, slot, function «ex», API, method.
- Object:
 - incref and decref;
 - interpreter incref and decref;
 - allocations: all, 512 bytes, 4 kiB, big;
 - free;
 - to/from free lists;
 - dictionary materialized/dematerialized;
 - type cache;
 - optimization attempts;
 - optimization traces created/executed;
 - uops executed.

- Garbage collector:
 - Garbage collections;
 - Objects visited;
 - Objects collected.

Added in version 3.11.

--disable-gil

Enables support for running Python without the *global interpreter lock* (GIL): free threading build.

Defines the `Py_GIL_DISABLED` macro and adds "t" to `sys.abiflags`.

See [whatsnew313-free-threaded-cpython](#) for more detail.

Added in version 3.13.

--enable-experimental-jit=[no|yes|yes-off|interpreter]

Indicate how to integrate the experimental just-in-time compiler.

- `no`: Don't build the JIT.
- `yes`: Enable the JIT. To disable it at runtime, set the environment variable `PYTHON_JIT=0`.
- `yes-off`: Build the JIT, but disable it by default. To enable it at runtime, set the environment variable `PYTHON_JIT=1`.
- `interpreter`: Enable the «JIT interpreter» (only useful for those debugging the JIT itself). To disable it at runtime, set the environment variable `PYTHON_JIT=0`.

`--enable-experimental-jit=no` is the default behavior if the option is not provided, and `--enable-experimental-jit` is shorthand for `--enable-experimental-jit=yes`. See [Tools/jit/README.md](#) for more information, including how to install the necessary build-time dependencies.

Nota

When building CPython with JIT enabled, ensure that your system has Python 3.11 or later installed.

Added in version 3.13.

PKG_CONFIG

Path to `pkg-config` utility.

PKG_CONFIG_LIBDIR

PKG_CONFIG_PATH

`pkg-config` options.

3.3.2 C compiler options

CC

Comando del compilador C.

CFLAGS

Banderas del compilador de C.

CPP

C preprocessor command.

CPPFLAGS

C preprocessor flags, e.g. `-Iinclude_dir`.

3.3.3 Opciones del enlazador

LD_FLAGS

Linker flags, e.g. `-Llibrary_directory`.

LIBS

Libraries to pass to the linker, e.g. `-llibrary`.

MACHDEP

Name for machine-dependent library files.

3.3.4 Options for third-party dependencies

Added in version 3.11.

BZIP2_CFLAGS**BZIP2_LIBS**

C compiler and linker flags to link Python to `libbz2`, used by `bz2` module, overriding `pkg-config`.

CURSES_CFLAGS**CURSES_LIBS**

C compiler and linker flags for `libncurses` or `libncursesw`, used by `curses` module, overriding `pkg-config`.

GDBM_CFLAGS**GDBM_LIBS**

C compiler and linker flags for `gdbm`.

LIBB2_CFLAGS**LIBB2_LIBS**

C compiler and linker flags for `libb2` (BLAKE2), used by `hashlib` module, overriding `pkg-config`.

LIBEDIT_CFLAGS**LIBEDIT_LIBS**

C compiler and linker flags for `libedit`, used by `readline` module, overriding `pkg-config`.

LIBFFI_CFLAGS**LIBFFI_LIBS**

C compiler and linker flags for `libffi`, used by `ctypes` module, overriding `pkg-config`.

LIBMPDEC_CFLAGS**LIBMPDEC_LIBS**

C compiler and linker flags for `libmpdec`, used by `decimal` module, overriding `pkg-config`.

Nota

These environment variables have no effect unless `--with-system-libmpdec` is specified.

LIBLZMA_CFLAGS**LIBLZMA_LIBS**

C compiler and linker flags for `liblzma`, used by `lzma` module, overriding `pkg-config`.

LIBREADLINE_CFLAGS

LIBREADLINE_LIBS

C compiler and linker flags for `libreadline`, used by `readline` module, overriding `pkg-config`.

LIBSQLITE3_CFLAGS**LIBSQLITE3_LIBS**

C compiler and linker flags for `libsqlite3`, used by `sqlite3` module, overriding `pkg-config`.

LIBUUID_CFLAGS**LIBUUID_LIBS**

C compiler and linker flags for `libuuid`, used by `uuid` module, overriding `pkg-config`.

LIBZSTD_CFLAGS**LIBZSTD_LIBS**

C compiler and linker flags for `libzstd`, used by `compression.zstd` module, overriding `pkg-config`.

Added in version 3.14.

PANEL_CFLAGS**PANEL_LIBS**

C compiler and linker flags for `PANEL`, overriding `pkg-config`.

C compiler and linker flags for `libpanel` or `libpanelw`, used by `curses.panel` module, overriding `pkg-config`.

TCLTK_CFLAGS**TCLTK_LIBS**

C compiler and linker flags for `TCLTK`, overriding `pkg-config`.

ZLIB_CFLAGS**ZLIB_LIBS**

C compiler and linker flags for `libzlib`, used by `gzip` module, overriding `pkg-config`.

3.3.5 Opciones de WebAssembly

--enable-wasm-dynamic-linking

Active la compatibilidad con enlaces dinámicos para WASM.

La vinculación dinámica habilita `dlopen`. El tamaño del archivo del ejecutable aumenta debido a la eliminación limitada de código muerto y características adicionales.

Added in version 3.11.

--enable-wasm-pthreads

Active la compatibilidad con `pthread`s para WASM.

Added in version 3.11.

3.3.6 Opciones de instalación

--prefix=PREFIX

Instala archivos independientes de la arquitectura en `PREFIX`. En Unix, el valor predeterminado es `/usr/local`.

Este valor se puede recuperar en tiempo de ejecución al usar `sys.prefix`.

Como ejemplo, se puede utilizar `--prefix="$HOME/.local/"` para instalar Python en su directorio raíz.

--exec-prefix=EPREFIX

Instala archivos independientes de la arquitectura en EPREFIX, el valor predeterminado es `--prefix`.

Este valor se puede recuperar en tiempo de ejecución al usar `sys.exec_prefix`.

--disable-test-modules

No construya ni instale módulos de prueba, como el paquete `test` o el módulo de extensión `_testcapi` (construido e instalado por defecto).

Added in version 3.10.

--with-ensurepip=[upgrade|install|no]

Selecciona el comando `ensurepip` que se ejecuta en la instalación de Python:

- `upgrade` (por defecto): ejecutar el comando `python -m ensurepip --altinstall --upgrade`.
- `install`: ejecutar el comando `python -m ensurepip --altinstall`;
- `no`: no ejecuta `ensurepip`;

Added in version 3.6.

3.3.7 Opciones de desempeño

Se recomienda configurar Python usando `--enable-optimizations --with-lto` (PGO + LTO) para obtener el mejor rendimiento. El indicador experimental `--enable-bolt` también se puede usar para mejorar el rendimiento.

--enable-optimizations

Habilite la Optimización Guiada por Perfiles (*PGO* por sus siglas en inglés) usando `PROFILE_TASK` (deshabilitado de forma predeterminada).

El compilador de C Clang requiere el programa `llvm-profdata` para PGO. En macOS, GCC también lo requiere: GCC es solo un alias de Clang en macOS.

Desactiva también la interposición semántica en libpython si se usa `--enable-shared` y GCC: agregar `-fno-semantic-interposition` a los flags del compilador y del enlazador.

Nota

During the build, you may encounter compiler warnings about profile data not being available for some source files. These warnings are harmless, as only a subset of the code is exercised during profile data acquisition. To disable these warnings on Clang, manually suppress them by adding `-Wno-profile-instr-unprofiled` to `CFLAGS`.

Added in version 3.6.

Distinto en la versión 3.10: Usar `-fno-semantic-interposition` en GCC.

PROFILE_TASK

Variable de entorno utilizada en el Makefile: argumentos de la línea de comando Python para la tarea de generación de PGO.

Por defecto: `-m test --pgo --timeout=$(TESTTIMEOUT)`.

Added in version 3.8.

Distinto en la versión 3.13: Task failure is no longer ignored silently.

--with-lto=[full|thin|no|yes]

Habilita la Optimización de Tiempo de Enlace (*LTO* por sus siglas en inglés) en cualquier compilación (deshabilitado de forma predeterminada).

El compilador de C Clang requiere `llvm-ar` para LTO (`ar` en macOS), así como un enlazador compatible con LTO (`ld.gold` o `lld`).

Added in version 3.6.

Added in version 3.11: Para usar la función ThinLTO, use `--with-lto=thin` en Clang.

Distinto en la versión 3.12: Utiliza ThinLTO como política de optimización predeterminada en Clang si el compilador acepta el indicador.

--enable-bolt

Habilita el uso del [optimizador binario post-enlace BOLT](#) (deshabilitado de forma predeterminada).

BOLT es parte del proyecto LLVM pero no siempre se incluye en sus distribuciones binarias. Este indicador necesita que `llvm-bolt` y `merge-fdata` estén disponibles.

BOLT aún es un proyecto bastante nuevo, así que este indicador debería considerarse experimental por ahora. Debido a que esta herramienta opera en código máquina, su éxito depende de una combinación del entorno de compilación + los otros argumentos de configuración de optimización + la arquitectura del CPU, y no todas las combinaciones son compatibles. Se sabe que las versiones de BOLT anteriores a LLVM 16 bloquean BOLT en algunos escenarios. Se recomienda encarecidamente utilizar LLVM 16 o posterior para la optimización de BOLT.

Las variables `configure` `BOLT_INSTRUMENT_FLAGS` y `BOLT_APPLY_FLAGS` se pueden definir para sobrescribir el conjunto predeterminado de argumentos de `llvm-bolt` para instrumentar y aplicar datos BOLT a binarios, respectivamente.

Added in version 3.12.

BOLT_APPLY_FLAGS

Arguments to `llvm-bolt` when creating a [BOLT optimized binary](#).

Added in version 3.12.

BOLT_INSTRUMENT_FLAGS

Arguments to `llvm-bolt` when instrumenting binaries.

Added in version 3.12.

--with-computed-gotos

Habilita los *gotos* calculados en el ciclo de evaluación (habilitado de forma predeterminada en los compiladores compatibles).

--with-tail-call-interp

Enable interpreters using tail calls in CPython. If enabled, enabling PGO (`--enable-optimizations`) is highly recommended. This option specifically requires a C compiler with proper tail call support, and the [preserve_none](#) calling convention. For example, Clang 19 and newer supports this feature.

Added in version 3.14.

--without-mimalloc

Disable the fast mimalloc allocator (enabled by default).

Consultar también la variable de entorno [PYTHONMALLOC](#).

--without-pymalloc

Deshabilita el asignador de memoria especializado de Python pymalloc (habilitado de forma predeterminada).

Consultar también la variable de entorno [PYTHONMALLOC](#).

--without-doc-strings

Deshabilita las cadenas de caracteres de documentación estáticas para reducir el espacio de memoria (habilitado de forma predeterminada). Las cadenas de caracteres de documentación definidas en Python no se ven afectadas.

No define la macro `WITH_DOC_STRINGS`.

Consultar la macro `PyDoc_STRVAR()`.

--enable-profiling

Habilita el análisis de rendimiento de código (*profiling*) de nivel C con `gprof` (deshabilitado de forma predeterminada).

--with-strict-overflow

Agrega `-fstrict-overflow` a los indicadores del compilador de C (el valor predeterminado que agregamos en su lugar es `-fno-strict-overflow`).

--without-remote-debug

Deactivate remote debugging support described in **PEP 768** (enabled by default). When this flag is provided the code that allows the interpreter to schedule the execution of a Python file in a separate process as described in **PEP 768** is not compiled. This includes both the functionality to schedule code to be executed and the functionality to receive code to be executed.

Py_REMOTE_DEBUG

This macro is defined by default, unless Python is configured with `--without-remote-debug`.

Note that even if the macro is defined, remote debugging may not be available (for example, on an incompatible platform).

Added in version 3.14.

3.3.8 Compilación de depuración de Python

Una compilación de depuración de Python se construye con la opción de configuración `--with-pydebug`.

Efectos de una compilación de depuración:

- Muestra todas las advertencias de forma predeterminada: la lista de filtros de advertencia predeterminados está vacía en el módulo `warnings`.
- Agrega `sys.abiflags`.
- Agrega la función `sys.gettotalrefcount()`.
- Agrega la opción de línea de comando `-X showrefcount`.
- Agrega la opción de línea de comando `-d` y la variable de entorno `PYTHONDEBUG` para depurar el analizador.
- Agregue soporte para la variable `__lltrace__`: habilite el seguimiento de bajo nivel en el ciclo de evaluación del código de bytes si la variable está definida.
- Instala ganchos de depuración en los asignadores de memoria para detectar el desbordamiento del búfer y otros errores de memoria.
- Define las macros `Py_DEBUG` y `Py_REF_DEBUG`.
- Agregue verificaciones de tiempo de ejecución: código rodeado por `#ifdef Py_DEBUG` y `#endif`. Habilite las aserciones `assert(...)` y `_PyObject_ASSERT(...)`: no configure la macro `NDEBUG` (vea también la opción de configuración `--with-assertions`). Comprobaciones principales de tiempo de ejecución:
 - Agregue controles de sanidad en los argumentos de la función.
 - Los objetos `unicode` e `int` se crean con su memoria completa con un patrón para detectar el uso de objetos no inicializados.
 - Asegúrese de que las funciones que pueden borrar o reemplazar la excepción actual no se invocan con una excepción lanzada.
 - Verifique que las funciones de desasignador no cambien la excepción actual.
 - El recolector de basura (función `gc.collect()`) ejecuta algunas comprobaciones básicas sobre la consistencia de los objetos.
 - La macro `Py_SAFE_DOWNCAST()` comprueba el subdesbordamiento y el desbordamiento de enteros al realizar una conversión descendente de tipos anchos a tipos estrechos.

Consultar también Modo de Desarrollo de Python y la opción de configuración `--with-trace-refs`.

Distinto en la versión 3.8: Release builds and debug builds are now ABI compatible: defining the `Py_DEBUG` macro no longer implies the `Py_TRACE_REFS` macro (see the `--with-trace-refs` option).

3.3.9 Opciones de depuración

`--with-pydebug`

Compila Python en modo de depuración: define la macro `Py_DEBUG` (deshabilitada por defecto).

`--with-trace-refs`

Habilita las referencias de seguimiento con fines de depuración (deshabilitado de forma predeterminada).

Efectos:

- Define la macro `Py_TRACE_REFS`.
- Add `sys.getobjects()` function.
- Agrega la variable de entorno `PYTHONDUMPREFS`.

The `PYTHONDUMPREFS` environment variable can be used to dump objects and reference counts still alive at Python exit.

Statically allocated objects are not traced.

Added in version 3.8.

Distinto en la versión 3.13: This build is now ABI compatible with release build and *debug build*.

`--with-assertions`

Compila con las aserciones de C habilitadas (el valor predeterminado es no): `assert(...);` y `_PyObject_ASSERT(...);`.

Si se establece, la macro `NDEBUG` no está definida en la variable del compilador `OPT`.

Consultar también la opción `--with-pydebug` (*compilación de depuración*) que también habilita las aserciones.

Added in version 3.6.

`--with-valgrind`

Habilite la compatibilidad con Valgrind (el valor predeterminado es no).

`--with-dtrace`

Habilite la compatibilidad con DTrace (el valor predeterminado es no).

Consultar Instrumentación de CPython con DTrace y SystemTap.

Added in version 3.6.

`--with-address-sanitizer`

Enable AddressSanitizer memory error detector, `asan` (default is no). To improve ASan detection capabilities you may also want to combine this with `--without-pymalloc` to disable the specialized small-object allocator whose allocations are not tracked by ASan.

Added in version 3.6.

`--with-memory-sanitizer`

Habilita el detector de errores de asignación MemorySanitizer, `msan` (el valor predeterminado es no).

Added in version 3.6.

`--with-undefined-behavior-sanitizer`

Habilita el detector de comportamiento indefinido UndefinedBehaviorSanitizer, `ubsan` (el valor predeterminado es no).

Added in version 3.6.

--with-thread-sanitizer

Enable ThreadSanitizer data race detector, `tsan` (default is no).

Added in version 3.13.

3.3.10 Opciones del enlazador

--enable-shared

Habilita la compilación de una biblioteca compartida de Python `:libpython` (el valor predeterminado es no).

--without-static-libpython

No compila `libpythonMAJOR.MINOR.a` y no instala `python.o` (compilado y habilitado de forma predefinida).

Added in version 3.10.

3.3.11 Opciones de bibliotecas

--with-libs='lib1 ...'

Enlace con bibliotecas adicionales (el valor predeterminado es no).

--with-system-expat

Compila el módulo `pyexpat` usando la biblioteca instalada `expat` instalada (por defecto es no).


--with-system-libmpdec

Build the `_decimal` extension module using an installed `mpdecimal` library, see the `decimal` module (default is yes).

Added in version 3.3.

Distinto en la versión 3.13: Default to using the installed `mpdecimal` library.

Deprecated since version 3.13, will be removed in version 3.15: A copy of the `mpdecimal` library sources will no longer be distributed with Python 3.15.

 **Ver también**

`LIBMPDEC_CFLAGS` and `LIBMPDEC_LIBS`.

--with-readline=readline|editline

Designate a backend library for the `readline` module.

- `readline`: Use `readline` as the backend.
- `editline`: Use `editline` as the backend.

Added in version 3.10.

--without-readline

No cree el módulo `readline` (es construido por defecto).

No defina la macro `HAVE_LIBREADLINE`.

Added in version 3.10.

--with-libm=STRING

Sobreescribe la biblioteca matemática `libm` a *STRING* (el valor predeterminado es dependiente del sistema).

--with-libc=STRING

Sobreescribe la biblioteca C `libc` a *STRING* (el valor predeterminado es dependiente del sistema).

--with-openssl=DIR

Raíz del directorio OpenSSL.

Added in version 3.7.

--with-openssl-rpath=[no|auto|DIR]

Configura el directorio de la biblioteca en tiempo de ejecución (rpath) para las bibliotecas OpenSSL:

- no (por defecto): no establece rpath;
- auto: autodetecta rpath desde `--with-openssl` y `pkg-config`;
- *DIR*: establece un rpath explícito.

Added in version 3.10.

3.3.12 Opciones de seguridad

--with-hash-algorithm=[fnv|siphash13|siphash24]

Selecciona el algoritmo hash para usar en `Python/pyhash.c`:

- siphash13 (por defecto);
- siphash24;
- fnv.

Added in version 3.4.

Added in version 3.11: Se agrega `siphash13` y es el nuevo valor predeterminado.

--with-builtin-hashlib-hashes=md5,sha1,sha256,sha512,sha3,blake2

Módulos hash incorporados:

- md5;
- sha1;
- sha256;
- sha512;
- sha3 (con shake);
- blake2.

Added in version 3.9.

--with-ssl-default-suites=[python|openssl|STRING]

Sobreescribe la cadena de conjuntos de cifrado predeterminada de OpenSSL:

- python (por defecto): usa la selección principal de Python;
- openssl: deja intactos los valores predeterminados de OpenSSL;
- *STRING*: usa una cadena de caracteres personalizada

Consultar el módulo `ssl`.

Added in version 3.7.

Distinto en la versión 3.10: Las configuraciones `python` y *STRING* también establecen TLS 1.2 como versión mínima del protocolo.

--disable-safety

Disable compiler options that are [recommended by OpenSSF](#) for security reasons with no performance overhead. If this option is not enabled, CPython will be built based on safety compiler options with no slow down. When this option is enabled, CPython will not be built with the compiler options listed below.

The following compiler options are disabled with `--disable-safety`:

- `-fstack-protector-strong`: Enable run-time checks for stack-based buffer overflows.
- `-Wtrampolines`: Enable warnings about trampolines that require executable stacks.

Added in version 3.14.

`--enable-slower-safety`

Enable compiler options that are [recommended by OpenSSF](#) for security reasons which require overhead. If this option is not enabled, CPython will not be built based on safety compiler options which performance impact. When this option is enabled, CPython will be built with the compiler options listed below.

The following compiler options are enabled with `--enable-slower-safety`:

- `-D_FORTIFY_SOURCE=3`: Fortify sources with compile- and run-time checks for unsafe libc usage and buffer overflows.

Added in version 3.14.

3.3.13 Opciones macOS

See [Mac/README.rst](#).

`--enable-universalsdk`

`--enable-universalsdk=SDKDIR`

Crea una compilación binaria universal. *SDKDIR* especifica qué macOS SDK debe usarse para realizar la compilación (el valor predeterminado es no).

`--enable-framework`

`--enable-framework=INSTALLDIR`

Crear un `Python.framework` en lugar de una instalación Unix tradicional. Opcionalmente *INSTALLDIR* especifica la ruta de instalación (el valor predeterminado es no).

`--with-universal-archs=ARCH`

Especifique el tipo de binario universal que se debe crear. Esta opción solo es válida cuando se establece `--enable-universalsdk`.

Opciones:

- `universal2` (x86-64 and arm64);
- `32-bit` (PPC and i386);
- `64-bit` (PPC64 and x86-64);
- `3-way` (i386, PPC and x86-64);
- `intel` (i386 and x86-64);
- `intel-32` (i386);
- `intel-64` (x86-64);
- `all` (PPC, i386, PPC64 and x86-64).

Note that values for this configuration item are *not* the same as the identifiers used for universal binary wheels on macOS. See the Python Packaging User Guide for details on the [packaging platform compatibility tags used on macOS](#)

`--with-framework-name=FRAMEWORK`

Especifica el nombre del framework de Python en macOS, solo es válido cuando `--enable-framework` está configurada (por defecto: `Python`).

`--with-app-store-compliance`

--with-app-store-compliance=PATCH-FILE

The Python standard library contains strings that are known to trigger automated inspection tool errors when submitted for distribution by the macOS and iOS App Stores. If enabled, this option will apply the list of patches that are known to correct app store compliance. A custom patch file can also be specified. This option is disabled by default.

Added in version 3.13.

3.3.14 iOS Options

See [iOS/README.rst](#).

--enable-framework=INSTALLDIR

Create a Python.framework. Unlike macOS, the *INSTALLDIR* argument specifying the installation path is mandatory.

--with-framework-name=FRAMEWORK

Specify the name for the framework (default: Python).

3.3.15 Opciones de compilación cruzada

La compilación cruzada, también conocida como construcción cruzada, se puede usar para construir Python para otra plataforma o arquitectura de CPU. La compilación cruzada requiere un intérprete de Python para la plataforma de compilación. La versión de Python de compilación debe coincidir con la versión de Python host de compilación cruzada.

--build=BUILD

configure para construir en BUILD, generalmente adivinado por `config.guess`.

--host=HOST

compilación cruzada para crear programas que se ejecuten en HOST (plataforma de destino)

--with-build-python=path/to/python

ruta para construir el binario `python` para compilación cruzada

Added in version 3.11.

CONFIG_SITE=file

Una variable de entorno que apunta a un archivo con anulaciones de configuración.

Example *config.site* file:

```
# config.site-aarch64
ac_cv_buggy_getaddrinfo=no
ac_cv_file__dev_ptmx=yes
ac_cv_file__dev_ptc=no
```

HOSTRUNNER

Program to run CPython for the host platform for cross-compilation.

Added in version 3.11.

Ejemplo de compilación cruzada:

```
CONFIG_SITE=config.site-aarch64 ../configure \
--build=x86_64-pc-linux-gnu \
--host=aarch64-unknown-linux-gnu \
--with-build-python=../x86_64/python
```

3.4 Sistema de compilación Python

3.4.1 Archivos principales del sistema de compilación

- `configure.ac` => `configure`;
- `Makefile.pre.in` => `Makefile` (creado por `configure`);
- `pyconfig.h` (creado por `configure`);
- `Modules/Setup`: Extensiones C creadas por `Makefile` usando el script de shell `Module/makesetup`;

3.4.2 Pasos principales de compilación

- Los archivos C (`.c`) se crean como archivos objeto (`.o`).
- La biblioteca estática `libpython(.a)` se crea a a partir de archivos de objetos.
- `python.o` y la biblioteca estática `libpython` están enlazadas al programa final `python`.
- Las extensiones C son creadas por `Makefile` (ver `Modules/Setup`).

3.4.3 Objetivos principales de Makefile

make

For the most part, when rebuilding after editing some code or refreshing your checkout from upstream, all you need to do is execute `make`, which (per Make's semantics) builds the default target, the first one defined in the `Makefile`. By tradition (including in the CPython project) this is usually the `all` target. The `configure` script expands an `autoconf` variable, `@DEF_MAKE_ALL_RULE@` to describe precisely which targets `make all` will build. The three choices are:

- `profile-opt` (configured with `--enable-optimizations`)
- `build_wasm` (chosen if the host platform matches `wasm32-wasi*` or `wasm32-emscripten`)
- `build_all` (configured without explicitly using either of the others)

Depending on the most recent source file changes, Make will rebuild any targets (object files and executables) deemed out-of-date, including running `configure` again if necessary. Source/target dependencies are many and maintained manually however, so Make sometimes doesn't have all the information necessary to correctly detect all targets which need to be rebuilt. Depending on which targets aren't rebuilt, you might experience a number of problems. If you have build or test problems which you can't otherwise explain, `make clean && make` should work around most dependency problems, at the expense of longer build times.

make platform

Build the `python` program, but don't build the standard library extension modules. This generates a file named `platform` which contains a single line describing the details of the build platform, e.g., `macosx-14.3-arm64-3.12` or `linux-x86_64-3.13`.

make profile-opt

Build Python using profile-guided optimization (PGO). You can use the `configure --enable-optimizations` option to make this the default target of the `make` command (`make all` or just `make`).

make clean

Remove built files.

make distclean

In addition to the work done by `make clean`, remove files created by the configure script. `configure` will have to be run before building again.¹

make install

Build the `all` target and install Python.

make test

Build the `all` target and run the Python test suite with the `--fast-ci` option without GUI tests. Variables:

- `TESTOPTS`: additional regrtest command-line options.
- `TESTPYTHONOPTS`: additional Python command-line options.
- `TESTTIMEOUT`: timeout in seconds (default: 10 minutes).

make ci

This is similar to `make test`, but uses the `-ugui` to also run GUI tests.

Added in version 3.14.

make buildbottest

This is similar to `make test`, but uses the `--slow-ci` option and default timeout of 20 minutes, instead of `--fast-ci` option.

make regen-all

Regenerate (almost) all generated files. These include (but are not limited to) bytecode cases, and parser generator file. `make regen-stdlib-module-names` and `autoconf` must be run separately for the remaining *generated files*.

3.4.4 Extensiones C

Some C extensions are built as built-in modules, like the `sys` module. They are built with the `Py_BUILD_CORE_BUILTIN` macro defined. Built-in modules have no `__file__` attribute:

```
>>> import sys
>>> sys
<module 'sys' (built-in)>
>>> sys.__file__
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'sys' has no attribute '__file__'
```

Other C extensions are built as dynamic libraries, like the `_asyncio` module. They are built with the `Py_BUILD_CORE_MODULE` macro defined. Example on Linux x86-64:

```
>>> import _asyncio
>>> _asyncio
<module '_asyncio' from '/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_
↳ 64-linux-gnu.so'>
>>> _asyncio.__file__
'/usr/lib64/python3.9/lib-dynload/_asyncio.cpython-39-x86_64-linux-gnu.so'
```

¹ `git clean -fdx` is an even more extreme way to «clean» your checkout. It removes all files not known to Git. When bug hunting using `git bisect`, this is *recommended between probes* to guarantee a completely clean build. **Use with care**, as it will delete all files not checked into Git, including your new, uncommitted work.

`Modules/Setup` se usa para generar objetivos Makefile para compilar extensiones C. Al principio de los archivos, las extensiones C se crean como módulos incorporados. Las extensiones definidas después del marcador `*shared*` se crean como bibliotecas dinámicas.

Las macros `PyAPI_FUNC()`, `PyAPI_DATA()` y `PyMODINIT_FUNC` de `Include/exports.h` se definen de manera diferente dependiendo si es definida la macro `Py_BUILD_CORE_MODULE`:

- Use `Py_EXPORTED_SYMBOL` si `Py_BUILD_CORE_MODULE` es definido
- Use `Py_IMPORTED_SYMBOL` de lo contrario.

Si la macro `Py_BUILD_CORE_BUILTIN` se usa por error en una extensión de C compilada como una biblioteca compartida, su función `PyInit_xxx()` no se exporta, provocando un `ImportError` en la importación.

3.5 Banderas de compilador y vinculación

Opciones establecidas por el script `./configure` y las variables de entorno y utilizadas por Makefile.

3.5.1 Banderas del preprocesador

CONFIGURE_CPPFLAGS

Valor de la variable `CPPFLAGS` pasado al script `./configure`.

Added in version 3.6.

CPPFLAGS

(Objetivo) Indicadores del preprocesador C/C++, p. ej. `-Iinclude_dir` si tiene encabezados en un directorio no estándar `include_dir`.

Ambos `CPPFLAGS` y `LDFLAGS` necesitan contener el valor del shell para poder compilar módulos de extensión usando los directorios especificados en las variables de entorno.

BASECPPFLAGS

Added in version 3.4.

PY_CPPFLAGS

Se agregaron indicadores de preprocesador adicionales para construir los archivos de objeto del intérprete.

Por defecto: `$(BASECPPFLAGS) -I. -I$(srcdir)/Include $(CONFIGURE_CPPFLAGS) $(CPPFLAGS)`.

Added in version 3.2.

3.5.2 Banderas del compilador

CC

Comando del compilador C.

Ejemplo: `gcc -pthread`.

CXX

Comando del compilador de C++.

Ejemplo: `g++ -pthread`.

CFLAGS

Banderas del compilador de C.

CFLAGS_NODIST

`CFLAGS_NODIST` se usa para compilar el intérprete y las extensiones `stdlib` C. Úselo cuando un indicador del compilador *no* deba ser parte de `CFLAGS` una vez que Python esté instalado ([gh-65320](#)).

En particular, `CFLAGS` no debe contener:

- el indicador del compilador `-I` (para configurar la ruta de búsqueda de archivos de inclusión). Los indicadores `-I` se procesan de izquierda a derecha, y cualquier indicador en `CFLAGS` tendrá prioridad sobre los indicadores `-I` proporcionados por el usuario y el paquete.
- banderas de endurecimiento como `-Werror` porque las distribuciones no pueden controlar si los paquetes instalados por los usuarios cumplen con estándares tan elevados.

Added in version 3.5.

COMPILEALL_OPTS

Las opciones pasadas a la línea de comando `compileall` al crear archivos PYC en `make install`. El valor predeterminado: `-j0`.

Added in version 3.12.

EXTRA_CFLAGS

Banderas adicionales del compilador de C.

CONFIGURE_CFLAGS

Valor de la variable `CFLAGS` pasada al script `./configure`.

Added in version 3.2.

CONFIGURE_CFLAGS_NODIST

Valor de la variable `CFLAGS_NODIST` pasada al script `./configure`.

Added in version 3.5.

BASECFLAGS

Banderas base del compilador.

OPT

Banderas de optimización.

CFLAGS_ALIASING

Banderas de alias estrictos o no estrictos que se utilizan para compilar `Python/dtoa.c`.

Added in version 3.7.

CCSHARED

Banderas del compilador que se utilizan para compilar una biblioteca compartida.

Por ejemplo, `-fPIC` se usa en Linux y BSD.

CFLAGSFORSHARED

Se agregaron banderas C adicionales para compilar los archivos de objeto del intérprete.

Por defecto: `$(CCSHARED)` cuando se usa `--enable-shared`, o una cadena de caracteres vacía en caso contrario.

PY_CFLAGS

Por defecto: `$(BASECFLAGS) $(OPT) $(CONFIGURE_CFLAGS) $(CFLAGS) $(EXTRA_CFLAGS)`.

PY_CFLAGS_NODIST

Por defecto: `$(CONFIGURE_CFLAGS_NODIST) $(CFLAGS_NODIST) -I$(srcdir)/Include/internal`.

Added in version 3.5.

PY_STDMODULE_CFLAGS

Banderas de C que se utilizan para compilar los archivos de objeto del intérprete.

Por defecto: `$(PY_CFLAGS) $(PY_CFLAGS_NODIST) $(PY_CPPFLAGS) $(CFLAGSFORSHARED)`.

Added in version 3.7.

PY_CORE_CFLAGS

Por defecto: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE`.

Added in version 3.2.

PY_BUILTIN_MODULE_CFLAGS

Banderas del compilador para construir un módulo de extensión de biblioteca estándar como un módulo incorporado, como el módulo `posix`.

Por defecto: `$(PY_STDMODULE_CFLAGS) -DPy_BUILD_CORE_BUILTIN`.

Added in version 3.8.

PURIFY

Comando Purify. Purify es un programa de depuración de memoria.

Por defecto: cadena de caracteres vacía (no utilizado).

3.5.3 Banderas de vinculación

LINKCC

Comando de vinculación usado para compilar programas como `python` y `_testembed`.

Por defecto: `$(PURIFY) $(CC)`.

CONFIGURE_LDFLAGS

Valor de la variable `LDFLAGS` pasada al script `./configure`.

Evite asignar `CFLAGS`, `LDFLAGS`, etc. así los usuarios pueden usarlos en la línea de comando para agregar estos valores sin pisar los valores preestablecidos.

Added in version 3.2.

LDFLAGS_NODIST

`LDFLAGS_NODIST` se usa de la misma manera que `CFLAGS_NODIST`. Usar cuando un indicador del enlazador *no* deba ser parte de `LDFLAGS` una vez que Python esté instalado ([gh-65320](#)).

En particular, `LDFLAGS` no debe contener:

- el indicador del compilador `-L` (para establecer la ruta de búsqueda de bibliotecas). Los indicadores `-L` se procesan de izquierda a derecha, y cualquier indicador en `LDFLAGS` tendrá prioridad sobre los indicadores `-L` proporcionados por el usuario y el paquete.

CONFIGURE_LDFLAGS_NODIST

Valor de la variable `LDFLAGS_NODIST` pasado al script `./configure`.

Added in version 3.8.

LDFLAGS

Indicadores de vinculación, p. ej. `-Llib_dir` si tiene bibliotecas en un directorio no estándar `lib_dir`.

Ambos `CPPFLAGS` y `LDFLAGS` necesitan contener el valor del shell para poder compilar módulos de extensión usando los directorios especificados en las variables de entorno.

LIBS

Banderas de vinculación para pasar bibliotecas al vinculador al enlazar el ejecutable de Python.

Ejemplo: `-lrt`.

LD_SHARED

Comando para construir una biblioteca compartida.

Por defecto: `@LD_SHARED@ $(PY_LDFLAGS)`.

BLDSHARED

Comando para compilar la biblioteca compartida `libpython`.

Por defecto: `@BLDSHARED@ $(PY_CORE_LDFLAGS)`.

PY_LDFLAGS

Por defecto: `$(CONFIGURE_LDFLAGS) $(LDFLAGS)`.

PY_LDFLAGS_NODIST

Por defecto: `$(CONFIGURE_LDFLAGS_NODIST) $(LDFLAGS_NODIST)`.

Added in version 3.8.

PY_CORE_LDFLAGS

Banderas de vinculación que se utilizan para crear los archivos de objeto del intérprete.

Added in version 3.8.

Uso de Python en Windows

Este documento pretende dar una visión general del comportamiento específico de Windows que se debería conocer al usar Python en Microsoft Windows.

Unlike most Unix systems and services, Windows does not include a system supported installation of Python. Instead, Python can be obtained from a number of distributors, including directly from the CPython team. Each Python distribution will have its own benefits and drawbacks, however, consistency with other tools you are using is generally a worthwhile benefit. Before committing to the process described here, we recommend investigating your existing tools to see if they can provide Python directly.

To obtain Python from the CPython team, use the Python Install Manager. This is a standalone tool that makes Python available as global commands on your Windows machine, integrates with the system, and supports updates over time. You can download the Python Install Manager from python.org/downloads or through the [Microsoft Store app](#).

Once you have installed the Python Install Manager, the global `python` command can be used from any terminal to launch your current latest version of Python. This version may change over time as you add or remove different versions, and the `py list` command will show which is current.

In general, we recommend that you create a virtual environment for each project and run `<env>\Scripts\Activate` in your terminal to use it. This provides isolation between projects, consistency over time, and ensures that additional commands added by packages are also available in your session. Create a virtual environment using `python -m venv <env path>`.

If the `python` or `py` commands do not seem to be working, please see the [Troubleshooting](#) section below. There are sometimes additional manual steps required to configure your PC.

Apart from using the Python install manager, Python can also be obtained as NuGet packages. See [El paquete de nuget.org](#) below for more information on these packages.

The embeddable distros are minimal packages of Python suitable for embedding into larger applications. They can be installed using the Python install manager. See [El paquete incrustable](#) below for more information on these packages.

4.1 Python Install Manager

4.1.1 Installation

The Python install manager can be installed from the [Microsoft Store app](#) or downloaded and installed from python.org/downloads. The two versions are identical.

To install through the Store, simply click «Install». After it has completed, open a terminal and type `python` to get started.

To install the file downloaded from python.org, either double-click and select «Install», or run `Add-AppxPackage <path to MSIX>` in Windows Powershell.

After installation, the `python`, `py`, and `pymanager` commands should be available. If you have existing installations of Python, or you have modified your `PATH` variable, you may need to remove them or undo the modifications. See [Troubleshooting](#) for more help with fixing non-working commands.

When you first install a runtime, you will likely be prompted to add a directory to your `PATH`. This is optional, if you prefer to use the `py` command, but is offered for those who prefer the full range of aliases (such as `python3.14.exe`) to be available. The directory will be `%LocalAppData%\Python\bin` by default, but may be customized by an administrator. Click Start and search for «Edit environment variables for your account» for the system settings page to add the path.

Each Python runtime you install will have its own directory for scripts. These also need to be added to `PATH` if you want to use them.

The Python install manager will be automatically updated to new releases. This does not affect any installs of Python runtimes. Uninstalling the Python install manager does not uninstall any Python runtimes.

If you are not able to install an MSIX in your context, for example, you are using automated deployment software that does not support it, or are targeting Windows Server 2019, please see [Advanced Installation](#) below for more information.

4.1.2 Basic Use

The recommended command for launching Python is `python`, which will either launch the version requested by the script being launched, an active virtual environment, or the default installed version, which will be the latest stable release unless configured otherwise. If no version is specifically requested and no runtimes are installed at all, the current latest release will be installed automatically.

For all scenarios involving multiple runtime versions, the recommended command is `py`. This may be used anywhere in place of `python` or the older `py.exe` launcher. By default, `py` matches the behaviour of `python`, but also allows command line options to select a specific version as well as subcommands to manage installations. These are detailed below.

Because the `py` command may already be taken by the previous version, there is also an unambiguous `pymanager` command. Scripted installs that are intending to use Python install manager should consider using `pymanager`, due to the lower chance of encountering a conflict with existing installs. The only difference between the two commands is when running without any arguments: `py` will install and launch your default interpreter, while `pymanager` will display help (`pymanager exec ...` provides equivalent behaviour to `py ...`).

Each of these commands also has a windowed version that avoids creating a console window. These are `pyw`, `pythonw` and `pymanagerw`. A `python3` command is also included that mimics the `python` command. It is intended to catch accidental uses of the typical POSIX command on Windows, but is not meant to be widely used or recommended.

To launch your default runtime, run `python` or `py` with the arguments you want to be passed to the runtime (such as script files or the module to launch):

```
$> py
...
$> python my-script.py
...
$> py -m this
...
```

The default runtime can be overridden with the `PYTHON_MANAGER_DEFAULT` environment variable, or a configuration file. See [Configuration](#) for information about configuration settings.

To launch a specific runtime, the `py` command accepts a `-V:<TAG>` option. This option must be specified before any others. The tag is part or all of the identifier for the runtime; for those from the CPython team, it looks like the version, potentially with the platform. For compatibility, the `V:` may be omitted in cases where the tag refers to an official release and starts with 3.

```
$> py -V:3.14 ...
$> py -V:3-arm64 ...
```

Runtimes from other distributors may require the *company* to be included as well. This should be separated from the tag by a slash, and may be a prefix. Specifying the company is optional when it is `PythonCore`, and specifying the tag is optional (but not the slash) when you want the latest release from a specific company.

```
$> py -V:Distributor\1.0 ...
$> py -V:distrib/ ...
```

If no version is specified, but a script file is passed, the script will be inspected for a *shebang line*. This is a special format for the first line in a file that allows overriding the command. See [Shebang lines](#) for more information. When there is no shebang line, or it cannot be resolved, the script will be launched with the default runtime.

If you are running in an active virtual environment, have not requested a particular version, and there is no shebang line, the default runtime will be that virtual environment. In this scenario, the `python` command was likely already overridden and none of these checks occurred. However, this behaviour ensures that the `py` command can be used interchangeably.

When you launch either `python` or `py` but do not have any runtimes installed, and the requested version is the default, it will be installed automatically and then launched. Otherwise, the requested version will be installed if automatic installation is configured (most likely by setting `PYTHON_MANAGER_AUTOMATIC_INSTALL` to `true`), or if the `py` `exec` or `pymanager exec` forms of the command were used.

4.1.3 Command Help

The `py help` command will display the full list of supported commands, along with their options. Any command may be passed the `-?` option to display its help, or its name passed to `py help`.

```
$> py help
$> py help install
$> py install /?
```

All commands support some common options, which will be shown by `py help`. These options must be specified after any subcommand. Specifying `-v` or `--verbose` will increase the amount of output shown, and `-vv` will increase it further for debugging purposes. Passing `-q` or `--quiet` will reduce output, and `-qq` will reduce it further.

The `--config=<PATH>` option allows specifying a configuration file to override multiple settings at once. See [Configuration](#) below for more information about these files.

4.1.4 Listing Runtimes

```
$> py list [-f|--format=<FMT>] [-1|--one] [--online|-s|--source=<URL>] [<TAG>...]
```

The list of installed runtimes can be seen using `py list`. A filter may be added in the form of one or more tags (with or without company specifier), and each may include a `<`, `<=`, `>=` or `>` prefix to restrict to a range.

A range of formats are supported, and can be passed as the `--format=<FMT>` or `-f <FMT>` option. Formats include `table` (a user friendly table view), `csv` (comma-separated table), `json` (a single JSON blob), `jsonl` (one JSON blob per result), `exe` (just the executable path), `prefix` (just the prefix path).

The `--one` or `-1` option only displays a single result. If the default runtime is included, it will be the one. Otherwise, the `<best>` result is shown (`<best>` is deliberately vaguely defined, but will usually be the most recent version). The result shown by `py list --one <TAG>` will match the runtime that would be launched by `py -V:<TAG>`.

The `--only-managed` option excludes results that were not installed by the Python install manager. This is useful when determining which runtimes may be updated or uninstalled through the `py` command.

The `--online` option is short for passing `--source=<URL>` with the default source. Passing either of these options will search the online index for runtimes that can be installed. The result shown by `py list --online --one <TAG>` will match the runtime that would be installed by `py install <TAG>`.

```
$> py list --online 3.14
```

For compatibility with the old launcher, the `--list`, `--list-paths`, `-0` and `-0p` commands (e.g. `py -0p`) are retained. They do not allow additional options, and will produce legacy formatted output.

4.1.5 Installing Runtimes

```
$> py install [-s|--source=<URL>] [-f|--force] [-u|--update] [--dry-run] [<TAG>...  
→]
```

New runtime versions may be added using `py install`. One or more tags may be specified, and the special tag `default` may be used to select the default. Ranges are not supported for installation.

The `--source=<URL>` option allows overriding the online index that is used to obtain runtimes. This may be used with an offline index, as shown in *Offline Installs*.

Passing `--force` will ignore any cached files and remove any existing install to replace it with the specified one.

Passing `--update` will replace existing installs if the new version is newer. Otherwise, they will be left. If no tags are provided with `--update`, all installs managed by the Python install manager will be updated if newer versions are available. Updates will remove any modifications made to the install, including globally installed packages, but virtual environments will continue to work.

Passing `--dry-run` will generate output and logs, but will not modify any installs.

In addition to the above options, the `--target` option will extract the runtime to the specified directory instead of doing a normal install. This is useful for embedding runtimes into larger applications.

```
$> py install ... [-t|--target=<PATH>] <TAG>
```

4.1.6 Offline Installs

To perform offline installs of Python, you will need to first create an offline index on a machine that has network access.

```
$> py install --download=<PATH> ... <TAG>...
```

The `--download=<PATH>` option will download the packages for the listed tags and create a directory containing them and an `index.json` file suitable for later installation. This entire directory can be moved to the offline machine and used to install one or more of the bundled runtimes:

```
$> py install --source="<PATH>\index.json" <TAG>...
```

The Python install manager can be installed by downloading its installer and moving it to another machine before installing.

Alternatively, the ZIP files in an offline index directory can simply be transferred to another machine and extracted. This will not register the install in any way, and so it must be launched by directly referencing the executables in the extracted directory, but it is sometimes a preferable approach in cases where installing the Python install manager is not possible or convenient.

In this way, Python runtimes can be installed and managed on a machine without access to the internet.

4.1.7 Uninstalling Runtimes

```
$> py uninstall [-y|--yes] <TAG>...
```

Runtimes may be removed using the `py uninstall` command. One or more tags must be specified. Ranges are not supported here.

The `--yes` option bypasses the confirmation prompt before uninstalling.

Instead of passing tags individually, the `--purge` option may be specified. This will remove all runtimes managed by the Python install manager, including cleaning up the Start menu, registry, and any download caches. Runtimes that were not installed by the Python install manager will not be impacted, and neither will manually created configuration files.

```
$> py uninstall [-y|--yes] --purge
```

The Python install manager can be uninstalled through the Windows «Installed apps» settings page. This does not remove any runtimes, and they will still be usable, though the global `python` and `py` commands will be removed. Reinstalling the Python install manager will allow you to manage these runtimes again. To completely clean up all Python runtimes, run with `--purge` before uninstalling the Python install manager.

4.1.8 Configuration

Python install manager is configured with a hierarchy of configuration files, environment variables, command-line options, and registry settings. In general, configuration files have the ability to configure everything, including the location of other configuration files, while registry settings are administrator-only and will override configuration files. Command-line options override all other settings, but not every option is available.

This section will describe the defaults, but be aware that modified or overridden installs may resolve settings differently.

A global configuration file may be configured by an administrator, and would be read first. The user configuration file is stored at `%AppData%\Python\pymanager.json` (by default) and is read next, overwriting any settings from earlier files. An additional configuration file may be specified as the `PYTHON_MANAGER_CONFIG` environment variable or the `--config` command line option (but not both).

The following settings are those that are considered likely to be modified in normal use. Later sections list those that are intended for administrative customization.

Standard configuration options

Config Key	Environment Variable	Descripción
<code>default_t</code>	<code>PYTHON_MANAGER_DEFAULT_TAG</code>	The preferred default version to launch or install. By default, this is interpreted as the most recent non-prerelease version from the CPython team.
<code>default_p</code>	<code>PYTHON_MANAGER_DEFAULT_PLATFORM</code>	The preferred default platform to launch or install. This is treated as a suffix to the specified tag, such that <code>py -V:3.14</code> would prefer an install for <code>3.14-64</code> if it exists (and <code>default_platform</code> is <code>-64</code>), but will use <code>3.14</code> if no tagged install exists.
<code>logs_dir</code>	<code>PYTHON_MANAGER_LOGS_DIR</code>	The location where log files are written. By default, <code>%TEMP%</code> .
<code>automatic</code>	<code>PYTHON_MANAGER_AUTOMATIC</code>	True to allow automatic installs when specifying a particular runtime to launch. By default, true.
<code>include_v</code>	<code>PYTHON_MANAGER_INCLUDE_V</code>	True to allow listing and launching runtimes that were not installed by the Python install manager, or false to exclude them. By default, true.
<code>shebang_c</code>	<code>PYTHON_MANAGER_SHEBANG_C</code>	True to allow shebangs in <code>.py</code> files to launch applications other than Python runtimes, or false to prevent it. By default, true.
<code>log_level</code>	<code>PYMANAGER_VERBOSE</code> <code>PYMANAGER_DEBUG</code>	Set the default level of output (0-50). By default, 20. Lower values produce more output. The environment variables are boolean, and may produce additional output during startup that is later suppressed by other configuration.
<code>confirm</code>	<code>PYTHON_MANAGER_CONFIRM</code>	True to confirm certain actions before taking them (such as uninstall), or false to skip the confirmation. By default, true.
<code>install_source</code>	<code>PYTHON_MANAGER_INSTALL_SOURCE</code>	Override the index feed to obtain new installs from.
<code>list.format</code>	<code>PYTHON_MANAGER_LIST_FORMAT</code>	Specify the default format used by the <code>py list</code> command. By default, <code>table</code> .

Dotted names should be nested inside JSON objects, for example, `list.format` would be specified as `{"list":`

```
{"format": "table"}.
```

4.1.9 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. The `python` and `py` commands allow the same facilities to be used with Python scripts on Windows.

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

- `/usr/bin/env <ALIAS>`
- `/usr/bin/env -S <ALIAS>`
- `/usr/bin/<ALIAS>`
- `/usr/local/bin/<ALIAS>`
- `<ALIAS>`

Por ejemplo, si la primera línea del script comienza con

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

The default Python or an active virtual environment 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 have `<ALIAS>` replaced by an alias from an installed runtime. That is, any command generated in the global aliases directory (which you may have added to your `PATH` environment variable) can be used in a shebang, even if it is not on your `PATH`. This allows the use of shebangs like `/usr/bin/python3.12` to select a particular runtime.

If no runtimes are installed, or if automatic installation is enabled, the requested runtime will be installed if necessary. See [Configuration](#) for information about configuration settings.

The `/usr/bin/env` form of shebang line will also search the `PATH` environment variable for unrecognized commands. This corresponds to the behaviour of the Unix `env` program, which performs the same search, but prefers launching known Python commands. A warning may be displayed when searching for arbitrary executables, and this search may be disabled by the `shebang_can_run_anything` configuration option.

Shebang lines that do not match any of patterns are treated as *Windows* executable paths that are absolute or relative to the directory containing the script file. This is a convenience for Windows-only scripts, such as those generated by an installer, since the behavior is not compatible with Unix-style shells. These paths may be quoted, and may include multiple arguments, after which the path to the script and any additional arguments will be appended. This functionality may be disabled by the `shebang_can_run_anything` configuration option.

Nota

The behaviour of shebangs in the Python install manager is subtly different from the previous `py.exe` launcher, and the old configuration options no longer apply. If you are specifically reliant on the old behaviour or configuration, we recommend keeping the legacy launcher. It may be [downloaded independently](#) and installed on its own. The legacy launcher's `py` command will override PyManager's one, and you will need to use `pymanager` commands for installing and uninstalling.

4.1.10 Advanced Installation

For situations where an MSIX cannot be installed, such as some older administrative distribution platforms, there is an MSI available from the [python.org](#) downloads page. This MSI has no user interface, and can only perform per-machine installs to its default location in Program Files. It will attempt to modify the system `PATH` environment variable to include this install location, but be sure to validate this on your configuration.

i Nota

Windows Server 2019 is the only version of Windows that CPython supports that does not support MSIX. For Windows Server 2019, you should use the MSI.

Be aware that the MSI package does not bundle any runtimes, and so is not suitable for installs into offline environments without also creating an offline install index. See [Offline Installs](#) and [Administrative Configuration](#) for information on handling these scenarios.

Runtimes installed by the MSI are shared with those installed by the MSIX, and are all per-user only. The Python install manager does not support installing runtimes per-machine. To emulate a per-machine install, you can use `py install --target=<shared location>` as administrator and add your own system-wide modifications to PATH, the registry, or the Start menu.

When the MSIX is installed, but commands are not available in the PATH environment variable, they can be found under `%LocalAppData%\Microsoft\WindowsApps\PythonSoftwareFoundation.PythonManager_3847v3x7pw1km` or `%LocalAppData%\Microsoft\WindowsApps\PythonSoftwareFoundation.PythonManager_qbz5n2kfra8p0`, depending on whether it was installed from python.org or through the Windows Store. Attempting to run the executable directly from Program Files is not recommended.

To programmatically install the Python install manager, it is easiest to use WinGet, which is included with all supported versions of Windows:

```
$> winget install 9NQ7512CXL7T -e --accept-package-agreements --disable-
↪interactivity

# Optionally run the configuration checker and accept all changes
$> py install --configure -y
```

To download the Python install manager and install on another machine, the following WinGet command will download the required files from the Store to your Downloads directory (add `-d <location>` to customize the output location). This also generates a YAML file that appears to be unnecessary, as the downloaded MSIX can be installed by launching or using the commands below.

```
$> winget download 9NQ7512CXL7T -e --skip-license --accept-package-agreements --
↪accept-source-agreements
```

To programmatically install or uninstall an MSIX using only PowerShell, the [Add-AppxPackage](#) and [Remove-AppxPackage](#) PowerShell cmdlets are recommended:

```
$> Add-AppxPackage C:\Downloads\python-manager-25.0.msix
...
$> Get-AppxPackage PythonSoftwareFoundation.PythonManager | Remove-AppxPackage
```

The latest release can be downloaded and installed by Windows by passing the AppInstaller file to the `Add-AppxPackage` command. This installs using the MSIX on python.org, and is only recommended for cases where installing via the Store (interactively or using WinGet) is not possible.

```
$> Add-AppxPackage -AppInstallerFile https://www.python.org/ftp/python/pymanager/
↪pymanager.appinstaller
```

Other tools and APIs may also be used to provision an MSIX package for all users on a machine, but Python does not consider this a supported scenario. We suggest looking into the PowerShell [Add-AppxProvisionedPackage](#) cmdlet, the native Windows [PackageManager](#) class, or the documentation and support for your deployment tool.

Regardless of the install method, users will still need to install their own copies of Python itself, as there is no way to trigger those installs without being a logged in user. When using the MSIX, the latest version of Python will be available for all users to install without network access.

Note that the MSIX downloadable from the Store and from the Python website are subtly different and cannot be installed at the same time. Wherever possible, we suggest using the above WinGet commands to download the package from the Store to reduce the risk of setting up conflicting installs. There are no licensing restrictions on the Python install manager that would prevent using the Store package in this way.

4.1.11 Administrative Configuration

There are a number of options that may be useful for administrators to override configuration of the Python install manager. These can be used to provide local caching, disable certain shortcut types, override bundled content. All of the above configuration options may be set, as well as those below.

Configuration options may be overridden in the registry by setting values under `HKEY_LOCAL_MACHINE\Software\Policies\Python\PyManager`, where the value name matches the configuration key and the value type is `REG_SZ`. Note that this key can itself be customized, but only by modifying the core config file distributed with the Python install manager. We recommend, however, that registry values are used only to set `base_config` to a JSON file containing the full set of overrides. Registry key overrides will replace any other configured setting, while `base_config` allows users to further modify settings they may need.

Note that most settings with environment variables support those variables because their default setting specifies the variable. If you override them, the environment variable will no longer work, unless you override it with another one. For example, the default value of `confirm` is literally `%PYTHON_MANAGER_CONFIRM%`, which will resolve the variable at load time. If you override the value to `yes`, then the environment variable will no longer be used. If you override the value to `%CONFIRM%`, then that environment variable will be used instead.

Configuration settings that are paths are interpreted as relative to the directory containing the configuration file that specified them.

Administrative configuration options

Config Key	Descripción
<code>base_config</code>	The highest priority configuration file to read. Note that only the built-in configuration file and the registry can modify this setting.
<code>user_config</code>	The second configuration file to read.
<code>additional_config</code>	The third configuration file to read.
<code>registry_override_key</code>	Registry location to check for overrides. Note that only the built-in configuration file can modify this setting.
<code>bundled_dir</code>	Read-only directory containing locally cached files.
<code>install.fallback_source</code>	Path or URL to an index to consult when the main index cannot be accessed.
<code>install.enable_shortcut_kinds</code>	Comma-separated list of shortcut kinds to allow (e.g. "pep514,start"). Enabled shortcuts may still be disabled by <code>disable_shortcut_kinds</code> .
<code>install.disable_shortcut_kinds</code>	Comma-separated list of shortcut kinds to exclude (e.g. "pep514,start"). Disabled shortcuts are not reactivated by <code>enable_shortcut_kinds</code> .
<code>pep514_root</code>	Registry location to read and write PEP 514 entries into. By default, <code>HKEY_CURRENT_USER\Software\Python</code> .
<code>start_folder</code>	Start menu folder to write shortcuts into. By default, <code>Python</code> . This path is relative to the user's Programs folder.
<code>virtual_env</code>	Path to the active virtual environment. By default, this is <code>%VIRTUAL_ENV%</code> , but may be set empty to disable venv detection.
<code>shebang_can_run_anything</code>	True to suppress visible warnings when a shebang launches an application other than a Python runtime.

4.1.12 Installing Free-threaded Binaries

Added in version 3.13: (Experimental)

i Nota

Everything described in this section is considered experimental, and should be expected to change in future releases.

Pre-built distributions of the experimental free-threaded build are available by installing tags with the `t` suffix.

```
$> py install 3.14t
$> py install 3.14t-arm64
$> py install 3.14t-32
```

This will install and register as normal. If you have no other runtimes installed, then `python` will launch this one. Otherwise, you will need to use `py -V:3.14t . . .` or, if you have added the global aliases directory to your `PATH` environment variable, the `python3.14t.exe` commands.

4.1.13 Troubleshooting

If your Python install manager does not seem to be working correctly, please work through these tests and fixes to see if it helps. If not, please report an issue at [our bug tracker](#), including any relevant log files (written to your `%TEMP%` directory by default).

Troubleshooting

Symptom	Things to try
<code>python</code> gives me a «command not found» error or opens the Store app when I type it in my terminal.	Did you <i>install the Python install manager</i> ? Click Start, open «Manage app execution aliases», and check that the aliases for «Python (default)» are enabled. If they already are, try disabling and re-enabling to refresh the command. The «Python (default windowed)» and «Python install manager» commands may also need refreshing.
<code>py</code> gives me a «command not found» error when I type it in my terminal.	Check that the <code>py</code> and <code>pymanager</code> commands work. Did you <i>install the Python install manager</i> ? Click Start, open «Manage app execution aliases», and check that the aliases for «Python (default)» are enabled. If they already are, try disabling and re-enabling to refresh the command. The «Python (default windowed)» and «Python install manager» commands may also need refreshing.
<code>py</code> gives me a «can't open file» error when I type commands in my terminal.	This usually means you have the legacy launcher installed and it has priority over the Python install manager. To remove, click Start, open «Installed apps», search for «Python launcher» and uninstall it.
<code>python</code> doesn't launch the same runtime as <code>py</code>	Click Start, open «Installed apps», look for any existing Python runtimes, and either remove them or Modify and disable the <code>PATH</code> options. Click Start, open «Manage app execution aliases», and check that your <code>python.exe</code> alias is set to «Python (default)»
<code>python</code> and <code>py</code> don't launch the runtime I expect	Check your <code>PYTHON_MANAGER_DEFAULT</code> environment variable or <code>default_tag</code> configuration. The <code>py list</code> command will show your default based on these settings. Installs that are managed by the Python install manager will be chosen ahead of unmanaged installs. Use <code>py install</code> to install the runtime you expect, or configure your default tag. Prerelease and experimental installs that are not managed by the Python install manager may be chosen ahead of stable releases. Configure your default tag or uninstall the prerelease runtime and reinstall using <code>py install</code> .
<code>pythonw</code> or <code>pyw</code> don't launch the same runtime as <code>python</code> or <code>py</code>	Click Start, open «Manage app execution aliases», and check that your <code>pythonw.exe</code> and <code>pyw.exe</code> aliases are consistent with your others.
<code>pip</code> gives me a «command not found» error when I type it in my terminal.	Have you activated a virtual environment? Run the <code>.venv\Scripts\activate</code> script in your terminal to activate. The package may be available but missing the generated executable. We recommend using the <code>python -m pip</code> command instead, or alternatively the <code>python -m pip install --force pip</code> command will recreate the executables and show you the path to add to <code>PATH</code> . These scripts are separated for each runtime, and so you may need to add multiple paths.

4.2 El paquete incrustable

Added in version 3.5.

La distribución incrustable consiste en un archivo ZIP que contiene un mínimo entorno de Python. Está destinado a

ser usado como parte de otra aplicación, en lugar de ser accedido directamente por los usuarios finales.

To install an embedded distribution, we recommend using `py install` with the `--target` option:

```
$> py install 3.14-embed --target=runtime
```

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`, `python313.dll`, `python.exe` and `pythonw.exe` are all provided. Tcl/tk (including all dependents, such as Idle), pip and the Python documentation are not included.

A default `._pth` file is included, which further restricts the default search paths (as described below in [Encontrar módulos](#)). This file is intended for embedders to modify as necessary.

Los paquetes de terceros deben ser instalados por el instalador de la aplicación junto a la distribución incrustada. El uso de pip para administrar dependencias como en una instalación de Python regular no es soportado por esta distribución, aunque con cierto cuidado es posible incluir y usar pip para automatizar las actualizaciones. En general, los paquetes de terceros deben ser tratados como parte de la aplicación («vendoring») para que el desarrollador pueda asegurar la compatibilidad con las nuevas versiones antes de proporcionar actualizaciones a los usuarios.

Los dos casos de uso recomendados para esta distribución se describen a continuación.

4.2.1 Aplicación Python

Una aplicación escrita en Python no necesariamente requiere que los usuarios sean conscientes de ese hecho. La distribución incrustada puede ser usada en este caso para incluir una versión privada de Python en un paquete de instalación. Dependiendo de lo transparente que deba ser (o por el contrario, de lo profesional que deba parecer), hay dos opciones.

El uso de un ejecutable especializado como lanzador requiere algo de código, pero proporciona la experiencia más transparente para los usuarios. Con un lanzador personalizado, no hay indicadores obvios de que el programa se ejecuta en Python: los íconos pueden ser personalizados, se puede especificar información de la compañía y de la versión, y las asociaciones de archivos se comportan correctamente. En la mayoría de los casos, un lanzador personalizado debería simplemente poder invocar `Py_Main` utilizando una línea de comandos codificada.

El enfoque más simple es proporcionar un archivo por lotes o un acceso directo generado que directamente invoque `python.exe` o `pythonw.exe` con los argumentos de línea de comandos requeridos. En este caso, la aplicación aparecerá como Python y no con su nombre real, y los usuarios podrían tener problemas para distinguirla de otros procesos Python en ejecución o asociaciones de archivos.

Con este último enfoque, los paquetes deben instalarse como directorios junto al ejecutable de Python para asegurar su disponibilidad en la ruta. Con el lanzador especializado, los paquetes pueden encontrarse en otras ubicaciones ya que hay oportunidad de especificar la ruta de búsqueda antes de iniciar la aplicación.

4.2.2 Incrustar Python

Las aplicaciones escritas en código nativo frecuentemente requieren algún tipo de lenguaje de scripting, y la distribución de Python incrustada puede ser utilizada con ese propósito. En general, la mayoría de la aplicación utiliza código nativo, y alguna parte invocará `python.exe` o usará `python3.dll` directamente. Para cualquiera de estos casos, la extracción de la distribución incrustable a un subdirectorio de la instalación de la aplicación es suficiente para proporcionar un intérprete de Python invocable.

Al igual que con el uso de la aplicación, los paquetes pueden ser instalados en cualquier ubicación, ya que existe la posibilidad de especificar rutas de búsqueda antes de inicializar el intérprete. Más allá de esto, no existen diferencias fundamentales entre el uso de la distribución incrustada y una instalación normal.

4.3 El paquete de nuget.org

Added in version 3.5.2.

El paquete de nuget.org es un entorno Python de tamaño reducido destinado a usarse en sistemas de integración continua y compilación que no posean una instalación de Python a nivel de sistema. Si bien nuget es «el administrador

de paquetes para .NET», también funciona perfectamente para paquetes que contienen herramientas de tiempo de compilación.

Visite nuget.org para obtener la información más actualizada sobre cómo usar nuget. Lo que sigue es un resumen que es suficiente para desarrolladores Python.

La herramienta de línea de comandos `nuget.exe` puede ser descargada directamente desde <https://aka.ms/nugetclidl>, por ejemplo usando `curl` o `PowerShell`. Con esta herramienta, la última versión de Python para máquinas de 64 o 32 bit se instala con:

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

Para seleccionar una versión en particular, agregue un `-Version 3.x.y`. El directorio de salida se puede cambiar de `.` y el paquete se instalará en un subdirectorio. De forma predeterminada, el subdirectorio tiene el mismo nombre que el paquete y, sin la opción `-ExcludeVersion`, este nombre incluirá la versión específica instalada. Dentro del subdirectorio hay un directorio `tools` que contiene la instalación de Python:

```
# 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
```

En general, los paquetes nuget no son actualizables, y versiones más nuevas deben ser instaladas en paralelo y referenciadas usando la ruta completa. Otra opción es borrar el directorio del paquete de forma manual e instalarlo de nuevo. Muchos sistemas de CI harán esto automáticamente si no mantienen archivos entre compilaciones.

Junto al directorio `tools` está el directorio `build\native`. Este contiene un archivo de propiedades `MSBuild Python.props` que puede ser usado en un proyecto C++ para referenciar la instalación de Python. Al incluir las configuraciones, automáticamente se usarán los encabezados y se importarán las bibliotecas en la compilación.

The package information pages on nuget.org are www.nuget.org/packages/python for the 64-bit version, www.nuget.org/packages/pythonx86 for the 32-bit version, and www.nuget.org/packages/pythonarm64 for the ARM64 version

4.3.1 Free-threaded packages

Added in version 3.13: (Experimental)

Nota

Everything described in this section is considered experimental, and should be expected to change in future releases.

Packages containing free-threaded binaries are named `python-freethreaded` for the 64-bit version, `pythonx86-freethreaded` for the 32-bit version, and `pythonarm64-freethreaded` for the ARM64 version. These packages contain both the `python3.13t.exe` and `python.exe` entry points, both of which run free threaded.

4.4 Distribuciones alternativas

Además de la distribución estándar de CPython, hay paquetes modificados que incluyen funcionalidad adicional. La siguiente es una lista de versiones populares y sus características clave:

ActivePython

Instalador compatible con múltiples plataformas, documentación, PyWin32

Anaconda

Módulos científicos populares (como `numpy`, `scipy` y `pandas`) y el gestor de paquetes `conda`.

Enthought Deployment Manager

«El administrador de paquetes y entorno de Python de próxima generación».

Anteriormente, Enthought proporcionaba Canopy, pero [llegó al final de su vida en 2016](#).

WinPython

Distribución específica para Windows con paquetes científicos precompilados y herramientas para construir paquetes.

Tenga en cuenta que estos paquetes pueden no incluir la última versión de Python u otras bibliotecas, y no son mantenidos ni respaldados por el equipo central de Python.

4.5 Supported Windows versions

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 3.14 supports Windows 10 and newer. If you require Windows 7 support, please install Python 3.8. If you require Windows 8.1 support, please install Python 3.12.

4.6 Quitar el límite de MAX_PATH

Windows históricamente ha limitado la longitud de las rutas a 260 caracteres. Esto significaba que rutas de mayor longitud no resolverían y se producirían errores.

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

Esto permite que la función `open()`, el módulo `os` y la mayoría de las demás funciones de ruta acepten y retornen rutas de más de 260 caracteres.

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

4.7 Modo UTF-8

Added in version 3.7.

Windows todavía usa codificaciones heredadas para la codificación del sistema (la página de códigos ANSI). Python lo usa para la codificación predeterminada de archivos de texto (por ejemplo, `locale.getencoding()`).

Esto puede causar problemas porque UTF-8 es ampliamente utilizado en internet y en la mayoría de los sistemas Unix, incluido WSL (subsistema de Windows para Linux).

Se puede utilizar el modo UTF-8 para cambiar la codificación predeterminada a UTF-8. El modo UTF-8 se puede activar mediante la opción de línea de comandos `-X utf-8`, o con la variable de entorno `PYTHONUTF8=1`. Consulte [PYTHONUTF8](#) para activar el modo UTF-8, y [Python Install Manager](#) para saber cómo modificar las variables de entorno.

Cuando el modo UTF-8 de Python es activado, usted puede seguir usando la codificación del sistema (La página de código ANSI) a través del códec «mbcs».

Tenga en cuenta que agregar `PYTHONUTF8=1` a las variables de entorno predeterminadas afectará a todas las aplicaciones de Python 3.7+ en el sistema. Si utiliza alguna aplicación de Python 3.7+ que depende de la codificación heredada del sistema, se recomienda que se configure la variable de entorno solo temporalmente o se use la opción de línea de comandos `-X utf8`.

Nota

Aún con el modo UTF-8 desactivado, Python utiliza UTF-8 de forma predeterminada en Windows para:

- E/S de consola, incluida la E/S estándar (consultar [PEP 528](#) para más detalles).
- La *codificación del sistema* (vea [PEP 529](#) para más detalles).

4.8 Encontrar módulos

Estas notas complementan la descripción en `sys-path-init` con notas detalladas de Windows.

Cuando no se encuentre ningún archivo `._pth`, así es como `sys.path` es completado en Windows:

- Se agrega una entrada vacía al comienzo, que corresponde al directorio actual.
- Si existe la variable de entorno `PYTHONPATH`, de acuerdo a lo descrito en *Variables de entorno*, sus entradas se agregan a continuación. Tenga en cuenta que en Windows, las rutas en esta variable deben estar separadas por punto y coma (;), para distinguirlas de los dos puntos utilizados en los identificadores de disco (C:\, etc.).
- Se pueden agregar al registro «rutas de aplicación» adicionales como subclaves de `\SOFTWARE\Python\PythonCore{version}\PythonPath` bajo los subárboles `HKEY_CURRENT_USER` y `HKEY_LOCAL_MACHINE`. Las subclaves que contienen un valor por defecto compuesto por cadenas de ruta separadas por punto y coma causan que cada una de esas rutas sea agregada a `sys.path`. (Tenga en cuenta que todos los instaladores conocidos solo utilizan HKLM, por lo que HKCU comúnmente se encuentra vacío.)
- Si se configura la variable de entorno `PYTHONHOME`, es asumida como el «Python Home» (el directorio de origen de Python). De lo contrario, la ruta del ejecutable principal de Python es utilizada para ubicar un «archivo de referencia» (ya sea `Lib\os.py` o `pythonXY.zip`) para deducir el «Python Home». Si el directorio de origen de Python es encontrado, los subdirectorios relevantes que se agregan a `sys.path` (`Lib`, `plat-win`, etc.) se basan en ese directorio. Por el contrario, la ruta principal de Python se construye a partir del `PythonPath` guardado en el registro.
- Si el Python Home no puede ser ubicado, `PYTHONPATH` no está especificado en el entorno y no se encuentra ninguna entrada en el registro, se usa una ruta predeterminada con entradas relativas (por ej. `.\Lib`; `.\plat-win`, etc.).

Si se encuentra el archivo `pyvenv.cfg` junto al ejecutable principal o en el directorio un nivel arriba del ejecutable, se aplica la siguiente variación:

- Si `home` es una ruta absoluta y `PYTHONHOME` no está configurada, se usa esta ruta en lugar de la ruta al ejecutable principal para deducir la ubicación del directorio de origen.

El resultado final de todo esto es:

- Cuando se ejecuta `python.exe`, o cualquier otro `.exe` en el directorio principal de Python (tanto la versión instalada como directamente desde el directorio PCbuild), se deduce la ruta principal, y se ignoran las rutas principales en el registro. Siempre se leen otras «rutas de aplicación» del registro.
- Cuando se aloja Python en otro `.exe` (distinto directorio, incrustado mediante COM, etc.), el «Python Home» no se deduce, y se utiliza la ruta principal del registro. Siempre se leen otras «rutas de aplicación» del registro.
- Si Python no puede encontrar su directorio de origen y no hay valores en el registro (un `.exe` congelado, una muy rara configuración de instalación) se obtiene una ruta relativa predeterminada.

Para aquellos que quieran incluir Python en su aplicación o distribución, los siguientes consejos evitarán conflictos con otras instalaciones:

- Incluya un archivo `._pth` junto al ejecutable, que contenga los directorios a incluir. Esto hará que se ignoren las rutas enumeradas en el registro y en las variables de entorno, y que también se ignore `site` a menos que se especifique `import site`.
- If you are loading `python3.dll` or `python37.dll` in your own executable, explicitly set `PyConfig.module_search_paths` before `Py_InitializeFromConfig()`.
- Limpie y/o sobrescriba `PYTHONPATH` y configure `PYTHONHOME` antes de iniciar `python.exe` desde su aplicación.

- Si no se pueden utilizar las sugerencias previas (por ejemplo, en una distribución que permite a los usuarios ejecutar `python.exe` directamente), hay que asegurarse de que el archivo de referencia (`Lib\os.py`) exista en el directorio de instalación. (Tener en cuenta que este no será detectado dentro de un archivo ZIP, pero si se detectará un ZIP correctamente nombrado.)

Esto asegura que los archivos de una instalación del sistema no tendrán precedencia por sobre la copia de la biblioteca estándar incluida en su aplicación. De otra manera, los usuarios podrían experimentar problemas al utilizar su aplicación. Tenga en cuenta que la primera sugerencia es la mejor, ya que las otras aún pueden ser afectadas por rutas no estándar en el registro y en el `site-packages` del usuario.

Distinto en la versión 3.6: Add `._pth` file support and removes `applocal` option from `pyenvv.cfg`.

Distinto en la versión 3.6: Add `pythonXX.zip` as a potential landmark when directly adjacent to the executable.

Obsoleto desde la versión 3.6: Los módulos especificados en el registro bajo `Modules` (no `PythonPath`) pueden ser importados por `importlib.machinery.WindowsRegistryFinder`. Este buscador está habilitado en Windows en la versión 3.6.0 y anteriores, pero es posible que deba agregarse explícitamente a `sys.meta_path` en el futuro.

4.9 Módulos adicionales

Aunque Python pretende ser portátil entre todas las plataformas, hay características que son exclusivas de Windows. Existen un par de módulos, de la biblioteca estándar y externos, y fragmentos de código para utilizar estas funciones.

Los módulos estándar específicos para Windows se encuentran documentados en `mswin-specific-services`.

4.9.1 PyWin32

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

- [Component Object Model \(COM\)](#)
- Invocación de la API Win32
- Registro
- Registro de eventos
- [Microsoft Foundation Classes \(MFC\)](#) user interfaces

`PythonWin` es una aplicación MFC de muestra distribuida con `PyWin32`. Es un IDE incrustable con depurador incorporado.

Ver también

[Win32 How Do I...?](#)

por Tim Golden

[Python and COM](#)

por David y Paul Boddie

4.9.2 cx_Freeze

`cx_Freeze` envuelve scripts de Python en programas ejecutables de Windows (archivos `*.exe`). Cuando hayas hecho esto, puedes distribuir tu aplicación sin requerir que tus usuarios instalen Python.

4.10 Compilar Python en Windows

Si desea compilar CPython usted mismo, lo primero que debe hacer es obtener el [source](#). Puede descargar la fuente de la última versión o simplemente obtener un [checkout](#) nuevo.

El árbol fuente contiene una solución de compilación y archivos de proyecto para Microsoft Visual Studio, que es el compilador que se usa para compilar las versiones oficiales de Python. Estos archivos están en el directorio `PCbuild`.

Consulte `PCbuild/readme.txt` para obtener información general acerca del proceso de compilación.

Para módulos de extensión, consulte `building-on-windows`.

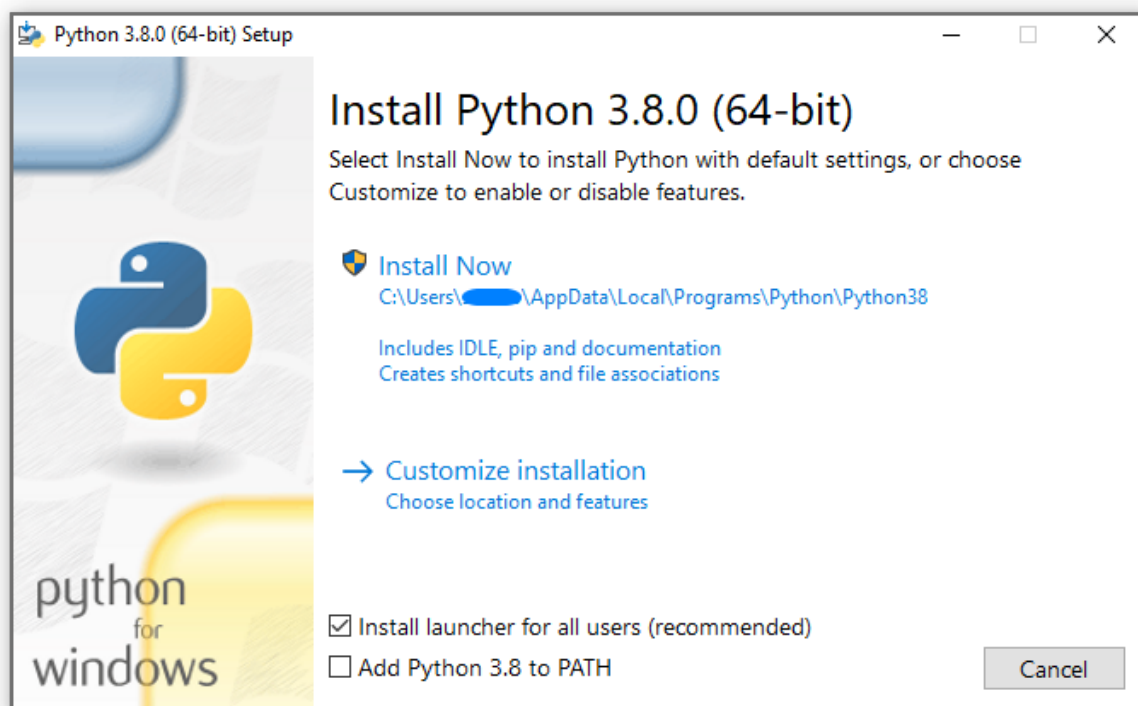
4.11 The full installer (deprecated)

Obsoleto desde la versión 3.14: This installer is deprecated since 3.14 and will not be produced for Python 3.16 or later. See *Python Install Manager* for the modern installer.

4.11.1 Pasos para la instalación

Cuatro instaladores de Python 3.14 están disponibles para descargar - dos por cada una de las versiones de 32-bit y 64-bit del intérprete. El *instalador web* es una pequeña descarga inicial que automáticamente descargará los componentes requeridos cuando sea necesario. El *instalador fuera de línea* incluye los componentes necesarios para una instalación por defecto y solo requiere de una conexión a internet para características opcionales. Consultar *Instalación sin descargas* para conocer otras formas de evitar descargas durante la instalación.

Luego de iniciar el instalador, se puede seleccionar una de estas dos opciones:



Si se selecciona «Install Now»:

- No necesitarás ser administrador (a menos que se requiera una actualización de sistema para C Runtime Library o se necesite instalar el *Python Install Manager* para todos los usuarios)
- Python será instalado en su directorio de usuario
- El *Python Install Manager* será instalado de acuerdo con la opción en la parte inferior de la primera página
- La biblioteca estándar, conjunto de pruebas, lanzador y pip serán instalados
- Si se selecciona, el directorio de instalación se agregará a su `PATH`
- Los accesos directos solo serán visibles para al usuario actual

Si selecciona «Customize installation» podrá elegir qué funciones instalar, el destino de la instalación y otras opciones o acciones posinstalación. Para instalar símbolos de depuración o binarios, necesitará usar esta opción.

Para realizar una instalación para todos los usuarios, deberá seleccionar «Customize installation». En este caso:

- Es posible que deba proporcionar credenciales administrativas o aprobación
- Python será instalado en el directorio Program Files
- El *Python Install Manager* será instalado en el directorio Windows
- Se pueden seleccionar características opcionales durante la instalación
- La biblioteca estándar puede ser precompilada a bytecode
- Si se selecciona, el directorio de instalación será agregado al `PATH` del sistema
- Los accesos directos están disponibles para todos los usuarios

4.11.2 Quitar el límite de `MAX_PATH`

Windows históricamente ha limitado la longitud de las rutas a 260 caracteres. Esto significaba que rutas de mayor longitud no resolverían y se producirían errores.

En las últimas versiones de Windows, esta limitación se puede ampliar a aproximadamente 32.000 caracteres. Su administrador deberá activar la política de grupo «Habilitar rutas largas de Win32» o establecer `LongPathsEnabled` en 1 en la clave de registro `HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\FileSystem`.

Esto permite que la función `open()`, el módulo `os` y la mayoría de las demás funciones de ruta acepten y retornen rutas de más de 260 caracteres.

Luego de cambiar la opción anterior, no es necesaria ninguna otra configuración.

Distinto en la versión 3.6: Se habilitó el soporte para rutas largas en Python.

4.11.3 Instalación sin interfaz de usuario

Todas las opciones disponibles desde la interfaz de usuario del instalador también pueden especificarse desde la línea de comandos, lo cual permite que instaladores mediante scripts repliquen una instalación en muchas máquinas sin la interacción del usuario. Estas opciones también pueden ser configuradas sin anular la interfaz de usuario con el fin de cambiar alguno de los valores predeterminados.

Las siguientes opciones (encontradas al ejecutar el instalador con `/?`) pueden ser pasadas al instalador

Nombre	Descripción
<code>/passive</code>	para mostrar el progreso sin requerir interacción del usuario
<code>/quiet</code>	para instalar/desinstalar sin mostrar ninguna interfaz
<code>/simple</code>	para prevenir la personalización del usuario
<code>/uninstall</code>	para eliminar Python (sin confirmación)
<code>/layout [directorio]</code>	para pre-descargar todos los componentes
<code>/log [nombre de archivo]</code>	para especificar la ubicación de los archivos de registro

Todas las otras opciones se especifican con la forma `nombre=valor`, siendo el valor usualmente 0 para deshabilitar una funcionalidad, 1 para habilitar una funcionalidad, o una ruta. La lista completa de opciones disponibles se muestra a continuación.

Nombre	Descripción	Predeterminado
InstallAllUsers	Realizar una instalación en todo el sistema.	0
TargetDir	El directorio de instalación	Seleccionado de acuerdo a InstallAllUsers
DefaultAllUsersTargetDir	El directorio predeterminado de instalación cuando se instala para todos los usuarios	%ProgramFiles%\Python X.Y o %ProgramFiles(x86)%\Python X.Y
DefaultJustForMeTargetDir	El directorio predeterminado de instalación para instalaciones del usuario actual solamente	%LocalAppData%\Programs\Python\PythonXY or %LocalAppData%\Programs\Python\PythonXY-32 or %LocalAppData%\Programs\Python\Python\PythonXY-64
DefaultCustomTargetDir	El valor predeterminado de directorio de instalación personalizado que se muestra en la interfaz de usuario	(vacío)
AssociateFiles	Crear asociaciones de archivos si el lanzador también es instalado.	1
CompileAll	Compilar todos los archivos .py a .pyc.	0
PrependPath	Anteponga los directorios de instalación y scripts a PATH y agregue .PY a PATHEXT	0
AppendPath	Agregue directorios de instalación y scripts a PATH y agregue .PY a PATHEXT	0
Shortcuts	Crear accesos directos para el intérprete, documentación e IDLE si está instalado.	1
Include_doc	Instalar el manual de Python	1
Include_debug	Instalar los binarios de depuración	0
Include_dev	Instale encabezados y bibliotecas de desarrollador. Omitir esto puede conducir a una instalación inutilizable.	1
Include_exe	Instale python.exe y archivos relacionados. Omitir esto puede conducir a una instalación inutilizable.	1
Include_launcher	Instalar <i>Python Install Manager</i> .	1
InstallAllUsers	Instala el lanzador para todos los usuarios. También requiere que Include_launcher se establezca en 1	1
Include_lib	Instale la biblioteca estándar y los módulos de extensión. Omitir esto puede conducir a una instalación inutilizable.	1
Include_pip	Instalar los paquetes pip y setuptools	1
Include_symbol	Instalar los símbolos de depuración (*.pdb)	0
Include_tcltk	Instalar IDLE y soporte para Tcl/Tk	1
Include_test	Instalar el conjunto de pruebas de la biblioteca estándar	1
Include_tools	Instalar scripts de utilidades	1
LauncherOnly	Instalar solo el lanzador. Esto anulará la mayoría de las otras opciones.	0
SimpleInstall	Deshabilitar muchas de las partes de la interfaz de usuario	0

Por ejemplo, para realizar de forma silenciosa una instalación predeterminada de Python para todo el sistema, se puede usar el siguiente comando (desde un símbolo del sistema con privilegios elevados):

```
python-3.9.0.exe /quiet InstallAllUsers=1 PrependPath=1 Include_test=0
```

Para permitir que los usuarios instalen fácilmente una copia personal de Python sin el conjunto de pruebas, se puede proporcionar un acceso directo con el siguiente comando. Esto mostrará una página inicial simplificada y no permitirá la personalización:

```
python-3.9.0.exe InstallAllUsers=0 Include_launcher=0 Include_test=0
SimpleInstall=1 SimpleInstallDescription="Just for me, no test suite."
```

(Tener en cuenta que al omitir el lanzador también se omiten las asociaciones de archivos y solo es recomendable hacerlo para instalaciones por usuario cuando ya hay una instalación en todo el sistema que incluye el lanzador.)

Las opciones enumeradas anteriormente también se pueden proporcionar en un archivo de nombre `unattend.xml` junto al ejecutable. Este archivo especifica una lista de opciones y valores. Cuando un valor se proporciona como un atributo, se convertirá a número si es posible. Los valores proporcionados como elementos de texto siempre se dejan como cadenas. Este archivo de ejemplo configura las mismas opciones que el ejemplo anterior:

```
<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>
```

4.11.4 Instalación sin descargas

Como algunas características de Python no se incluyen con la descarga inicial del instalador, la selección de estas características podría requerir de una conexión a internet. Para evitar esta necesidad, todos los posibles componentes pueden ser descargados a pedido para crear una estructura que no necesitará una conexión a internet, independientemente de las características que se seleccionen. Tener en cuenta que esta descarga puede ser más grande de lo necesario, pero si se va a realizar un gran número de instalaciones es muy útil tener una copia en la caché local.

Ejecute el siguiente comando desde el símbolo del sistema para descargar todos los archivos necesarios posibles. Recuerde sustituir `python-3.9.0.exe` por el nombre real de su instalador y crear diseños en sus propios directorios para evitar colisiones entre archivos con el mismo nombre.

```
python-3.9.0.exe /layout [optional target directory]
```

También se puede especificar la opción `/quiet` para no mostrar el progreso.

4.11.5 Modificar una instalación

Una vez que Python ha sido instalado, se puede agregar o quitar funciones a través de la herramienta Programas y características que es parte de Windows. Seleccionar la entrada Python y elegir «Desinstalar/Cambiar» para abrir el instalador en modo mantenimiento.

«Cambiar» permite agregar o eliminar características modificando las casillas de verificación - aquellas casillas que no se cambien no agregarán ni quitarán nada. Algunas opciones no pueden cambiarse de esta forma, como el directorio de instalación; para modificarlas es necesario eliminar y reinstalar Python completamente.

«Reparar» verificará todos los archivos que deben instalarse con la configuración actual y reemplazará los que se hayan eliminado o modificado.

«Desinstalar» eliminará Python completamente, a excepción del *Python Install Manager*, el cual posee su propia entrada en Programas y características.

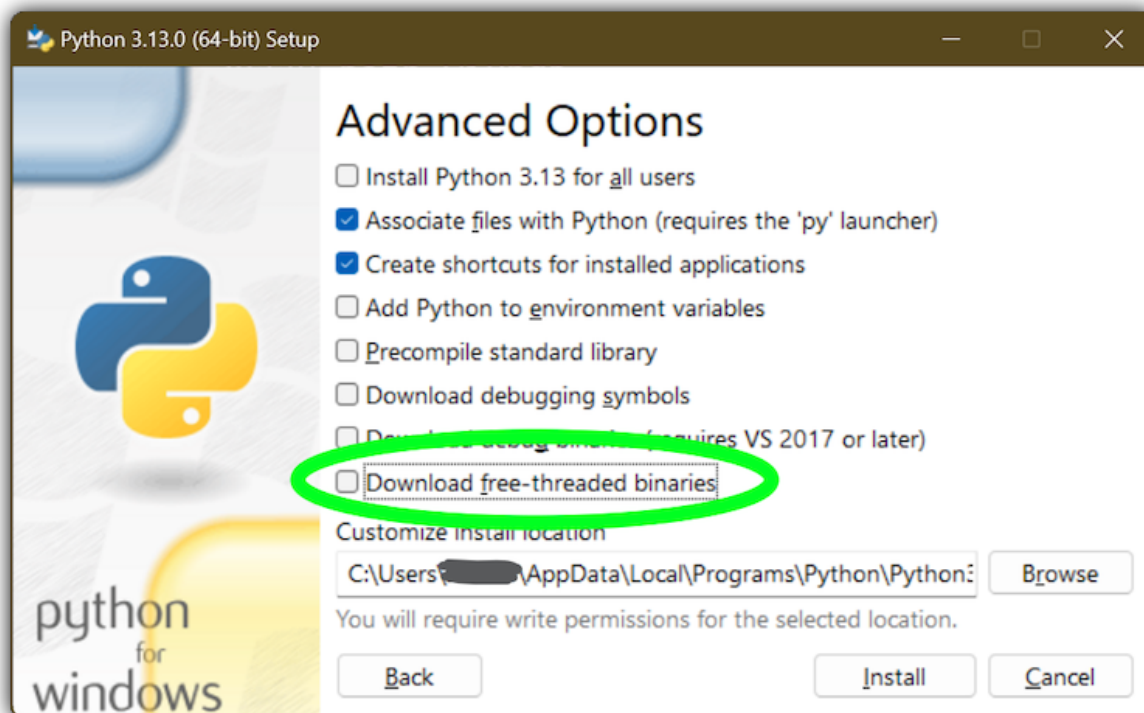
4.11.6 Installing Free-threaded Binaries

Added in version 3.13: (Experimental)

i Nota

Everything described in this section is considered experimental, and should be expected to change in future releases.

To install pre-built binaries with free-threading enabled (see [PEP 703](#)), you should select «Customize installation». The second page of options includes the «Download free-threaded binaries» checkbox.



Selecting this option will download and install additional binaries to the same location as the main Python install. The main executable is called `python3.13t.exe`, and other binaries either receive a `t` suffix or a full ABI suffix. Python source files and bundled third-party dependencies are shared with the main install.

The free-threaded version is registered as a regular Python install with the tag `3.13t` (with a `-32` or `-arm64` suffix as normal for those platforms). This allows tools to discover it, and for the *Python Install Manager* to support `py.exe -3.13t`. Note that the launcher will interpret `py.exe -3` (or a `python3` shebang) as «the latest 3.x install», which will prefer the free-threaded binaries over the regular ones, while `py.exe -3.13` will not. If you use the short style of option, you may prefer to not install the free-threaded binaries at this time.

To specify the install option at the command line, use `Include_freethreaded=1`. See *Instalación sin descargas* for instructions on pre-emptively downloading the additional binaries for offline install. The options to include debug symbols and binaries also apply to the free-threaded builds.

Free-threaded binaries are also available [on nuget.org](https://nuget.org).

4.12 Python Launcher for Windows (Deprecated)

Obsoleto desde la versión 3.14: The launcher and this documentation have been superseded by the Python Install Manager described above. This is preserved temporarily for historical interest.

Added in version 3.3.

El lanzador de Python para Windows es una utilidad que ayuda en la ubicación y ejecución de diferentes versiones de Python. Este permite que los scripts (o la línea de comandos) indiquen preferencia por una versión específica de Python, y ubicará y ejecutará esa versión.

A diferencia de la variable `PATH`, el lanzador seleccionará correctamente la versión más apropiada de Python. Priorizará instalaciones del usuario por sobre instalaciones de todo el sistema, y ordenará por versión del lenguaje en lugar de utilizar la más recientemente instalada.

El lanzador se especificó originalmente en [PEP 397](#).

4.12.1 Comenzar

Desde la línea de comandos

Distinto en la versión 3.6.

Las instalaciones de todo el sistema de Python 3.3 y versiones posteriores colocarán el lanzador en su `PATH`. El lanzador es compatible con todas las versiones disponibles de Python, por lo que no importa qué versión esté instalada. Para comprobar que el lanzador está disponible, ejecute el siguiente comando en el símbolo del sistema:

```
py
```

Debería suceder que se inicia la última versión de Python instalada - se puede cerrar normalmente, y todo argumento adicional especificado por línea de comandos será enviado directamente a Python.

Si tiene varias versiones de Python instaladas (por ejemplo, 3.7 y 3.14), habrá notado que Python 3.14 se inició: para iniciar Python 3.7, pruebe el comando:

```
py -3.7
```

Si quieres la última versión de Python 2 que tienes instalada, prueba el comando:

```
py -2
```

Si ve el siguiente error, no tiene instalado el lanzador:

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

El comando:

```
py --list
```

muestra la(s) versión(es) actualmente instalada(s) de Python.

El argumento `-x.y` es la forma corta del argumento `-V:Compañía/Etiqueta`, que permite seleccionar un entorno de ejecución Python específico, incluidos aquellos que pueden haber provenido de lugares diferentes a python.org. Cualquier entorno registrado siguiendo el [PEP 514](#) será detectable. El comando `--list` muestra todos los entornos disponibles usando el formato `-V:`.

Al usar el argumento `-V:`, especificar la Compañía limitará la selección a entornos de ese proveedor, mientras que especificar solo la Etiqueta seleccionará de todos los proveedores. Tenga en cuenta que omitir la barra implica una etiqueta:

```
# Select any '3.*' tagged runtime
py -V:3

# Select any 'PythonCore' released runtime
py -V:PythonCore/

# Select PythonCore's latest Python 3 runtime
py -V:PythonCore/3
```

La forma corta del argumento (`-3`) solo selecciona de las versiones principales de Python y no de otras distribuciones. Sin embargo, la forma más larga (`-v:3`) seleccionará de cualquiera.

The Company is matched on the full string, case-insensitive. The Tag is matched on either the full string, or a prefix, provided the next character is a dot or a hyphen. This allows `-v:3.1` to match `3.1-32`, but not `3.10`. Tags are sorted using numerical ordering (`3.10` is newer than `3.1`), but are compared using text (`-v:3.01` does not match `3.1`).

Entornos virtuales

Added in version 3.5.

Si el lanzador es ejecutado sin explícita especificación de la versión de Python, y un entorno virtual se encuentra activo (creado con el módulo `venv` de la biblioteca estándar o con la herramienta externa `virtualenv`), el lanzador ejecutará el intérprete del entorno virtual en lugar del global. Para ejecutar el intérprete global, desactive el entorno virtual o especifique explícitamente la versión global de Python.

Desde un script

Vamos a crear un script de Python para una prueba - cree un archivo llamado `hello.py` con el siguiente contenido

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

Desde el directorio en el que vive `hello.py`, ejecute el comando:

```
py hello.py
```

Debería notar que se imprime el número de versión de la última instalación de Python 2.x. Ahora pruebe cambiando la primera línea por:

```
#!/ python3
```

Volver a ejecutar el comando ahora debería imprimir la información más reciente de Python 3.x. Al igual que con los ejemplos de línea de comandos anteriores, puede especificar un calificador de versión más explícito. Suponiendo que tiene instalado Python 3.7, intente cambiar la primera línea a `#!/ python3.7` y debería encontrar la información de la versión 3.7 impresa.

Tenga en cuenta que a diferencia del uso interactivo, el comando «python» (a secas) utilizará la última versión de Python 2.x que esté instalada. Esto es así por compatibilidad con versiones anteriores y por compatibilidad con Unix, donde el comando `python` usualmente refiere a Python 2.

Desde asociaciones de archivos

El lanzador debería haber sido asociado con archivos de Python (por ej. archivos `.py`, `.pyw` y `.pyc`) cuando fue instalado. Esto significa que cuando se haga doble click sobre alguno de estos archivos desde el explorador de Windows se utilizará el lanzador, por lo que se pueden utilizar las mismas funciones descritas anteriormente para que el script especifique la versión que debería usarse.

El beneficio clave de esto es que un único lanzador puede soportar múltiples versiones de Python al mismo tiempo dependiendo del contenido de la primera línea.

4.12.2 Líneas shebang

Si la primera línea de un script comienza con `#!`, esta se denomina línea «shebang». Linux y otros sistemas operativos tipo Unix soportan de forma nativa este tipo de líneas y son comúnmente utilizadas en dichos sistemas para indicar cómo debería ser ejecutado un script. Este lanzador permite que la misma funcionalidad pueda ser utilizada con scripts de Python en Windows, y los ejemplos anteriores demuestran su uso.

Para permitir que las líneas shebang de scripts de Python sean trasladables entre Unix y Windows, este lanzador soporta varios comandos “virtuales” para especificar qué intérprete utilizar. Los comandos virtuales soportados son:

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

Por ejemplo, si la primera línea del script comienza con

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

The default Python or an active virtual environment 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/python3.7-32` will request usage of the 32-bit Python 3.7. If a virtual environment is active, the version will be ignored and the environment will be used.

Added in version 3.7: Desde la versión 3.7 del lanzador de Python es posible solicitar la versión de 64 bit con el sufijo «-64». Además es posible especificar una versión mayor y la arquitectura sin la versión menor (por ej. `/usr/bin/python3-64`).

Distinto en la versión 3.11: El sufijo «-64» está en desuso y ahora implica «cualquier arquitectura que comprobable no sea i386/32 bits». Para solicitar un entorno específico, use el nuevo argumento `-v:TAG` con la etiqueta completa.

Distinto en la versión 3.13: Virtual commands referencing `python` now prefer an active virtual environment rather than searching `PATH`. This handles cases where the shebang specifies `/usr/bin/env python3` but `python3.exe` is not present in the active environment.

La forma `/usr/bin/env` de la línea shebang tiene otra propiedad especial adicional. Antes de buscar intérpretes de Python instalados, esta forma buscará en el `PATH` de ejecución un ejecutable de Python que coincida con el nombre proporcionado como primer argumento. Esto corresponde al comportamiento del programa Unix `env`, que realiza una búsqueda en `PATH`. Si no se puede encontrar un ejecutable que coincida con el primer argumento después del comando `env` pero el argumento comienza con `python`, será manejado como se describe para los otros comandos virtuales. La variable de entorno `PYLAUNCHER_NO_SEARCH_PATH` puede ser establecida (a cualquier valor) para omitir esta búsqueda en `PATH`.

Las líneas shebang que no coincidan con ninguno de estos patrones se buscan en la sección `[commands]` del *archivo .INI* del lanzador. Esto se puede usar para manejar ciertos comandos de una manera que tenga sentido para tu sistema. El nombre del comando debe ser un único argumento (sin espacios en el ejecutable shebang), y el valor sustituido es la ruta completa al ejecutable (los argumentos adicionales especificados en el *.INI* se citarán como parte del nombre del archivo).

```
[commands]
/bin/xpython=C:\Program Files\XPython\python.exe
```

Cualquier comando que no se encuentre en el archivo *.INI* se trata como rutas ejecutables de **Windows** que son absolutas o relativas al directorio que contiene el archivo de script. Esto es una comodidad para los scripts solo para Windows, como aquellos generados por un instalador, ya que el comportamiento no es compatible con los shells al estilo Unix. Estas rutas pueden estar entre comillas y pueden incluir varios argumentos, tras los cuales se añadirá la ruta al script y cualquier argumento adicional.

4.12.3 Argumentos en líneas shebang

Las líneas shebang también pueden especificar opciones adicionales para que sean pasadas al intérprete de Python. Por ej. si se tiene esta línea shebang:

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

Entonces Python se iniciará con la opción `-v`

4.12.4 Personalización

Personalización con archivos INI

El lanzador buscará dos archivos `.ini` - `py.ini` en el directorio de datos de aplicación del usuario actual (`%LOCALAPPDATA%` o `$env:LocalAppData`) y `py.ini` en el mismo directorio que el lanzador. Los mismos archivos `.ini` son utilizados tanto para la versión “consola” del lanzador (es decir, `py.exe`) como para la versión “windows” (es decir `pyw.exe`).

La personalización especificada en el «directorio de aplicación» tendrá precedencia por sobre la que esté junto al ejecutable, por lo que un usuario, que podría no tener acceso de escritura al archivo `.ini` que está junto al lanzador, puede sobrescribir comandos en ese archivo `.ini` global.

Personalizar las versiones de Python predeterminadas

En algunos casos, un calificador de versión puede ser incluido en un comando para dictar qué versión de Python será utilizada por dicho comando. Un calificador de versión comienza con el número mayor de la versión y puede ser seguido opcionalmente por un punto (“.”) y el número menor de la versión. Además es posible especificar si se solicita una implementación de 32 o 64 bit agregando «-32» o «-64».

Por ejemplo, una línea shebang como `#!/python` no posee calificador de versión, mientras que `#!/python3` sí tiene un calificador de versión el cual especifica solo el número mayor de la versión.

Si no se encuentra un calificador de versión en el comando, la variable de entorno `PY_PYTHON` puede configurarse para especificar un calificador de versión predeterminado. Si esta no está configurada, por defecto es «3». La variable puede especificar cualquier valor que pueda ser pasado por línea de comandos, como «3», «3.7», «3.7-32» o «3.7-64». (Tener en cuenta que la opción «-64» solo está disponible con el lanzador incluido con Python 3.7 o versiones posteriores.)

Si no se encuentra ningún calificador de versión menor, la variable de entorno `PY_PYTHON{major}` (donde {major} es el actual calificador de versión mayor según lo determinado antes) puede ser configurada para especificar la versión completa. Si dicha opción no se encuentra, el lanzador enumerará las versiones de Python instaladas y utilizará la última versión menor encontrada para la versión mayor, la cual es probable, aunque no se garantiza, que sea la versión más recientemente instalada de esa familia.

En un Windows de 64 bit con ambas implementaciones de 32 y 64 bit de la misma versión (mayor.menor) de Python instaladas, la versión de 64 bit siempre tendrá precedencia. Esto se cumple para ambas implementaciones de 32 y 64 bit del lanzador - un lanzador de 32 bit priorizará ejecutar una instalación de Python de 64 bit de la versión especificada si está disponible. Esto es así para que el comportamiento del lanzador pueda ser predecible sabiendo solamente qué versiones están instaladas en la PC y sin importar el orden en el cual fueron instaladas (esto es, sin saber si una versión de Python de 32 o 64 bit y su correspondiente lanzador fue la última instalada). Como se especificó antes, el sufijo «-32» o «-64» puede ser utilizado en el especificador de versión para cambiar este comportamiento.

Ejemplos:

- Si no se configura ninguna opción relevante, los comandos `python` y `python2` utilizarán la última versión de Python 2.x instalada y el comando `python3` utilizará el último Python 3.x instalado.
- El comando `python3.7` no consultará ninguna opción ya que las versiones están completamente especificadas.
- Si `PY_PYTHON=3`, los comandos `python` y `python3` utilizarán ambos la última versión instalada de Python 3.
- Si es `PY_PYTHON=3.7-32`, el comando `python` usará la implementación de 32 bits de 3.7, mientras que el comando `python3` usará la última versión de Python instalada (`PY_PYTHON` no se consideró en absoluto porque se especificó una versión principal).
- Si `PY_PYTHON=3` y `PY_PYTHON3=3.7`, los comandos `python` y `python3` usarán específicamente 3.7

Además de las variables de entorno, las mismas configuraciones pueden realizarse desde el archivo `.INI` utilizado por el lanzador. La sección en el archivo INI se llama `[defaults]` y el nombre de cada clave será igual al de la variable de entorno pero sin el prefijo `PY_` (tenga en cuenta que los nombres de clave en el archivo INI son indiferentes a mayúsculas y minúsculas). El contenido de las variables de entorno sobrescribirá los valores especificados en el archivo INI.

Por ejemplo:

- La configuración de `PY_PYTHON=3.7` es equivalente al archivo INI que contiene:

```
[defaults]
python=3.7
```

- La configuración de `PY_PYTHON=3` y `PY_PYTHON3=3.7` es equivalente al archivo INI que contiene:

```
[defaults]
python=3
python3=3.7
```

4.12.5 Diagnóstico

Si se establece una variable de entorno `PYLAUNCHER_DEBUG` (en cualquier valor), el lanzador imprimirá información de diagnóstico en `stderr` (es decir, en la consola). Si bien esta información logra ser simultáneamente detallada *and* concisa, debería permitirle ver qué versiones de Python se ubicaron, por qué se eligió una versión en particular y la línea de comando exacta utilizada para ejecutar el Python de destino. Está destinado principalmente a pruebas y depuración.

4.12.6 Ejecución en seco

Si se establece una variable de entorno `PYLAUNCHER_DRYRUN` (en cualquier valor), el lanzador generará el comando que habría ejecutado, pero en realidad no iniciará Python. Esto puede ser útil para las herramientas que desean usar el lanzador para detectar y luego iniciar Python directamente. Tenga en cuenta que el comando escrito en la salida estándar siempre se codifica con UTF-8 y es posible que no se represente correctamente en la consola.

4.12.7 Instalación bajo demanda

Si se establece una variable de entorno `PYLAUNCHER_ALLOW_INSTALL` (en cualquier valor) y la versión de Python solicitada no está instalada pero está disponible en Microsoft Store, el lanzador intentará instalarla. Esto puede requerir la interacción del usuario para completarse y es posible que deba ejecutar el comando nuevamente.

Una variable `PYLAUNCHER_ALWAYS_INSTALL` adicional hace que el lanzador siempre intente instalar Python, incluso si se detecta. Esto está diseñado principalmente para pruebas (y debe usarse con `PYLAUNCHER_DRYRUN`).

4.12.8 Códigos de retorno

El lanzador de Python puede retornar los siguientes códigos de salida. Desafortunadamente, no hay forma de distinguirlos del código de salida de Python.

Los nombres de los códigos son como se usan en las fuentes y son solo para referencia. No hay forma de acceder a ellos o resolverlos aparte de leer esta página. Las entradas se enumeran en orden alfabético de nombres.

Nombre	Valor	Descripción
<code>RC_BAD_VENV_CFG</code>	107	Se encontró un <code>pyvenv.cfg</code> pero está corrupto.
<code>RC_CREATE_PROCESS</code>	101	No se pudo iniciar Python.
<code>RC_INSTALLING</code>	111	Se inició una instalación, pero será necesario volver a ejecutar el comando una vez que se complete.
<code>RC_INTERNAL_ERROR</code>	109	Error inesperado. Por favor, informe de un error.
<code>RC_NO_COMMANDLINE</code>	108	No se puede obtener la línea de comandos del sistema operativo.
<code>RC_NO_PYTHON</code>	103	No se puede encontrar la versión solicitada.
<code>RC_NO_VENV_CFG</code>	106	Se requería un <code>pyvenv.cfg</code> pero no se encontró.

Using Python on macOS

This document aims to give an overview of macOS-specific behavior you should know about to get started with Python on Mac computers. Python on a Mac running macOS is very similar to Python on other Unix-derived platforms, but there are some differences in installation and some features.

There are various ways to obtain and install Python for macOS. Pre-built versions of the most recent versions of Python are available from a number of distributors. Much of this document describes use of the Pythons provided by the CPython release team for download from the python.org website. See *Alternative Distributions* for some other options.

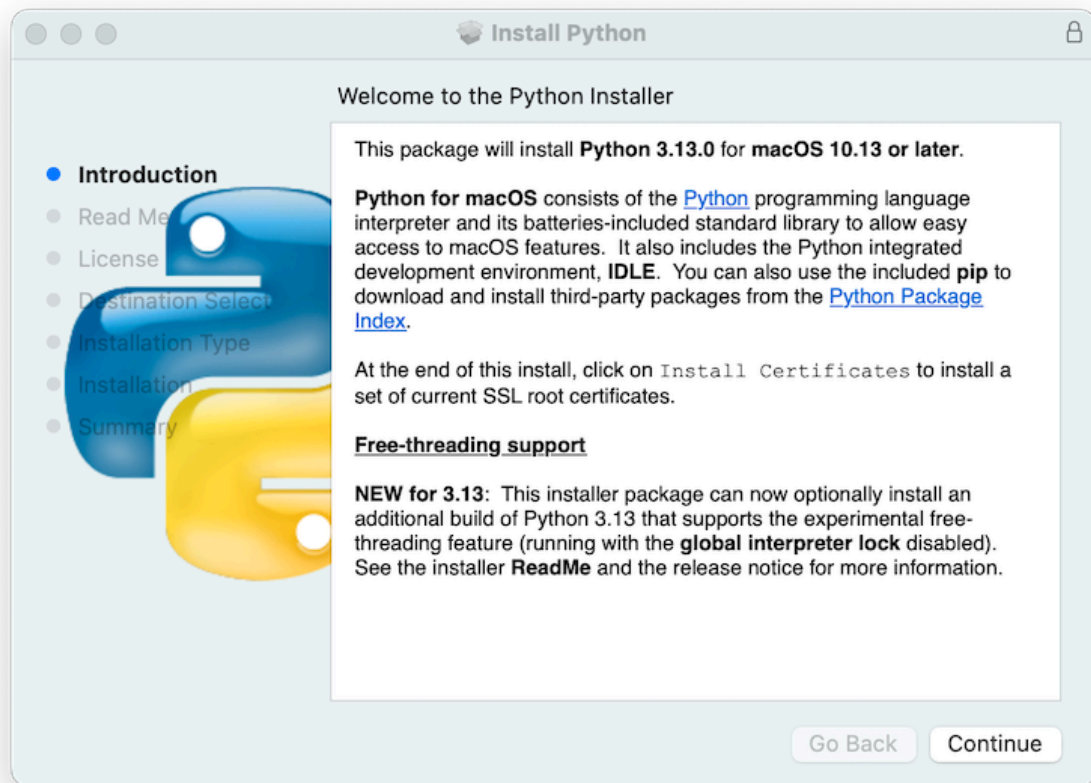
5.1 Using Python for macOS from `python.org`

5.1.1 Installation steps

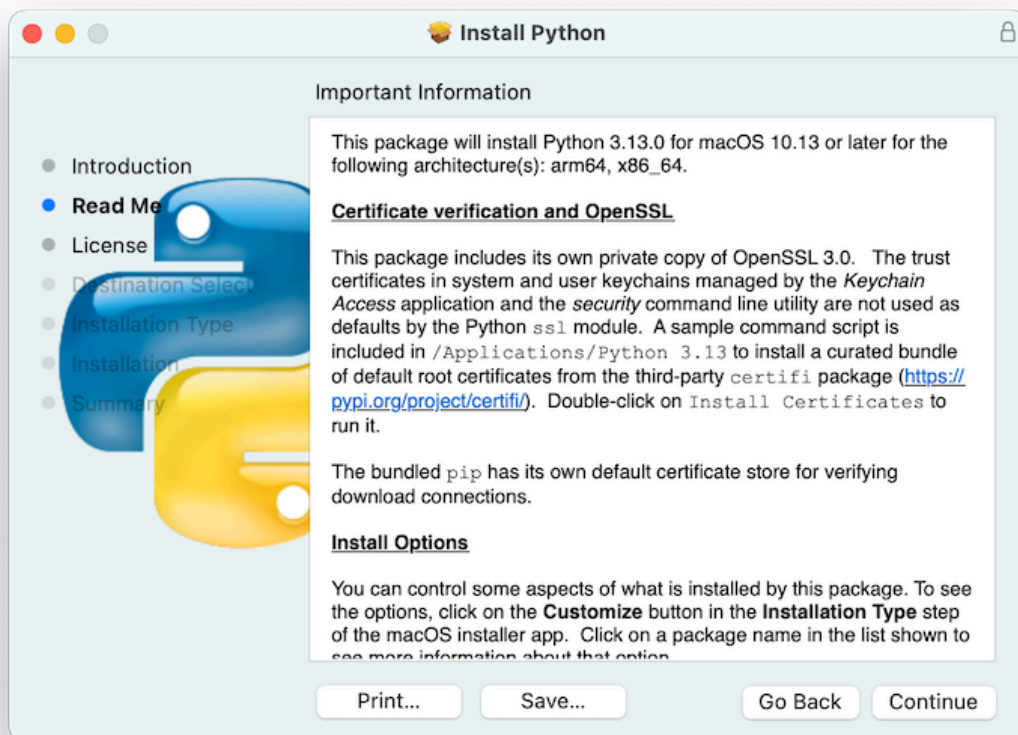
For [current Python versions](#) (other than those in `security` status), the release team produces a **Python for macOS** installer package for each new release. A list of available installers is available [here](#). We recommend using the most recent supported Python version where possible. Current installers provide a [universal2 binary](#) build of Python which runs natively on all Macs (Apple Silicon and Intel) that are supported by a wide range of macOS versions, currently typically from at least **macOS 10.15 Catalina** on.

The downloaded file is a standard macOS installer package file (`.pkg`). File integrity information (checksum, size, sigstore signature, etc) for each file is included on the release download page. Installer packages and their contents are signed and notarized with Python Software Foundation Apple Developer ID certificates to meet [macOS Gatekeeper requirements](#).

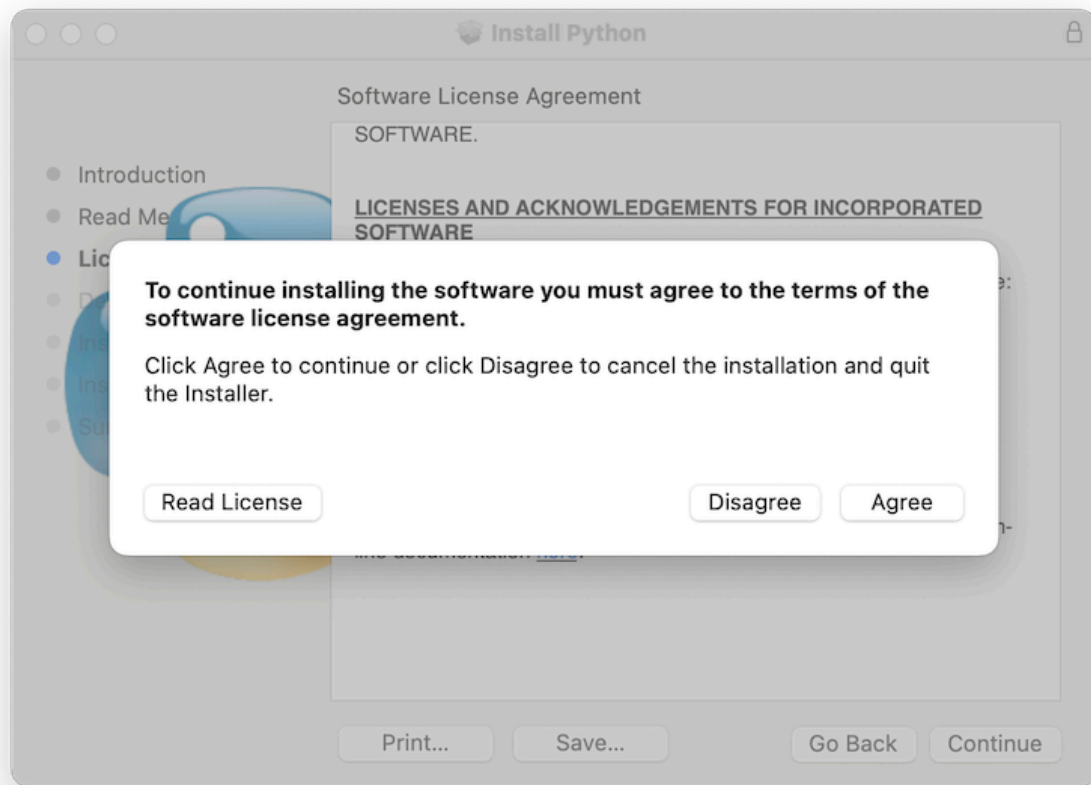
For a default installation, double-click on the downloaded installer package file. This should launch the standard macOS Installer app and display the first of several installer windows steps.



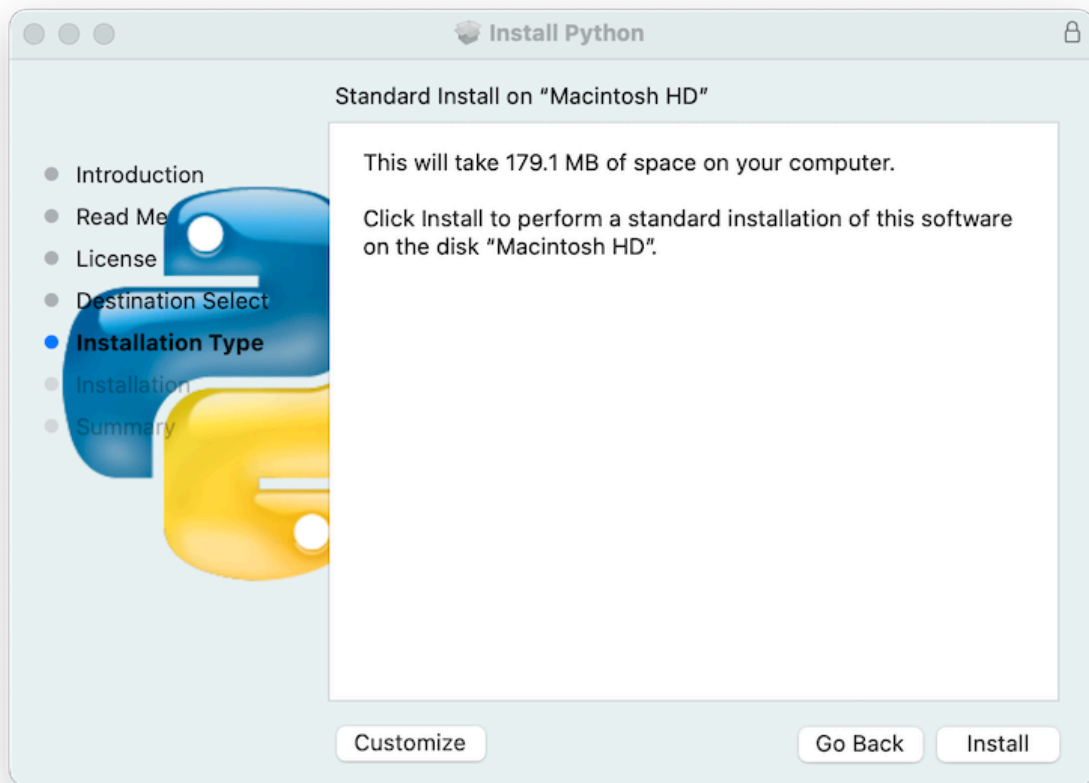
Clicking on the **Continue** button brings up the **Read Me** for this installer. Besides other important information, the **Read Me** documents which Python version is going to be installed and on what versions of macOS it is supported. You may need to scroll through to read the whole file. By default, this **Read Me** will also be installed in `/Applications/Python 3.14/` and available to read anytime.



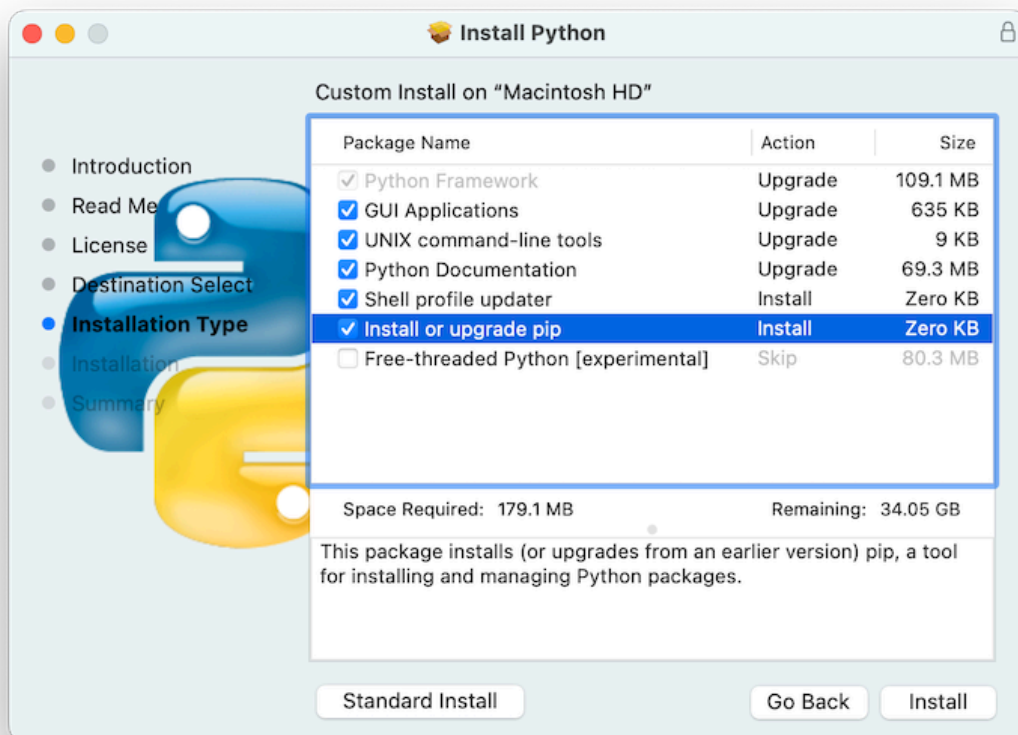
Clicking on **Continue** proceeds to display the license for Python and for other included software. You will then need to **Agree** to the license terms before proceeding to the next step. This license file will also be installed and available to be read later.



After the license terms are accepted, the next step is the **Installation Type** display. For most uses, the standard set of installation operations is appropriate.



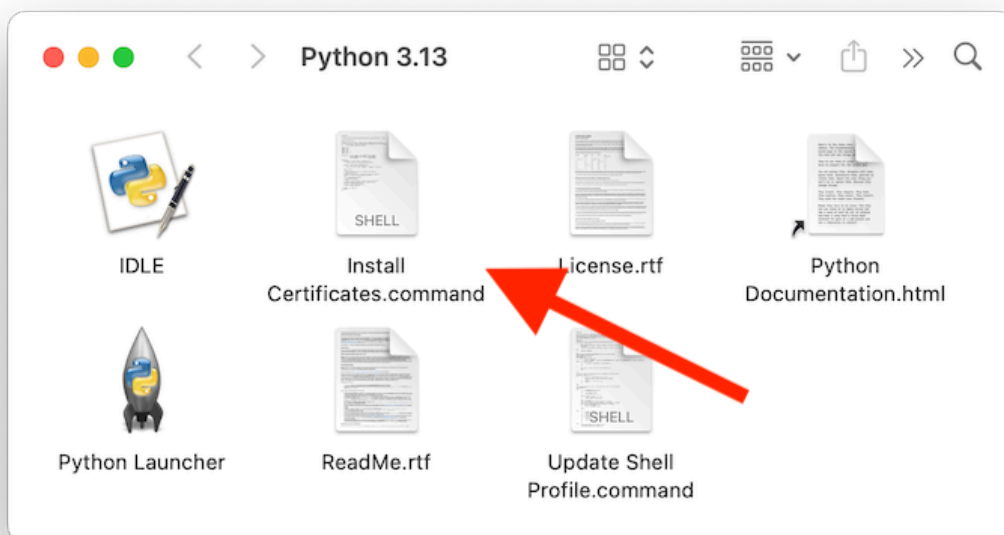
By pressing the **Customize** button, you can choose to omit or select certain package components of the installer. Click on each package name to see a description of what it installs. To also install support for the optional free-threaded feature, see [Installing Free-threaded Binaries](#).



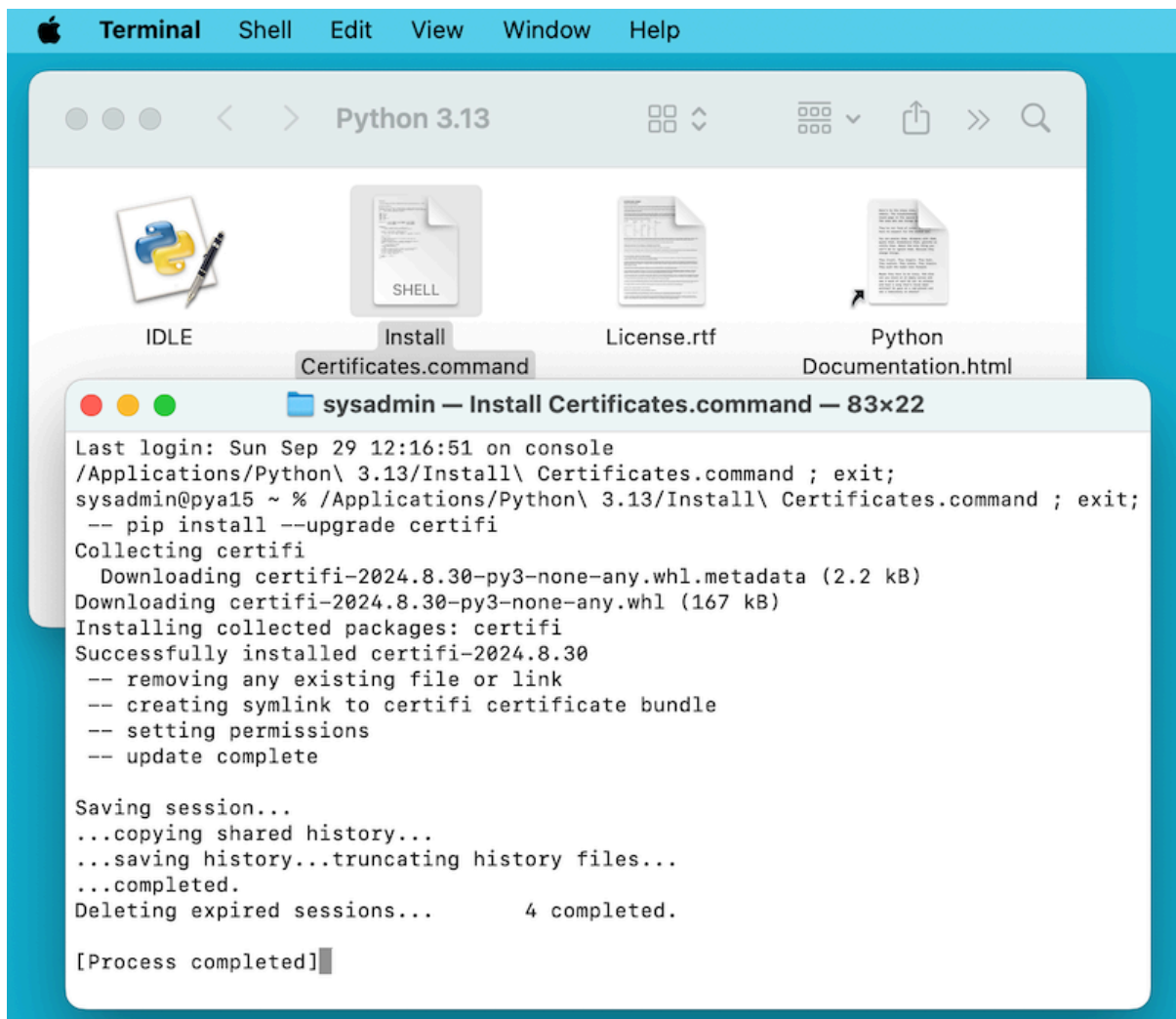
In either case, clicking **Install** will begin the install process by asking permission to install new software. A macOS user name with `Administrator` privilege is needed as the installed Python will be available to all users of the Mac. When the installation is complete, the **Summary** window will appear.



Double-click on the **Install Certificates.command** icon or file in the `/Applications/Python 3.14/` window to complete the installation.



This will open a temporary **Terminal** shell window that will use the new Python to download and install SSL root certificates for its use.



If `Successfully installed certifi` and `update complete` appears in the terminal window, the installation is complete. Close this terminal window and the installer window.

A default install will include:

- A **Python 3.14** folder in your **Applications** folder. In here you find **IDLE**, the development environment that is a standard part of official Python distributions; and **Python Launcher**, which handles double-clicking Python scripts from the macOS **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 Python, you can remove these three things. Symlinks to the Python executable are placed in `/usr/local/bin/`.

i Nota

Recent versions of macOS include a **python3** command in `/usr/bin/python3` that links to a usually older and incomplete version of Python provided by and for use by the Apple development tools, **Xcode** or the **Command Line Tools for Xcode**. You should never modify or attempt to delete this installation, as it is Apple-controlled and is used by Apple-provided or third-party software. If you choose to install a newer Python version from `python.org`, you will have two different but functional Python installations on your computer that can co-exist. The default installer options should ensure that its **python3** will be used instead of the system **python3**.

5.1.2 Cómo ejecutar un *script* de Python

There are two ways to invoke the Python interpreter. If you are familiar with using a Unix shell in a terminal window, you can invoke `python3.14` or `python3` optionally followed by one or more command line options (described in *Línea de comandos y entorno*). The Python tutorial also has a useful section on using Python interactively from a shell.

You can also invoke the interpreter through an integrated development environment. `idle` is a basic editor and interpreter environment which is included with the standard distribution of Python. **IDLE** includes a Help menu that allows you to access Python documentation. If you are completely new to Python, you can read the tutorial introduction in that document.

There are many other editors and IDEs available, see *Editores e IDEs* for more information.

To run a Python script file from the terminal window, you can invoke the interpreter with the name of the script file:

```
python3.14 myscript.py
```

To run your script from the Finder, you can either:

- Drag it to **Python Launcher**.
- Select **Python Launcher** as the default application to open your script (or any `.py` script) through the Finder Info window and double-click it. **Python Launcher** 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.

Be aware that running the script directly from the macOS Finder might produce different results than when running from a terminal window as the script will not be run in the usual shell environment including any setting of environment variables in shell profiles. And, as with any other script or program, be certain of what you are about to run.

5.2 Alternative Distributions

Besides the standard `python.org` for macOS installer, there are third-party distributions for macOS that may include additional functionality. Some popular distributions and their key features:

ActivePython

Installer with multi-platform compatibility, documentation

Anaconda

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

Homebrew

Package manager for macOS including multiple versions of Python and many third-party Python-based packages (including `numpy`, `scipy`, and `pandas`).

MacPorts

Another package manager for macOS including multiple versions of Python and many third-party Python-based packages. May include pre-built versions of Python and many packages for older versions of macOS.

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

5.3 Instalación de paquetes adicionales de Python

Refer to the [Python Packaging User Guide](#) for more information.

5.4 GUI Programming

Hay varias opciones para crear aplicaciones GUI en Mac con Python.

The standard Python GUI toolkit is `tkinter`, based on the cross-platform Tk toolkit (<https://www.tcl.tk>). A macOS-native version of Tk is included with the installer.

PyObjC is a Python binding to Apple's Objective-C/Cocoa framework. Information on PyObjC is available from pyobjc.org.

A number of alternative macOS GUI toolkits are available including:

- **PySide**: Official Python bindings to the Qt GUI toolkit.
- **PyQt**: Alternative Python bindings to Qt.
- **Kivy**: A cross-platform GUI toolkit that supports desktop and mobile platforms.
- **Toga**: Part of the [BeeWare Project](http://beeWare.org); supports desktop, mobile, web and console apps.
- **wxPython**: A cross-platform toolkit that supports desktop operating systems.

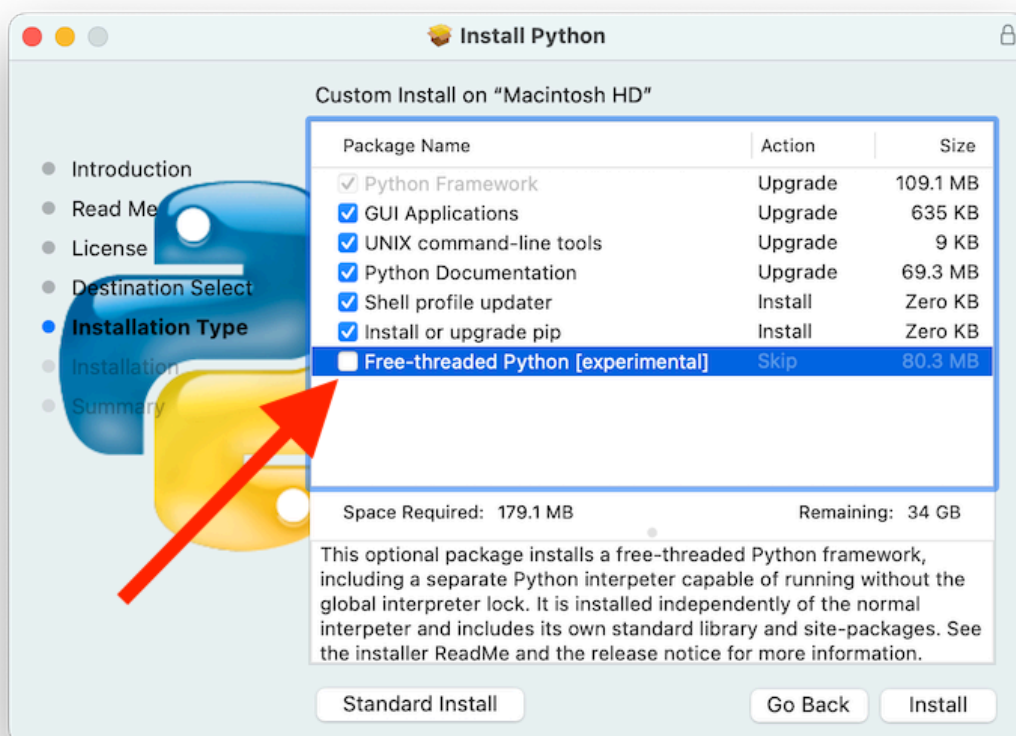
5.5 Advanced Topics

5.5.1 Installing Free-threaded Binaries

Added in version 3.13.

The python.org *Python for macOS* installer package can optionally install an additional build of Python 3.14 that supports **PEP 703**, the free-threading feature (running with the *global interpreter lock* disabled). Check the release page on python.org for possible updated information.

The free-threaded mode is working and continues to be improved, but there is some additional overhead in single-threaded workloads compared to the regular build. Additionally, third-party packages, in particular ones with an *extension module*, may not be ready for use in a free-threaded build, and will re-enable the *GIL*. Therefore, the support for free-threading is not installed by default. It is packaged as a separate install option, available by clicking the **Customize** button on the **Installation Type** step of the installer as described above.



If the box next to the **Free-threaded Python** package name is checked, a separate `PythonT.framework` will also be installed alongside the normal `Python.framework` in `/Library/Frameworks`. This configuration allows a free-threaded Python 3.14 build to co-exist on your system with a traditional (GIL only) Python 3.14 build with minimal risk while installing or testing. This installation layout may change in future releases.

Known cautions and limitations:

- The **UNIX command-line tools** package, which is selected by default, will install links in `/usr/local/bin` for `python3.14t`, the free-threaded interpreter, and `python3.14t-config`, a configuration utility which may be useful for package builders. Since `/usr/local/bin` is typically included in your shell `PATH`, in most cases no changes to your `PATH` environment variables should be needed to use `python3.14t`.
- For this release, the **Shell profile updater** package and the `Update Shell Profile.command` in `/Applications/Python 3.14/` do not support the free-threaded package.
- The free-threaded build and the traditional build have separate search paths and separate `site-packages` directories so, by default, if you need a package available in both builds, it may need to be installed in both. The free-threaded package will install a separate instance of **pip** for use with `python3.14t`.

- To install a package using **pip** without a **venv**:

```
python3.14t -m pip install <package_name>
```

- When working with multiple Python environments, it is usually safest and easiest to create and use virtual environments. This can avoid possible command name conflicts and confusion about which Python is in use:

```
python3.14t -m venv <venv_name>
```

then **activate**.

- To run a free-threaded version of IDLE:

```
python3.14t -m idlelib
```

- The interpreters in both builds respond to the same *PYTHON environment variables* which may have unexpected results, for example, if you have `PYTHONPATH` set in a shell profile. If necessary, there are *command line options* like `-E` to ignore these environment variables.
- The free-threaded build links to the third-party shared libraries, such as OpenSSL and Tk, installed in the traditional framework. This means that both builds also share one set of trust certificates as installed by the **Install Certificates.command** script, thus it only needs to be run once.
- If you cannot depend on the link in `/usr/local/bin` pointing to the `python.org` free-threaded `python3.14t` (for example, if you want to install your own version there or some other distribution does), you can explicitly set your shell `PATH` environment variable to include the `PythonT.framework` bin directory:

```
export PATH="/Library/Frameworks/PythonT.framework/Versions/3.14/bin": "$PATH"
```

The traditional framework installation by default does something similar, except for `Python.framework`. Be aware that having both framework `bin` directories in `PATH` can lead to confusion if there are duplicate names like `python3.14` in both; which one is actually used depends on the order they appear in `PATH`. The which `python3.x` or which `python3.xt` commands can show which path is being used. Using virtual environments can help avoid such ambiguities. Another option might be to create a shell **alias** to the desired interpreter, like:

```
alias py3.14="/Library/Frameworks/Python.framework/Versions/3.14/bin/python3.  
→14"
```

```
alias py3.14t="/Library/Frameworks/PythonT.framework/Versions/3.14/bin/python3.  
→14t"
```

5.5.2 Installing using the command line

If you want to use automation to install the `python.org` installer package (rather than by using the familiar macOS **Installer** GUI app), the macOS command line **installer** utility lets you select non-default options, too. If you are not familiar with **installer**, it can be somewhat cryptic (see **man installer** for more information). As an example, the following shell snippet shows one way to do it, using the 3.14.0b2 release and selecting the free-threaded interpreter option:

```
RELEASE="python-3.14.0b2-macos11.pkg"

# download installer pkg
curl -O https://www.python.org/ftp/python/3.14.0/${RELEASE}

# create installer choicechanges to customize the install:
#   enable the PythonTFramework-3.14 package
#   while accepting the other defaults (install all other packages)
cat > ./choicechanges.plist <<EOF
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
↳PropertyList-1.0.dtd">
<plist version="1.0">
<array>
    <dict>
        <key>attributeSetting</key>
        <integer>1</integer>
        <key>choiceAttribute</key>
        <string>selected</string>
        <key>choiceIdentifier</key>
        <string>org.python.Python.PythonTFramework-3.14</string>
    </dict>
</array>
</plist>
EOF

sudo installer -pkg ./${RELEASE} -applyChoiceChangesXML ./choicechanges.plist
↳-target /
```

You can then test that both installer builds are now available with something like:

```
$ # test that the free-threaded interpreter was installed if the Unix Command
↳Tools package was enabled
$ /usr/local/bin/python3.14t -VV
Python 3.14.0b2 free-threading build (v3.14.0b2:3a83b172af, Jun  5 2024, 12:57:31)
↳[Clang 15.0.0 (clang-1500.3.9.4)]
$ # and the traditional interpreter
$ /usr/local/bin/python3.14 -VV
Python 3.14.0b2 (v3.14.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0
↳(clang-1500.3.9.4)]
$ # test that they are also available without the prefix if /usr/local/bin is on
↳$PATH
$ python3.14t -VV
Python 3.14.0b2 free-threading build (v3.14.0b2:3a83b172af, Jun  5 2024, 12:57:31)
↳[Clang 15.0.0 (clang-1500.3.9.4)]
$ python3.14 -VV
Python 3.14.0b2 (v3.14.0b2:3a83b172af, Jun  5 2024, 12:50:24) [Clang 15.0.0
↳(clang-1500.3.9.4)]
```

Nota

Current `python.org` installers only install to fixed locations like `/Library/Frameworks/`, `/Applications`, and `/usr/local/bin`. You cannot use the **installer** `-domain` option to install to other locations.

5.5.3 Distributing Python Applications

A range of tools exist for converting your Python code into a standalone distributable application:

- [py2app](#): Supports creating macOS `.app` bundles from a Python project.
- [Briefcase](#): Part of the [BeeWare Project](#); a cross-platform packaging tool that supports creation of `.app` bundles on macOS, as well as managing signing and notarization.
- [PyInstaller](#): A cross-platform packaging tool that creates a single file or folder as a distributable artifact.

5.5.4 App Store Compliance

Apps submitted for distribution through the macOS App Store must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged. Therefore, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a [patch file](#) that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when CPython is configured with the `--with-app-store-compliance` option.

This patch is not normally required to use CPython on a Mac; nor is it required if you are distributing an app *outside* the macOS App Store. It is *only* required if you are using the macOS App Store as a distribution channel.

5.6 Otros recursos

The [python.org Help page](#) has links to many useful resources. The [Pythonmac-SIG mailing list](#) is another support resource specifically for Python users and developers on the Mac.

Using Python on Android

Python on Android is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a `python` executable and entering commands at an interactive prompt, or by running a Python script.

On Android, there is no concept of installing as a system resource. The only unit of software distribution is an «app». There is also no console where you could run a `python` executable, or interact with a Python REPL.

As a result, the only way you can use Python on Android is in embedded mode – that is, by writing a native Android application, embedding a Python interpreter using `libpython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged into your app for its own private use.

The Python standard library has some notable omissions and restrictions on Android. See the API availability guide for details.

6.1 Adding Python to an Android app

Most app developers should use one of the following tools, which will provide a much easier experience:

- [Briefcase](#), from the BeeWare project
- [Buildozer](#), from the Kivy project
- [Chaquopy](#)
- [pyqtdeploy](#)
- [Termux](#)

If you're sure you want to do all of this manually, read on. You can use the [testbed](#) app as a guide; each step below contains a link to the relevant file.

- Build Python by following the instructions in [Android/README.md](#). This will create the directory `cross-build/HOST/prefix`.
- Add code to your [build.gradle](#) file to copy the following items into your project. All except your own Python code can be copied from `prefix/lib`:
 - In your JNI libraries:
 - * `libpython*.so`
 - * `lib*_python.so` (external libraries such as OpenSSL)

- In your assets:
 - * `python*. *` (the Python standard library)
 - * `python*/site-packages` (your own Python code)
- Add code to your app to [extract the assets to the filesystem](#).
- Add code to your app to [start Python in embedded mode](#). This will need to be C code called via JNI.

6.2 Building a Python package for Android

Python packages can be built for Android as wheels and released on PyPI. The recommended tool for doing this is [cibuildwheel](#), which automates all the details of setting up a cross-compilation environment, building the wheel, and testing it on an emulator.

Using Python on iOS

Authors

Russell Keith-Magee (2024-03)

Python on iOS is unlike Python on desktop platforms. On a desktop platform, Python is generally installed as a system resource that can be used by any user of that computer. Users then interact with Python by running a **python** executable and entering commands at an interactive prompt, or by running a Python script.

On iOS, there is no concept of installing as a system resource. The only unit of software distribution is an «app». There is also no console where you could run a **python** executable, or interact with a Python REPL.

As a result, the only way you can use Python on iOS is in embedded mode - that is, by writing a native iOS application, and embedding a Python interpreter using `libPython`, and invoking Python code using the Python embedding API. The full Python interpreter, the standard library, and all your Python code is then packaged as a standalone bundle that can be distributed via the iOS App Store.

If you're looking to experiment for the first time with writing an iOS app in Python, projects such as [BeeWare](#) and [Kivy](#) will provide a much more approachable user experience. These projects manage the complexities associated with getting an iOS project running, so you only need to deal with the Python code itself.

7.1 Python at runtime on iOS

7.1.1 iOS version compatibility

The minimum supported iOS version is specified at compile time, using the `--host` option to `configure`. By default, when compiled for iOS, Python will be compiled with a minimum supported iOS version of 13.0. To use a different minimum iOS version, provide the version number as part of the `--host` argument - for example, `--host=arm64-apple-ios15.4-simulator` would compile an ARM64 simulator build with a deployment target of 15.4.

7.1.2 Platform identification

When executing on iOS, `sys.platform` will report as `ios`. This value will be returned on an iPhone or iPad, regardless of whether the app is running on the simulator or a physical device.

Information about the specific runtime environment, including the iOS version, device model, and whether the device is a simulator, can be obtained using `platform.ios_ver()`. `platform.system()` will report `ios` or `iPadOS`, depending on the device.

`os.uname()` reports kernel-level details; it will report a name of `Darwin`.

7.1.3 Standard library availability

The Python standard library has some notable omissions and restrictions on iOS. See the API availability guide for iOS for details.

7.1.4 Binary extension modules

One notable difference about iOS as a platform is that App Store distribution imposes hard requirements on the packaging of an application. One of these requirements governs how binary extension modules are distributed.

The iOS App Store requires that *all* binary modules in an iOS app must be dynamic libraries, contained in a framework with appropriate metadata, stored in the `Frameworks` folder of the packaged app. There can be only a single binary per framework, and there can be no executable binary material outside the `Frameworks` folder.

This conflicts with the usual Python approach for distributing binaries, which allows a binary extension module to be loaded from any location on `sys.path`. To ensure compliance with App Store policies, an iOS project must post-process any Python packages, converting `.so` binary modules into individual standalone frameworks with appropriate metadata and signing. For details on how to perform this post-processing, see the guide for [adding Python to your project](#).

To help Python discover binaries in their new location, the original `.so` file on `sys.path` is replaced with a `.fwork` file. This file is a text file containing the location of the framework binary, relative to the app bundle. To allow the framework to resolve back to the original location, the framework must contain a `.origin` file that contains the location of the `.fwork` file, relative to the app bundle.

For example, consider the case of an import `from foo.bar import _whiz`, where `_whiz` is implemented with the binary module `sources/foo/bar/_whiz.abi3.so`, with `sources` being the location registered on `sys.path`, relative to the application bundle. This module *must* be distributed as `Frameworks/foo.bar._whiz.framework/foo.bar._whiz` (creating the framework name from the full import path of the module), with an `Info.plist` file in the `.framework` directory identifying the binary as a framework. The `foo.bar._whiz` module would be represented in the original location with a `sources/foo/bar/_whiz.abi3.fwork` marker file, containing the path `Frameworks/foo.bar._whiz/foo.bar._whiz`. The framework would also contain `Frameworks/foo.bar._whiz.framework/foo.bar._whiz.origin`, containing the path to the `.fwork` file.

When running on iOS, the Python interpreter will install an `AppleFrameworkLoader` that is able to read and import `.fwork` files. Once imported, the `__file__` attribute of the binary module will report as the location of the `.fwork` file. However, the `ModuleSpec` for the loaded module will report the `origin` as the location of the binary in the framework folder.

7.1.5 Compiler stub binaries

Xcode doesn't expose explicit compilers for iOS; instead, it uses an `xcrun` script that resolves to a full compiler path (e.g., `xcrun --sdk iphoneos clang` to get the `clang` for an iPhone device). However, using this script poses two problems:

- The output of `xcrun` includes paths that are machine specific, resulting in a `sysconfig` module that cannot be shared between users; and
- It results in `CC/CPP/LD/AR` definitions that include spaces. There is a lot of C ecosystem tooling that assumes that you can split a command line at the first space to get the path to the compiler executable; this isn't the case when using `xcrun`.

To avoid these problems, Python provided stubs for these tools. These stubs are shell script wrappers around the underlying `xcrun` tools, distributed in a `bin` folder distributed alongside the compiled iOS framework. These scripts are relocatable, and will always resolve to the appropriate local system paths. By including these scripts in the `bin` folder that accompanies a framework, the contents of the `sysconfig` module becomes useful for end-users to compile their own modules. When compiling third-party Python modules for iOS, you should ensure these stub binaries are on your path.

7.2 Installing Python on iOS

7.2.1 Tools for building iOS apps

Building for iOS requires the use of Apple's Xcode tooling. It is strongly recommended that you use the most recent stable release of Xcode. This will require the use of the most (or second-most) recently released macOS version, as Apple does not maintain Xcode for older macOS versions. The Xcode Command Line Tools are not sufficient for iOS development; you need a *full* Xcode install.

If you want to run your code on the iOS simulator, you'll also need to install an iOS Simulator Platform. You should be prompted to select an iOS Simulator Platform when you first run Xcode. Alternatively, you can add an iOS Simulator Platform by selecting from the Platforms tab of the Xcode Settings panel.

7.2.2 Adding Python to an iOS project

Python can be added to any iOS project, using either Swift or Objective C. The following examples will use Objective C; if you are using Swift, you may find a library like [PythonKit](#) to be helpful.

To add Python to an iOS Xcode project:

1. Build or obtain a Python `XCFramework`. See the instructions in [iOS/README.rst](#) (in the CPython source distribution) for details on how to build a Python `XCFramework`. At a minimum, you will need a build that supports `arm64-apple-ios`, plus one of either `arm64-apple-ios-simulator` or `x86_64-apple-ios-simulator`.
2. Drag the `XCframework` into your iOS project. In the following instructions, we'll assume you've dropped the `XCframework` into the root of your project; however, you can use any other location that you want by adjusting paths as needed.
3. Drag the `iOS/Resources/dylib-Info-template.plist` file into your project, and ensure it is associated with the app target.
4. Add your application code as a folder in your Xcode project. In the following instructions, we'll assume that your user code is in a folder named `app` in the root of your project; you can use any other location by adjusting paths as needed. Ensure that this folder is associated with your app target.
5. Select the app target by selecting the root node of your Xcode project, then the target name in the sidebar that appears.
6. In the «General» settings, under «Frameworks, Libraries and Embedded Content», add `Python.xcframework`, with «Embed & Sign» selected.
7. In the «Build Settings» tab, modify the following:
 - Build Options
 - User Script Sandboxing: No
 - Enable Testability: Yes
 - Search Paths
 - Framework Search Paths: `$(PROJECT_DIR)`
 - Header Search Paths: `"$(BUILT_PRODUCTS_DIR)/Python.framework/Headers"`
 - Apple Clang - Warnings - All languages
 - Quoted Include In Framework Header: No
8. Add a build step that copies the Python standard library into your app. In the «Build Phases» tab, add a new «Run Script» build step *before* the «Embed Frameworks» step, but *after* the «Copy Bundle Resources» step. Name the step «Install Target Specific Python Standard Library», disable the «Based on dependency analysis» checkbox, and set the script content to:

```

set -e

mkdir -p "$CODESIGNING_FOLDER_PATH/python/lib"
if [ "$EFFECTIVE_PLATFORM_NAME" = "-iphonesimulator" ]; then
    echo "Installing Python modules for iOS Simulator"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64_x86_64-
    ↪imulator/lib/" "$CODESIGNING_FOLDER_PATH/python/lib/"
else
    echo "Installing Python modules for iOS Device"
    rsync -au --delete "$PROJECT_DIR/Python.xcframework/ios-arm64/lib/" "
    ↪$CODESIGNING_FOLDER_PATH/python/lib/"
fi

```

Note that the name of the simulator «slice» in the XCframework may be different, depending the CPU architectures your XCFramework supports.

9. Add a second build step that processes the binary extension modules in the standard library into «Framework» format. Add a «Run Script» build step *directly after* the one you added in step 8, named «Prepare Python Binary Modules». It should also have «Based on dependency analysis» unchecked, with the following script content:

```

set -e

install_dylib () {
    INSTALL_BASE=$1
    FULL_EXT=$2

    # The name of the extension file
    EXT=$(basename "$FULL_EXT")
    # The location of the extension file, relative to the bundle
    RELATIVE_EXT=${FULL_EXT#$CODESIGNING_FOLDER_PATH/}
    # The path to the extension file, relative to the install base
    PYTHON_EXT=${RELATIVE_EXT/$INSTALL_BASE/}
    # The full dotted name of the extension module, constructed from the file_
    ↪path.
    FULL_MODULE_NAME=$(echo $PYTHON_EXT | cut -d "." -f 1 | tr "/" ".");
    # A bundle identifier; not actually used, but required by Xcode framework_
    ↪packaging
    FRAMEWORK_BUNDLE_ID=$(echo $PRODUCT_BUNDLE_IDENTIFIER.$FULL_MODULE_NAME |_
    ↪tr "_" "-")
    # The name of the framework folder.
    FRAMEWORK_FOLDER="Frameworks/$FULL_MODULE_NAME.framework"

    # If the framework folder doesn't exist, create it.
    if [ ! -d "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER" ]; then
        echo "Creating framework for $RELATIVE_EXT"
        mkdir -p "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER"
        cp "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist" "$CODESIGNING_
        ↪FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleExecutable -string "$FULL_MODULE_NAME" "
        ↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
        plutil -replace CFBundleIdentifier -string "$FRAMEWORK_BUNDLE_ID" "
        ↪$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/Info.plist"
    fi

    echo "Installing binary for $FRAMEWORK_FOLDER/$FULL_MODULE_NAME"
    mv "$FULL_EXT" "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_FOLDER/$FULL_MODULE_

```

(continúe en la próxima página)

(proviene de la página anterior)

```

→NAME"
    # Create a placeholder .fwork file where the .so was
    echo "$FRAMEWORK_FOLDER/$FULL_MODULE_NAME" > ${FULL_EXT%.so}.fwork
    # Create a back reference to the .so file location in the framework
    echo "${RELATIVE_EXT%.so}.fwork" > "$CODESIGNING_FOLDER_PATH/$FRAMEWORK_
→FOLDER/$FULL_MODULE_NAME.origin"
}

PYTHON_VER=$(ls -1 "$CODESIGNING_FOLDER_PATH/python/lib")
echo "Install Python $PYTHON_VER standard library extension modules..."
find "$CODESIGNING_FOLDER_PATH/python/lib/$PYTHON_VER/lib-dynload" -name "*.so
→" | while read FULL_EXT; do
    install_dylib python/lib/$PYTHON_VER/lib-dynload/ "$FULL_EXT"
done

# Clean up dylib template
rm -f "$CODESIGNING_FOLDER_PATH/dylib-Info-template.plist"

echo "Signing frameworks as $EXPANDED_CODE_SIGN_IDENTITY_NAME ($EXPANDED_CODE_
→SIGN_IDENTITY)..."
find "$CODESIGNING_FOLDER_PATH/Frameworks" -name "*.framework" -exec /usr/bin/
→codesign --force --sign "$EXPANDED_CODE_SIGN_IDENTITY" ${OTHER_CODE_SIGN_
→FLAGS:-} -o runtime --timestamp=none --preserve-metadata=identifier,
→entitlements,flags --generate-entitlement-der "{}" \;

```

10. Add Objective C code to initialize and use a Python interpreter in embedded mode. You should ensure that:

- UTF-8 mode (PyPreConfig.utf8_mode) is *enabled*;
- Buffered stdio (PyConfig.buffered_stdio) is *disabled*;
- Writing bytecode (PyConfig.write_bytecode) is *disabled*;
- Signal handlers (PyConfig.install_signal_handlers) are *enabled*;
- System logging (PyConfig.use_system_logger) is *enabled* (optional, but strongly recommended; this is enabled by default);
- `PYTHONHOME` for the interpreter is configured to point at the `python` subfolder of your app's bundle; and
- The `PYTHONPATH` for the interpreter includes:
 - the `python/lib/python3.X` subfolder of your app's bundle,
 - the `python/lib/python3.X/lib-dynload` subfolder of your app's bundle, and
 - the `app` subfolder of your app's bundle

Your app's bundle location can be determined using `[[NSBundle mainBundle] resourcePath]`.

Steps 8, 9 and 10 of these instructions assume that you have a single folder of pure Python application code, named `app`. If you have third-party binary modules in your app, some additional steps will be required:

- You need to ensure that any folders containing third-party binaries are either associated with the app target, or copied in as part of step 8. Step 8 should also purge any binaries that are not appropriate for the platform a specific build is targeting (i.e., delete any device binaries if you're building an app targeting the simulator).
- Any folders that contain third-party binaries must be processed into framework form by step 9. The invocation of `install_dylib` that processes the `lib-dynload` folder can be copied and adapted for this purpose.
- If you're using a separate folder for third-party packages, ensure that folder is included as part of the `PYTHONPATH` configuration in step 10.

- If any of the folders that contain third-party packages will contain `.pth` files, you should add that folder as a *site directory* (using `site.addsitedir()`), rather than adding to `PYTHONPATH` or `sys.path` directly.

7.2.3 Testing a Python package

The CPython source tree contains a [testbed project](#) that is used to run the CPython test suite on the iOS simulator. This testbed can also be used as a testbed project for running your Python library's test suite on iOS.

After building or obtaining an iOS XCFramework (See [iOS/README.rst](#) for details), create a clone of the Python iOS testbed project by running:

```
$ python iOS/testbed clone --framework <path/to/Python.xcframework> --app <path/to/  
↪module1> --app <path/to/module2> app-testbed
```

You will need to modify the `iOS/testbed` reference to point to that directory in the CPython source tree; any folders specified with the `--app` flag will be copied into the cloned testbed project. The resulting testbed will be created in the `app-testbed` folder. In this example, the `module1` and `module2` would be importable modules at runtime. If your project has additional dependencies, they can be installed into the `app-testbed/iOSTestbed/app_packages` folder (using `pip install --target app-testbed/iOSTestbed/app_packages` or similar).

You can then use the `app-testbed` folder to run the test suite for your app. For example, if `module1.tests` was the entry point to your test suite, you could run:

```
$ python app-testbed run -- module1.tests
```

This is the equivalent of running `python -m module1.tests` on a desktop Python build. Any arguments after the `--` will be passed to the testbed as if they were arguments to `python -m` on a desktop machine.

You can also open the testbed project in Xcode by running:

```
$ open app-testbed/iOSTestbed.xcodeproj
```

This will allow you to use the full Xcode suite of tools for debugging.

The arguments used to run the test suite are defined as part of the test plan. To modify the test plan, select the test plan node of the project tree (it should be the first child of the root node), and select the «Configurations» tab. Modify the «Arguments Passed On Launch» value to change the testing arguments.

The test plan also disables parallel testing, and specifies the use of the `iOSTestbed.lldbinit` file for providing configuration of the debugger. The default debugger configuration disables automatic breakpoints on the `SIGINT`, `SIGUSR1`, `SIGUSR2`, and `SIGXFSZ` signals.

7.3 App Store Compliance

The only mechanism for distributing apps to third-party iOS devices is to submit the app to the iOS App Store; apps submitted for distribution must pass Apple's app review process. This process includes a set of automated validation rules that inspect the submitted application bundle for problematic code.

The Python standard library contains some code that is known to violate these automated rules. While these violations appear to be false positives, Apple's review rules cannot be challenged; so, it is necessary to modify the Python standard library for an app to pass App Store review.

The Python source tree contains a [patch file](#) that will remove all code that is known to cause issues with the App Store review process. This patch is applied automatically when building for iOS.

Existen numerosos IDEs que admiten el lenguaje de programación Python. Muchos editores e IDEs proporcionan resaltado de sintaxis, herramientas de depuración y comprobaciones de **PEP 8**.

8.1 IDLE — Python editor and shell

IDLE is Python's Integrated Development and Learning Environment and is generally bundled with Python installs. If you are on Linux and do not have IDLE installed see [Installing IDLE on Linux](#). For more information see the IDLE docs.

8.2 Other Editors and IDEs

Python's community wiki has information submitted by the community on Editors and IDEs. Please go to [Python Editors and Integrated Development Environments](#) for a comprehensive list.

>>>

The default Python prompt of the *interactive* shell. Often seen for code examples which can be executed interactively in the interpreter.

...

Puede referirse a:

- 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.
- La constante incorporada `Ellipsis`.

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`.

annotate function

A function that can be called to retrieve the *annotations* of an object. This function is accessible as the `__annotate__` attribute of functions, classes, and modules. Annotate functions are a subset of *evaluate functions*.

anotación

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

Annotations of local variables cannot be accessed at runtime, but annotations of global variables, class attributes, and functions can be retrieved by calling `annotationlib.get_annotations()` on modules, classes, and functions, respectively.

See *variable annotation*, *function annotation*, **PEP 484**, **PEP 526**, and **PEP 649**, which describe this functionality. Also see *annotations-howto* for best practices on working with annotations.

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__()` y `__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

An object created by an *asynchronous generator* function.

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

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *asynchronous generator iterator* effectively resumes with another *awaitable* returned by `__anext__()`, it picks up where it left off. See [PEP 492](#) and [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 `__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](#).

attached thread state

A *thread state* that is active for the current OS thread.

When a *thread state* is attached, the OS thread has access to the full Python C API and can safely invoke the bytecode interpreter.

Unless a function explicitly notes otherwise, attempting to call the C API without an attached thread state will result in a fatal error or undefined behavior. A thread state can be attached and detached explicitly by the

user through the C API, or implicitly by the runtime, including during blocking C calls and by the bytecode interpreter in between calls.

On most builds of Python, having an attached thread state implies that the caller holds the *GIL* for the current interpreter, so only one OS thread can have an attached thread state at a given moment. In *free-threaded* builds of Python, threads can concurrently hold an attached thread state, allowing for true parallelism of the bytecode interpreter.

atributo

Un valor asociado a un objeto al que se suele hacer referencia por su nombre utilizando expresiones punteadas. Por ejemplo, si un objeto *o* tiene un atributo *a* se referenciaría como *o.a*.

Es posible dar a un objeto un atributo cuyo nombre no sea un identificador definido por `__identifiers__`, por ejemplo usando `setattr()`, si el objeto lo permite. Dicho atributo no será accesible utilizando una expresión con puntos, y en su lugar deberá ser recuperado con `getattr()`.

a la espera

Un objeto que puede utilizarse en una expresión `await`. Puede ser una *corutina* o un objeto con un método `__await__()`. Véase también [PEP 492](#).

BDFL

Sigla de *Benevolent Dictator For Life*, benevolente dictador vitalicio, es decir [Guido van Rossum](#), el creador de Python.

archivo binario

A *file object* able to read and write *bytes-like objects*. Examples of binary files are files opened in binary mode ('rb', 'wb' or 'rb+'), `sys.stdin.buffer`, `sys.stdout.buffer`, and instances of `io.BytesIO` and `gzip.GzipFile`.

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

referencia prestada

En la API C de Python, una referencia prestada es una referencia a un objeto, donde el código usando el objeto no posee la referencia. Se convierte en un puntero colgante si se destruye el objeto. Por ejemplo, una recolección de basura puede eliminar el último *strong reference* del objeto y así destruirlo.

Se recomienda llamar a `Py_INCREF()` en la *referencia prestada* para convertirla en una *referencia fuerte* in situ, excepto cuando el objeto no se puede destruir antes del último uso de la referencia prestada. La función `Py_NewRef()` se puede utilizar para crear una nueva *referencia fuerte*.

objetos tipo binarios

Un objeto que soporta `bufferobjects` y puede exportar un búfer *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 búfer 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 correr 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 máquinas 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`.

callable

Un callable es un objeto que puede ser llamado, posiblemente con un conjunto de argumentos (véase *argument*), con la siguiente sintaxis:

```
callable(argument1, argument2, argumentN)
```

Una *function*, y por extensión un *method*, es un callable. Una instancia de una clase que implementa el método `__call__()` también es un callable.

retrollamada

Una función de subrutina que se pasa como un argumento para ejecutarse en algún momento en el futuro.

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).

closure variable

A *free variable* referenced from a *nested scope* that is defined in an outer scope rather than being resolved at runtime from the globals or builtin namespaces. May be explicitly defined with the `nonlocal` keyword to allow write access, or implicitly defined if the variable is only being read.

For example, in the `inner` function in the following code, both `x` and `print` are *free variables*, but only `x` is a *closure variable*:

```
def outer():
    x = 0
    def inner():
        nonlocal x
        x += 1
        print(x)
    return inner
```

Due to the `codeobject.co_freevars` attribute (which, despite its name, only includes the names of closure variables rather than listing all referenced free variables), the more general *free variable* term is sometimes used even when the intended meaning is to refer specifically to closure variables.

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 `cmath`. El uso de números complejos es matemática bastante avanzada. Si no le parecen necesarios, puede ignorarlos sin inconvenientes.

context

This term has different meanings depending on where and how it is used. Some common meanings:

- The temporary state or environment established by a *context manager* via a `with` statement.
- The collection of keyvalue bindings associated with a particular `contextvars.Context` object and accessed via `ContextVar` objects. Also see *context variable*.
- A `contextvars.Context` object. Also see *current context*.

context management protocol

The `__enter__()` and `__exit__()` methods called by the `with` statement. See [PEP 343](#).

administrador de contextos

An object which implements the *context management protocol* and controls the environment seen in a `with` statement. See [PEP 343](#).

variable de contexto

A variable whose value depends on which context is the *current context*. Values are accessed via

`contextvars.ContextVar` objects. Context variables are primarily used to isolate state between concurrent asynchronous tasks.

contiguo

Un búfer es considerado contiguo con precisión si es *C-contiguo* o *Fortran contiguo*. Los búferes 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-contiguos, el último índice varía más velozmente en el orden de las direcciones de memoria. Sin embargo, en arreglos Fortran contiguos, el primer índice varía más rápidamente.

corrutina

Las corrutinas son una forma más generalizadas de las subrutinas. A las subrutinas se ingresa por un punto y se sale por otro punto. Las corrutinas pueden ser iniciadas, finalizadas y reanudadas en muchos puntos diferentes. Pueden ser implementadas con la sentencia `async def`. Vea además [PEP 492](#).

función corrutina

Una 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*.

current context

The *context* (`contextvars.Context` object) that is currently used by `ContextVar` objects to access (get or set) the values of *context variables*. Each thread has its own current context. Frameworks for executing asynchronous tasks (see `asyncio`) associate each task with a context which becomes the current context whenever the task starts or resumes execution.

cyclic isolate

A subgroup of one or more objects that reference each other in a reference cycle, but are not referenced by objects outside the group. The goal of the *cyclic garbage collector* is to identify these groups and break the reference cycles so that the memory can be reclaimed.

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 `staticmethod()`.

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

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

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

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

Any object which defines the methods `__get__()`, `__set__()`, or `__delete__()`. When a class attribute is a descriptor, its special binding behavior is triggered upon attribute lookup. Normally, using `a.b` to get, set or delete an attribute looks up the object named `b` in the class dictionary for `a`, but if `b` is a descriptor, the respective descriptor method gets called. Understanding descriptors is a key to a deep understanding of Python because they are the basis for many features including functions, methods, properties, class methods, static methods, and reference to super classes.

Para obtener más información sobre los métodos de los descriptores, consulte `descriptors` o Guía práctica de uso de los descriptores.

diccionario

An associative array, where arbitrary keys are mapped to values. The keys can be any object with `__hash__()` and `__eq__()` methods. Called a hash in Perl.

comprensión de diccionarios

Una forma compacta de procesar todos o parte de los elementos en un iterable y retornar un diccionario con los resultados. `results = {n: n ** 2 for n in range(10)}` genera un diccionario que contiene la clave `n` asignada al valor `n ** 2`. Ver `comprehensions`.

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

A string literal which appears as the first expression in a class, function or module. While ignored when the suite is executed, it is recognized by the compiler and put into the `__doc__` attribute of the enclosing class, function or module. Since it is available via introspection, it is the canonical place for documentation of the object.

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 pregunta con `hasattr()` o *EAFP*.

dunder

An informal short-hand for «double underscore», used when talking about a *special method*. For example, `__init__` is often pronounced «dunder init».

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.

evaluate function

A function that can be called to evaluate a lazily evaluated attribute of an object, such as the value of type aliases created with the `type` statement.

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 *statements* 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

f-strings

String literals prefixed with `f` or `F` are commonly called «f-strings» which is short for formatted string literals. See also [PEP 498](#).

objeto archivo

An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource.

Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

Existen tres categorías de objetos archivo: crudos *raw archivos binarios*, con búfer *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*.

codificación del sistema de archivos y manejador de errores

Controlador de errores y codificación utilizado por Python para decodificar bytes del sistema operativo y codificar Unicode en el sistema operativo.

La codificación del sistema de archivos debe garantizar la decodificación exitosa de todos los bytes por debajo de 128. Si la codificación del sistema de archivos no proporciona esta garantía, las funciones de API pueden lanzar `UnicodeError`.

Las funciones `sys.getfilesystemencoding()` y `sys.getfilesystemencodeerrors()` se pueden utilizar para obtener la codificación del sistema de archivos y el controlador de errores.

La *codificación del sistema de archivos y el manejador de errores* se configuran al inicio de Python mediante la función `PyConfig_Read()`: consulte los miembros `filesystem_encoding` y `filesystem_errors` de `PyConfig`.

Vea también *locale encoding*.

buscador

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

There are two types of finder: *meta path finders* for use with `sys.meta_path`, and *path entry finders* for use with `sys.path_hooks`.

See finders-and-loaders and `importlib` for much more detail.

división entera a la baja

Una división matemática que se redondea hacia el entero menor más cercano. El operador de la división entera a la baja 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](#).

free threading

A threading model where multiple threads can run Python bytecode simultaneously within the same interpreter. This is in contrast to the *global interpreter lock* which allows only one thread to execute Python bytecode at a time. See [PEP 703](#).

free variable

Formally, as defined in the language execution model, a free variable is any variable used in a namespace which is not a local variable in that namespace. See *closure variable* for an example. Pragmatically, due to the name of the `codeobject.co_freevars` attribute, the term is also sometimes used as a synonym for *closure variable*.

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 *indicadores de tipo*, por ejemplo, se espera que una función tome dos argumentos de clase `int` y también se espera que retorne 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*.

Consulte [variable annotation](#) y [PEP 484](#), que describen esta funcionalidad. Consulte también [annotations-howto](#) para conocer las mejores prácticas sobre cómo trabajar con anotaciones.

`__future__`

Un future statement, `from __future__ import <feature>`, indica al compilador que compile el módulo actual utilizando una sintaxis o semántica que se convertirá en estándar en una versión futura de Python. El módulo `__future__` documenta los posibles valores de *feature*. Al importar este módulo y evaluar sus variables, puede ver cuándo se agregó por primera vez una nueva característica al lenguaje y cuándo se convertirá (o se convirtió) en la predeterminada:

```
>>> 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*.

Each `yield` temporarily suspends processing, remembering the execution state (including local variables and pending try-statements). When the *generator iterator* resumes, it picks up where it left off (in contrast to functions which start fresh on every invocation).

expresión generadora

An *expression* that returns an *iterator*. It looks like a normal expression followed by a `for` clause defining a loop variable, range, and an optional `if` clause. The combined expression generates values for an enclosing function:

```
>>> 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](#).

tipos genéricos

Un *type* que se puede parametrizar; normalmente un container class como `list` o `dict`. Usado para *type hints* y *annotations*.

Para más detalles, véase generic alias types, [PEP 483](#), [PEP 484](#), [PEP 585](#), y el módulo `typing`.

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.

As of Python 3.13, the GIL can be disabled using the `--disable-gil` build configuration. After building Python with this option, code must be run with `-X gil=0` or after setting the `PYTHON_GIL=0` environment variable. This feature enables improved performance for multi-threaded applications and makes it easier to use multi-core CPUs efficiently. For more details, see [PEP 703](#).

In prior versions of Python's C API, a function might declare that it requires the GIL to be held in order to use it. This refers to having an *attached thread state*.

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

An object is *hashable* if it has a hash value which never changes during its lifetime (it needs a `__hash__()` method), and can be compared to other objects (it needs an `__eq__()` method). Hashable objects which compare equal must have the same hash value.

Ser *hashable* hace a un objeto utilizable como clave de un diccionario y miembro de un set, porque éstas estructuras de datos usan los valores de hash internamente.

La mayoría de los objetos inmutables incorporados en Python son *hashables*; los contenedores mutables (como las listas o los diccionarios) no lo son; los contenedores inmutables (como tuplas y conjuntos *frozensets*) son *hashables* si sus elementos son *hashables*. Los objetos que son instancias de clases definidas por el usuario son *hashables* por defecto. Todos se comparan como desiguales (excepto consigo mismos), y su valor de hash está derivado de su función `id()`.

IDLE

Un Entorno Integrado de Desarrollo y Aprendizaje para Python. idle es un editor básico y un entorno de intérprete que se incluye con la distribución estándar de Python.

immortal

Immortal objects are a CPython implementation detail introduced in [PEP 683](#).

If an object is immortal, its *reference count* is never modified, and therefore it is never deallocated while the interpreter is running. For example, `True` and `None` are immortal in CPython.

Immortal objects can be identified via `sys._is_immortal()`, or via `PyUnstable_IsImmortal()` in the C API.

immutable

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 has an interactive interpreter which means you can enter statements and expressions at the interpreter prompt, immediately execute them and see their results. Just launch `python` with no arguments (possibly by selecting it from your computer's main menu). It is a very powerful way to test out new ideas or inspect modules and packages (remember `help(x)`). For more on interactive mode, see `tut-interac`.

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 intérprete es que el módulo `__main__` o el script que estaba corriendo termine su ejecución.

iterable

An object capable of returning its members one at a time. Examples of iterables include all sequence types (such as `list`, `str`, and `tuple`) and some non-sequence types like `dict`, *file objects*, and objects of any classes you define with an `__iter__()` method or with a `__getitem__()` method that implements *sequence* semantics.

Iterables can be used in a `for` loop and in many other places where a sequence is needed (`zip()`, `map()`, ...). When an iterable object is passed as an argument to the built-in function `iter()`, it returns an iterator for the object. This iterator is good for one pass over the set of values. When using iterables, it is usually not necessary to call `iter()` or deal with iterator objects yourself. The `for` statement does that automatically for you, creating a temporary unnamed variable to hold the iterator for the duration of the loop. See also *iterator*, *sequence*, and *generator*.

iterador

An object representing a stream of data. Repeated calls to the iterator's `__next__()` method (or passing it to the built-in function `next()`) return successive items in the stream. When no more data are available a `StopIteration` exception is raised instead. At this point, the iterator object is exhausted and any further calls to its `__next__()` method just raise `StopIteration` again. Iterators are required to have an `__iter__()` method that returns the iterator object itself so every iterator is also iterable and may be used in most places where other iterables are accepted. One notable exception is code which attempts multiple iteration passes. A container object (such as a `list`) produces a fresh new iterator each time you pass it to the `iter()` function or use it in a `for` loop. Attempting this with an iterator will just return the same exhausted iterator object used in the previous iteration pass, making it appear like an empty container.

Puede encontrar más información en *typeiter*.

Detalles de implementación de CPython: CPython does not consistently apply the requirement that an iterator define `__iter__()`. And also please note that the free-threading CPython does not guarantee the thread-safety of iterator operations.

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 distinguen mayúsculas de minúsculas. Como alternativa, una función clave puede ser realizada con una expresión `lambda` como `lambda r: (r[0], r[2])`. Además, `operator.attrgetter()`, `operator.itemgetter()` y `operator.methodcaller()` son tres constructores de funciones clave. Consulte *Sorting HOW TO* para ver ejemplos de cómo crear y utilizar 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.

lexical analyzer

Formal name for the *tokenizer*; see [token](#).

lista

A built-in Python [sequence](#). Despite its name it is more akin to an array in other languages than to a linked list since access to elements is $O(1)$.

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

An object that loads a module. It must define the `exec_module()` and `create_module()` methods to implement the `Loader` interface. A loader is typically returned by a [finder](#). See also:

- `finders-and-loaders`
- `importlib.abc.Loader`
- **PEP 302**

codificación de la configuración regional

En Unix, es la codificación de la configuración regional `LC_CTYPE`. Se puede configurar con `locale.setlocale(locale.LC_CTYPE, new_locale)`.

En Windows, es la página de códigos ANSI (por ejemplo, "cp1252").

En Android y VxWorks, Python utiliza "utf-8" como codificación regional.

`locale.getencoding()` can be used to get the locale encoding.

Vea también [filesystem encoding and error handler](#).

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 `collections.abc.Mapping` o `collections.abc.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.

metaclass

La clase de una clase. Las definiciones de clases crean nombres de clase, un diccionario de clase, y una lista

de clases base. Las metaclasses 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 metaclasses a medida. La mayoría de los usuarios nunca necesitarán esta herramienta, pero cuando la necesidad surge, las metaclasses 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

Method Resolution Order is the order in which base classes are searched for a member during lookup. See `python_2.3_mro` for details of the algorithm used by the Python interpreter since the 2.3 release.

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`.

See also `module-specs`.

MRO

Vea *method resolution order*.

mutable

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

tupla nombrada

La denominación «tupla nombrada» se aplica a cualquier tipo o clase que hereda de una tupla y cuyos elementos indexables son también accesibles usando atributos nombrados. Este tipo o clase puede tener además otras capacidades.

Varios tipos incorporados son tuplas nombradas, incluyendo los valores retornados por `time.localtime()` y `os.stat()`. Otro ejemplo es `sys.float_info`:

```
>>> sys.float_info[1]                # indexed access
1024
>>> sys.float_info.max_exp            # named field access
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 by inheriting `typing.NamedTuple`, or with the factory function `collections.namedtuple()`. The latter techniques also add 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

A *package* which serves only as a container for subpackages. Namespace packages may have no physical representation, and specifically are not like a *regular package* because they have no `__init__.py` file.

Namespace packages allow several individually installable packages to have a common parent package. Otherwise, it is recommended to use a *regular package*.

For more information, see [PEP 420](#) and [reference-namespace-package](#).

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

Old name for the flavor of classes now used for all class objects. In earlier Python versions, only new-style classes could use Python's newer, versatile features like `__slots__`, descriptors, properties, `__getattr__()`, class methods, and static methods.

objeto

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

optimized scope

A scope where target local variable names are reliably known to the compiler when the code is compiled, allowing optimization of read and write access to these names. The local namespaces for functions, generators, coroutines, comprehensions, and generator expressions are optimized in this fashion. Note: most interpreter optimizations are applied to all scopes, only those relying on a known set of local and nonlocal variable names are restricted to optimized scopes.

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. Los parámetros sólo posicionales pueden ser definidos incluyendo un carácter `/` en la lista de parámetros de la función después de ellos, como *posonly1* y *posonly2* en el ejemplo que sigue:

```
def func(posonly1, posonly2, /, positional_or_keyword): ...
```

- *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 simple `*` 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 precedente.

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 ruta implementan.

gancho a entrada de ruta

A callable on the `sys.path_hooks` list which returns a *path entry finder* if it knows how to find modules on a specific *path entry*.

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()` o `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 provisional

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 provisionarias, 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 411](#) 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

The number of references to an object. When the reference count of an object drops to zero, it is deallocated. Some objects are *immortal* and have reference counts that are never modified, and therefore the objects are never deallocated. Reference counting is generally not visible to Python code, but it is a key element of the *CPython* implementation. Programmers can call the `sys.getrefcount()` function to return the reference count for a particular object.

In *CPython*, reference counts are not considered to be stable or well-defined values; the number of references to an object, and how that number is affected by Python code, may be different between versions.

paquete regular

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

Vea también *namespace package*.

REPL

An acronym for the «read–eval–print loop», another name for the *interactive* interpreter shell.

`__slots__`

Es una declaración dentro de una clase que ahorra memoria predeclarando 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

An *iterable* which supports efficient element access using integer indices via the `__getitem__()` special method and defines a `__len__()` method that returns the length of the sequence. Some built-in sequence types are `list`, `str`, `tuple`, and `bytes`. Note that `dict` also supports `__getitem__()` and `__len__()`, but is considered a mapping rather than a sequence because the lookups use arbitrary *hashable* keys rather than integers.

The `collections.abc.Sequence` abstract base class defines a much richer interface that goes beyond just `__getitem__()` and `__len__()`, adding `count()`, `index()`, `__contains__()`, and `__reversed__()`. Types that implement this expanded interface can be registered explicitly using `register()`. For more documentation on sequence methods generally, see Common Sequence Operations.

comprensión de conjuntos

Una forma compacta de procesar todos o parte de los elementos en un iterable y retornar un conjunto con los resultados. `results = {c for c in 'abracadabra' if c not in 'abc'}` genera el conjunto de cadenas `{'r', 'd'}`. Ver comprehensions.

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*.

soft deprecated

A soft deprecated API should not be used in new code, but it is safe for already existing code to use it. The API remains documented and tested, but will not be enhanced further.

Soft deprecation, unlike normal deprecation, does not plan on removing the API and will not emit warnings.

See [PEP 387: Soft Deprecation](#).

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`.

standard library

The collection of *packages*, *modules* and *extension modules* distributed as a part of the official Python interpreter package. The exact membership of the collection may vary based on platform, available system libraries, or other criteria. Documentation can be found at `library-index`.

See also `sys.stdlib_module_names` for a list of all possible standard library module names.

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`.

static type checker

An external tool that reads Python code and analyzes it, looking for issues such as incorrect types. See also [type hints](#) and the `typing` module.

stdlib

An abbreviation of [standard library](#).

referencia fuerte

En la API de C de Python, una referencia fuerte es una referencia a un objeto que es propiedad del código que mantiene la referencia. La referencia fuerte se toma llamando a `Py_INCREF()` cuando se crea la referencia y se libera con `Py_DECREF()` cuando se elimina la referencia.

La función `Py_NewRef()` se puede utilizar para crear una referencia fuerte a un objeto. Por lo general, se debe llamar a la función `Py_DECREF()` en la referencia fuerte antes de salir del alcance de la referencia fuerte, para evitar filtrar una referencia.

Consulte también [borrowed reference](#).

t-string**t-strings**

String literals prefixed with `t` or `T` are commonly called «t-strings» which is short for template string literals.

codificación de texto

Una cadena de caracteres en Python es una secuencia de puntos de código Unicode (en el rango `U+0000–U+10FFFF`). Para almacenar o transferir una cadena de caracteres, es necesario serializarla como una secuencia de bytes.

La serialización de una cadena de caracteres en una secuencia de bytes se conoce como «codificación», y la recreación de la cadena de caracteres a partir de la secuencia de bytes se conoce como «decodificación».

Existe una gran variedad de serializaciones de texto codecs, que se denominan colectivamente «codificaciones de texto».

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](#).

thread state

The information used by the [CPython](#) runtime to run in an OS thread. For example, this includes the current exception, if any, and the state of the bytecode interpreter.

Each thread state is bound to a single OS thread, but threads may have many thread states available. At most, one of them may be [attached](#) at once.

An [attached thread state](#) is required to call most of Python's C API, unless a function explicitly documents otherwise. The bytecode interpreter only runs under an attached thread state.

Each thread state belongs to a single interpreter, but each interpreter may have many thread states, including multiple for the same OS thread. Thread states from multiple interpreters may be bound to the same thread, but only one can be [attached](#) in that thread at any given moment.

See Thread State and the Global Interpreter Lock for more information.

token

A small unit of source code, generated by the lexical analyzer (also called the *tokenizer*). Names, numbers, strings, operators, newlines and similar are represented by tokens.

The `tokenize` module exposes Python's lexical analyzer. The `token` module contains information on the various types of tokens.

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

The type of a Python object determines what kind of object it is; every object has a type. An object's type is accessible as its `__class__` attribute or can be retrieved with `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:

```
def remove_gray_shades(  
    colors: list[tuple[int, int, int]]) -> list[tuple[int, int, int]]:  
    pass
```

podría ser más legible así:

```
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.

Type hints are optional and are not enforced by Python but they are useful to *static type checkers*. They can also aid IDEs with code completion and refactoring.

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 `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`.

Consulte *function annotation*, **PEP 484** y **PEP 526**, que describen esta funcionalidad. Consulte también *annotations-howto* para conocer las mejores prácticas sobre cómo trabajar con anotaciones.

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*.

walrus operator

A light-hearted way to refer to the assignment expression operator `:` because it looks a bit like a walrus if you turn your head.

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.

About this documentation

Python's documentation is generated from [reStructuredText](#) sources using [Sphinx](#), a documentation generator originally created for Python and now maintained as an independent project.

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., the creator of the original Python documentation toolset and author of much of the content;
- el proyecto [Docutils](#) para creación de reStructuredText y la suite Docutils;
- Fredrik Lundh por su proyecto Referencia Alternativa de Python del que Sphinx obtuvo muchas buenas ideas.

B.1 Contributors to the Python documentation

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!

Historia y Licencia

C.1 Historia del 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, which became Zope Corporation. 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 was 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.

Lanzamiento	Derivado de	Año	Dueño/a	GPL-compatible? (1)
0.9.0 hasta 1.2	n/a	1991-1995	CWI	sí
1.3 hasta 1.5.2	1.2	1995-1999	CNRI	sí
1.6	1.5.2	2000	CNRI	no
2.0	1.6	2000	BeOpen.com	no
1.6.1	1.6	2001	CNRI	yes (2)
2.1	2.0+1.6.1	2001	PSF	no
2.0.1	2.0+1.6.1	2001	PSF	sí
2.1.1	2.1+2.0.1	2001	PSF	sí
2.1.2	2.1.1	2002	PSF	sí
2.1.3	2.1.2	2002	PSF	sí
2.2 y superior	2.1.1	2001-ahora	PSF	sí

i Nota

- (1) 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.

- (2) According to Richard Stallman, 1.6.1 is not GPL-compatible, because its license has a choice of law clause. According to CNRI, however, Stallman's lawyer has told CNRI's lawyer that 1.6.1 is «not incompatible» with the GPL.

Gracias a los muchos voluntarios externos que han trabajado bajo la dirección de Guido para hacer posibles estos lanzamientos.

C.2 Términos y condiciones para acceder o usar Python

Python software and documentation are licensed under the Python Software Foundation License Version 2.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Version 2 and the *Zero-Clause BSD license*.

Parte del software incorporado en Python está bajo diferentes licencias. Las licencias se enumeran con el código correspondiente a esa licencia. Consulte *Licencias y reconocimientos para software incorporado* para obtener una lista incompleta de estas licencias.

C.2.1 PYTHON SOFTWARE FOUNDATION LICENSE VERSION 2

1. This LICENSE AGREEMENT is between the Python Software Foundation ("PSF"), and the Individual or Organization ("Licensee") accessing and otherwise using this software ("Python") 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 alone or in any derivative version, provided, however, that PSF's License Agreement and PSF's notice of copyright, i.e., "Copyright © 2001 Python Software Foundation; All Rights Reserved" are retained in Python 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 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.
4. PSF is making Python 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 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON, 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

(continúe en la próxima página)

(proviene de la página anterior)

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, Licensee agrees to be bound by the terms and conditions of this License Agreement.

C.2.2 ACUERDO DE LICENCIA DE BEOPEN.COM PARA PYTHON 2.0

ACUERDO DE LICENCIA DE CÓDIGO ABIERTO DE BEOPEN PYTHON VERSIÓN 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 ACUERDO DE LICENCIA CNRI PARA 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.

(continúe en la próxima página)

(proviene de la página anterior)

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 ACUERDO DE LICENCIA CWI PARA PYTHON 0.9.0 HASTA 1.2

Copyright © 1991 - 1995, Stichting Mathematisch Centrum Amsterdam, The Netherlands. 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 Stichting Mathematisch Centrum or CWI not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

STICHTING MATHEMATISCH CENTRUM DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE, INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS, IN NO EVENT SHALL STICHTING MATHEMATISCH 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.

C.2.5 ZERO-CLAUSE BSD LICENSE FOR CODE IN THE PYTHON DOCUMENTATION

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, 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 Licencias y reconocimientos para software incorporado

Esta sección es una lista incompleta, pero creciente, de licencias y reconocimientos para software de terceros incorporado en la distribución de Python.

C.3.1 Mersenne Twister

La extensión `C_random` subyacente al módulo `random` incluye código basado en una descarga de <http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/MT2002/emt19937ar.html>. Los siguientes son los comentarios textuales del código original:

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)`

(continúe en la próxima página)

(proviene de la página anterior)

```
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
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

El módulo `socket` usa las funciones, `getaddrinfo()`, y `getnameinfo()`, que están codificadas en archivos fuente separados del Proyecto WIDE, <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.

(continúe en la próxima página)

(proviene de la página anterior)

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 Servicios de socket asincrónicos

Los módulos `test.support.asyncchat` y `test.support.asyncore` contienen el siguiente aviso:

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 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 Gestión de cookies

El módulo `http.cookies` contiene el siguiente aviso:

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

(continúe en la próxima página)

(proviene de la página anterior)

```
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 Seguimiento de ejecución

El módulo `trace` contiene el siguiente aviso:

```
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.  
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 funciones `UUencode` y `UUdecode`

The `uu` codec 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
```

(continúe en la próxima página)

(proviene de la página anterior)

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 Llamadas a procedimientos remotos XML

El módulo `xmlrpc.client` contiene el siguiente aviso:

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 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 MERCHANTABILITY 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

El módulo `test.test_epoll` contiene el siguiente aviso:

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:

(continúe 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.3.9 Seleccionar kqueue

El módulo `select` contiene el siguiente aviso para la interfaz `kqueue`:

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 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

El archivo `Python/pyhash.c` contiene la implementación de Marek Majkowski del algoritmo SipHash24 de Dan Bernstein. Contiene la siguiente nota:

<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.

(continúe en la próxima página)

(proviene de la página anterior)

</MIT License>

Original location:

<https://github.com/majek/csiphash/>

Solution inspired by code from:

Samuel Neves ([supercooper/crypto_auth/siphash24/little](https://github.com/supercooper/crypto_auth/siphash24/little))djb ([supercooper/crypto_auth/siphash24/little2](https://github.com/supercooper/crypto_auth/siphash24/little2))Jean-Philippe Aumasson (<https://131002.net/siphash/siphash24.c>)

C.3.11 strtod y dtoa

El archivo `Python/dtoa.c`, que proporciona las funciones de C `dtoa` y `strtod` para la conversión de dobles C hacia y desde cadenas de caracteres, se deriva del archivo del mismo nombre por David M. Gay, actualmente disponible en <https://web.archive.org/web/20220517033456/http://www.netlib.org/fp/dtoa.c>. El archivo original, recuperado el 16 de marzo de 2009, contiene el siguiente aviso de licencia y derechos de autor:

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

C.3.12 OpenSSL

The modules `hashlib`, `posix` and `ssl` use the OpenSSL library for added performance if made available by the operating system. Additionally, the Windows and macOS installers for Python may include a copy of the OpenSSL libraries, so we include a copy of the OpenSSL license here. For the OpenSSL 3.0 release, and later releases derived from that, the Apache License v2 applies:

```
Apache License
Version 2.0, January 2004
https://www.apache.org/licenses/
```

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

(continúe en la próxima página)

(proviene de la página anterior)

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of,

(continúe en la próxima página)

(proviene de la página anterior)

publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.
4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
 - (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and
 - (b) You must cause any modified files to carry prominent notices stating that You changed the files; and
 - (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
 - (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or

(continúe en la próxima página)

(proviene de la página anterior)

for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

C.3.13 expat

La extensión `pyexpat` se construye usando una copia incluida de las fuentes de expatriados a menos que la construcción esté configurada `--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

La extensión `C_ctypes` subyacente al módulo `ctypes` se construye usando una copia incluida de las fuentes de `libffi` a menos que la construcción esté configurada `--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:

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

La extensión `zlib` se crea utilizando una copia incluida de las fuentes de `zlib` si la versión de `zlib` encontrada en el sistema es demasiado antigua para ser utilizada para la compilación:

```
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      jloup@gzip.org
```

```
Mark Adler            madler@alumni.caltech.edu
```

C.3.16 cfuhash

La implementación de la tabla hash utilizada por `tracemalloc` se basa en el proyecto `cfuhash`:

```
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.
- * 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,
```

(continúe en la próxima página)

(proviene de la página anterior)

```
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

La extensión `C_decimal` subyacente al módulo `decimal` se construye usando una copia incluida de la biblioteca `libmpdec` a menos que la construcción esté configurada `--with-system-libmpdec`:

```
Copyright (c) 2008-2020 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.

C.3.18 Conjunto de pruebas W3C C14N

El conjunto de pruebas C14N 2.0 en el paquete `test` (`Lib/test/xmltestdata/c14n-20/`) se recuperó del sitio web de W3C en <https://www.w3.org/TR/xml-c14n2-testcases/> y se distribuye bajo la licencia BSD de 3 cláusulas:

```
Copyright (c) 2013 W3C(R) (MIT, ERCIM, Keio, Beihang),
All Rights Reserved.
```

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

- * Redistributions of works must retain the original copyright notice, this list of conditions and the following disclaimer.
- * Redistributions in binary form must reproduce the original 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 W3C nor the names of its contributors may be

(continúe en la próxima página)

(proviene de la página anterior)

used to endorse or promote products derived from this work 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.3.19 mimalloc

MIT License:

Copyright (c) 2018–2021 Microsoft Corporation, Daan Leijen

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.3.20 asyncio

Parts of the `asyncio` module are incorporated from [uvloop 0.16](#), which is distributed under the MIT license:

Copyright (c) 2015–2021 MagicStack Inc. <http://magic.io>

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.

(continúe en la próxima página)

(proviene de la página anterior)

```
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.3.21 Global Unbounded Sequences (GUS)

The file `Python/qsbr.c` is adapted from FreeBSD's «Global Unbounded Sequences» safe memory reclamation scheme in `subr_smr.c`. The file is distributed under the 2-Clause BSD License:

```
Copyright (c) 2019,2020 Jeffrey Roberson <jeff@FreeBSD.org>
```

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 unmodified, 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 "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 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.3.22 Zstandard bindings

Zstandard bindings in `Modules/_zstd` and `Lib/compression/zstd` are based on code from the [pyzstd library](#), copyright Ma Lin and contributors. The `pyzstd` code is distributed under the 3-Clause BSD License:

```
Copyright (c) 2020-present, Ma Lin and contributors.
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 copyright holder nor the names of its contributors may be used to endorse or promote products derived from

(continúe en la próxima página)

(proviene de la página anterior)

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 HOLDER 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

Derechos de autor

Python y esta documentación es:

Copyright © 2001 Python Software Foundation. All rights reserved.

Derechos de autor © 2000 BeOpen.com. Todos los derechos reservados.

Derechos de autor © 1995-2000 Corporation for National Research Initiatives. Todos los derechos reservados.

Derechos de autor © 1991-1995 Stichting Mathematisch Centrum. Todos los derechos reservados.

Consulte [Historia](#) y [Licencia](#) para obtener información completa sobre licencias y permisos.

No alfabético

..., [97](#)

-?

opción de línea de comando, [5](#)

>>>, [97](#)

__future__, [104](#)

__slots__, [112](#)

A

a la espera, [99](#)

administrador asincrónico de contexto, [98](#)

administrador de contextos, [100](#)

alcances anidados, [109](#)

alias de tipos, [114](#)

annotate function, [97](#)

anotación, [97](#)

anotación de función, [103](#)

anotación de variable, [114](#)

apagado del intérprete, [106](#)

API provisional, [110](#)

archivo binario, [99](#)

archivo de texto, [113](#)

argumento, [97](#)

argumento nombrado, [107](#)

argumento posicional, [110](#)

atributo, [99](#)

attached thread state, [98](#)

B

-b

opción de línea de comando, [6](#)

-B

opción de línea de comando, [6](#)

BDFL, [99](#)

bloqueo global del intérprete, [104](#)

BOLT_APPLY_FLAGS

opción de línea de comando, [33](#)

BOLT_INSTRUMENT_FLAGS

opción de línea de comando, [33](#)

--build

opción de línea de comando, [39](#)

buscador, [103](#)

buscador basado en ruta, [110](#)

buscador de entradas de ruta, [110](#)

bytecode, [99](#)

BZIP2_CFLAGS

opción de línea de comando, [30](#)

BZIP2_LIBS

opción de línea de comando, [30](#)

C

-c

opción de línea de comando, [4](#)

cadena con triple comilla, [113](#)

callable, [99](#)

cargador, [107](#)

CC

opción de línea de comando, [29](#)

C-contiguous, [101](#)

CFLAGS, [32](#), [4244](#)

opción de línea de comando, [29](#)

CFLAGS_NODIST, [4244](#)

--check-hash-based-pycs

opción de línea de comando, [6](#)

clase, [100](#)

clase base abstracta, [97](#)

clase de nuevo estilo, [109](#)

closure variable, [100](#)

codificación de la configuración

regional, [107](#)

codificación de texto, [113](#)

codificación del sistema de archivos y

manejador de errores, [103](#)

comprensión de conjuntos, [112](#)

comprensión de diccionarios, [102](#)

comprensión de listas, [107](#)

CONFIG_SITE

opción de línea de comando, [39](#)

contador de referencias, [111](#)

context, [100](#)

context management protocol, [100](#)

contiguo, [101](#)

corrutina, [101](#)

CPP

opción de línea de comando, [29](#)

CPPFLAGS, [42](#), [44](#)

opción de línea de comando, [29](#)

CPython, [101](#)

current context, [101](#)
CURSES_CFLAGS
 opción de línea de comando, [30](#)
CURSES_LIBS
 opción de línea de comando, [30](#)
cyclic isolate, [101](#)

D

-d
 opción de línea de comando, [6](#)
decorador, [101](#)
descriptor, [101](#)
despacho único, [112](#)
diccionario, [102](#)
--disable-gil
 opción de línea de comando, [29](#)
--disable-ipv6
 opción de línea de comando, [26](#)
--disable-safety
 opción de línea de comando, [37](#)
--disable-test-modules
 opción de línea de comando, [32](#)
división entera a la baja, [103](#)
docstring, [102](#)
dunder, [102](#)

E

-E
 opción de línea de comando, [6](#)
EAFP, [102](#)
--enable-big-digits
 opción de línea de comando, [26](#)
--enable-bolt
 opción de línea de comando, [33](#)
--enable-experimental-jit
 opción de línea de comando, [29](#)
--enable-framework
 opción de línea de comando, [38](#), [39](#)
--enable-loadable-sqlite-extensions
 opción de línea de comando, [26](#)
--enable-optimizations
 opción de línea de comando, [32](#)
--enable-profiling
 opción de línea de comando, [33](#)
--enable-pystats
 opción de línea de comando, [27](#)
--enable-shared
 opción de línea de comando, [36](#)
--enable-slower-safety
 opción de línea de comando, [38](#)
--enable-universalsdk
 opción de línea de comando, [38](#)
--enable-wasm-dynamic-linking
 opción de línea de comando, [31](#)
--enable-wasm-pthreads
 opción de línea de comando, [31](#)
entorno virtual, [114](#)
entrada de ruta, [110](#)

espacio de nombres, [108](#)
especificador de módulo, [108](#)
evaluate function, [102](#)
--exec-prefix
 opción de línea de comando, [31](#)
expresión, [102](#)
expresión generadora, [104](#)

F

f-string, [102](#)
f-strings, [102](#)
Fortran contiguous, [101](#)
free threading, [103](#)
free variable, [103](#)
función, [103](#)
función clave, [106](#)
función corrutina, [101](#)
función genérica, [104](#)

G

gancho a entrada de ruta, [110](#)
GDBM_CFLAGS
 opción de línea de comando, [30](#)
GDBM_LIBS
 opción de línea de comando, [30](#)
generador, [104](#)
generador asincrónico, [98](#)
GIL, [104](#)

H

-h
 opción de línea de comando, [5](#)
hash-based pyc, [105](#)
hashable, [105](#)
--help
 opción de línea de comando, [5](#)
--help-all
 opción de línea de comando, [5](#)
--help-env
 opción de línea de comando, [5](#)
--help-xoptions
 opción de línea de comando, [5](#)
--host
 opción de línea de comando, [39](#)
HOSTRUNNER
 opción de línea de comando, [39](#)

I

-i
 opción de línea de comando, [6](#)
-I
 opción de línea de comando, [7](#)
IDLE, [105](#)
immortal, [105](#)
importador, [105](#)
importar, [105](#)
indicador de tipo, [114](#)
immutable, [105](#)

interactivo, **105**
 interpretado, **106**
 iterable, **106**
 iterable asincrónico, **98**
 iterador, **106**
 iterador asincrónico, **98**
 iterador generador, **104**
 iterador generador asincrónico, **98**

L

lambda, **107**
 LBYL, **107**
 LDFLAGS, **42, 44**
 opción de línea de comando, **30**
 LDFLAGS_NODIST, **44**
 lexical analyzer, **107**
 LIBB2_CFLAGS
 opción de línea de comando, **30**
 LIBB2_LIBS
 opción de línea de comando, **30**
 LIBEDIT_CFLAGS
 opción de línea de comando, **30**
 LIBEDIT_LIBS
 opción de línea de comando, **30**
 LIBFFI_CFLAGS
 opción de línea de comando, **30**
 LIBFFI_LIBS
 opción de línea de comando, **30**
 LIBLZMA_CFLAGS
 opción de línea de comando, **30**
 LIBLZMA_LIBS
 opción de línea de comando, **30**
 LIBMPDEC_CFLAGS
 opción de línea de comando, **30**
 LIBMPDEC_LIBS
 opción de línea de comando, **30**
 LIBREADLINE_CFLAGS
 opción de línea de comando, **30**
 LIBREADLINE_LIBS
 opción de línea de comando, **30**
 LIBS
 opción de línea de comando, **30**
 LIBSQLITE3_CFLAGS
 opción de línea de comando, **31**
 LIBSQLITE3_LIBS
 opción de línea de comando, **31**
 LIBUUID_CFLAGS
 opción de línea de comando, **31**
 LIBUUID_LIBS
 opción de línea de comando, **31**
 LIBZSTD_CFLAGS
 opción de línea de comando, **31**
 LIBZSTD_LIBS
 opción de línea de comando, **31**
 lista, **107**

M

-m

 opción de línea de comando, **4**
 MACHDEP
 opción de línea de comando, **30**
 magic
 método, **107**
 mapeado, **107**
 máquina virtual, **115**
 meta buscadores de ruta, **107**
 metacalse, **107**
 método, **108**
 magic, **107**
 special, **112**
 método especial, **112**
 método mágico, **107**
 módulo, **108**
 módulo de extensión, **102**
 MRO, **108**
 mutable, **108**

N

nombre calificado, **111**
 número complejo, **100**

O

-O
 opción de línea de comando, **7**
 objeto, **109**
 objeto archivo, **102**
 objeto tipo ruta, **110**
 objetos tipo archivo, **103**
 objetos tipo binarios, **99**
 -OO
 opción de línea de comando, **7**
 opción de línea de comando
 -?, **5**
 -b, **6**
 -B, **6**
 BOLT_APPLY_FLAGS, **33**
 BOLT_INSTRUMENT_FLAGS, **33**
 --build, **39**
 BZIP2_CFLAGS, **30**
 BZIP2_LIBS, **30**
 -c, **4**
 CC, **29**
 CFLAGS, **29**
 --check-hash-based-pycs, **6**
 CONFIG_SITE, **39**
 CPP, **29**
 CPPFLAGS, **29**
 CURSES_CFLAGS, **30**
 CURSES_LIBS, **30**
 -d, **6**
 --disable-gil, **29**
 --disable-ipv6, **26**
 --disable-safety, **37**
 --disable-test-modules, **32**
 -E, **6**
 --enable-big-digits, **26**

```
--enable-bolt, 33
--enable-experimental-jit, 29
--enable-framework, 38, 39
--enable-loadable-sqlite-extensions,
    26
--enable-optimizations, 32
--enable-profiling, 33
--enable-pystats, 27
--enable-shared, 36
--enable-slower-safety, 38
--enable-universalsdk, 38
--enable-wasm-dynamic-linking, 31
--enable-wasm-pthreads, 31
--exec-prefix, 31
GDBM_CFLAGS, 30
GDBM_LIBS, 30
-h, 5
--help, 5
--help-all, 5
--help-env, 5
--help-xoptions, 5
--host, 39
HOSTRUNNER, 39
-i, 6
-I, 7
LDFLAGS, 30
LIBB2_CFLAGS, 30
LIBB2_LIBS, 30
LIBEDIT_CFLAGS, 30
LIBEDIT_LIBS, 30
LIBFFI_CFLAGS, 30
LIBFFI_LIBS, 30
LIBLZMA_CFLAGS, 30
LIBLZMA_LIBS, 30
LIBMPDEC_CFLAGS, 30
LIBMPDEC_LIBS, 30
LIBREADLINE_CFLAGS, 30
LIBREADLINE_LIBS, 30
LIBS, 30
LIBSQLITE3_CFLAGS, 31
LIBSQLITE3_LIBS, 31
LIBUUID_CFLAGS, 31
LIBUUID_LIBS, 31
LIBZSTD_CFLAGS, 31
LIBZSTD_LIBS, 31
-m, 4
MACHDEP, 30
-O, 7
-OO, 7
-P, 7
PANEL_CFLAGS, 31
PANEL_LIBS, 31
PKG_CONFIG, 29
PKG_CONFIG_LIBDIR, 29
PKG_CONFIG_PATH, 29
--prefix, 31
-q, 7
-R, 7
-s, 7
-S, 8
TCLTK_CFLAGS, 31
TCLTK_LIBS, 31
-u, 8
-v, 8
-V, 6
--version, 6
-W, 8
--with-address-sanitizer, 35
--with-app-store-compliance, 38
--with-assertions, 35
--with-build-python, 39
--with-builtin-hashlib-hashes, 37
--with-computed-gotos, 33
--with-dbmlliborder, 27
--with-dtrace, 35
--with-ensurepip, 32
--with-framework-name, 38, 39
--with-hash-algorithm, 37
--with-libc, 36
--with-libm, 36
--with-libs, 36
--with-lto, 32
--with-memory-sanitizer, 35
--with-openssl, 36
--with-openssl-rpath, 37
--without-c-locale-coercion, 27
--without-decimal-contextvar, 27
--without-doc-strings, 33
--without-mimalloc, 33
--without-pymalloc, 33
--without-readline, 36
--without-remote-debug, 34
--without-static-libpython, 36
--with-pkg-config, 27
--with-platlibdir, 27
--with-pydebug, 35
--with-readline, 36
--with-ssl-default-suites, 37
--with-strict-overflow, 34
--with-suffix, 26
--with-system-expat, 36
--with-system-libmpdec, 36
--with-tail-call-interp, 33
--with-thread-sanitizer, 35
--with-trace-refs, 35
--with-tzpath, 27
--with-undefined-behavior-sanitizer,
    35
--with-universal-archs, 38
--with-valgrind, 35
--with-wheel-pkg-dir, 27
-x, 9
-X, 9
ZLIB_CFLAGS, 31
ZLIB_LIBS, 31
OPT, 35
```

optimized scope, [109](#)

orden de resolución de métodos, [108](#)

P

-P

opción de línea de comando, [7](#)

PANEL_CFLAGS

opción de línea de comando, [31](#)

PANEL_LIBS

opción de línea de comando, [31](#)

paquete, [109](#)

paquete de espacios de nombres, [109](#)

paquete provisorio, [111](#)

paquete regular, [112](#)

parámetro, [109](#)

PATH, [12](#), [23](#), [48](#), [52](#), [53](#), [55](#), [56](#), [6264](#), [67](#), [69](#)

PATHEXT, [64](#)

PEP, [110](#)

PKG_CONFIG

opción de línea de comando, [29](#)

PKG_CONFIG_LIBDIR

opción de línea de comando, [29](#)

PKG_CONFIG_PATH

opción de línea de comando, [29](#)

porción, [110](#)

--prefix

opción de línea de comando, [31](#)

PROFILE_TASK, [32](#)

PY_PYTHON, [70](#)

Py_REMOTE_DEBUG (*C macro*), [34](#)

PYLAUNCHER_ALLOW_INSTALL, [71](#)

PYLAUNCHER_ALWAYS_INSTALL, [71](#)

PYLAUNCHER_DEBUG, [71](#)

PYLAUNCHER_DRYRUN, [71](#)

PYLAUNCHER_NO_SEARCH_PATH, [69](#)

Python 3000, [111](#)

Python Enhancement Proposals

PEP 1, [110](#)

PEP 7, [25](#)

PEP 8, [95](#)

PEP 11, [25](#), [59](#)

PEP 238, [103](#)

PEP 278, [114](#)

PEP 302, [107](#)

PEP 338, [4](#)

PEP 343, [100](#)

PEP 362, [98](#), [110](#)

PEP 370, [8](#), [14](#)

PEP 397, [67](#)

PEP 411, [111](#)

PEP 420, [109](#), [110](#)

PEP 443, [104](#)

PEP 483, [104](#)

PEP 484, [97](#), [104](#), [114](#)

PEP 488, [7](#)

PEP 492, [98](#), [99](#), [101](#)

PEP 498, [102](#)

PEP 514, [67](#)

PEP 519, [110](#)

PEP 525, [98](#)

PEP 526, [97](#), [114](#)

PEP 528, [60](#)

PEP 529, [15](#), [60](#)

PEP 538, [16](#), [27](#)

PEP 585, [104](#)

PEP 649, [97](#)

PEP 683, [105](#)

PEP 703, [66](#), [82](#), [103](#), [105](#)

PEP 768, [10](#), [17](#), [34](#)

PEP 3116, [114](#)

PEP 3155, [111](#)

PYTHON_COLORS, [11](#)

PYTHON_CONTEXT_AWARE_WARNINGS, [11](#)

PYTHON_CPU_COUNT, [10](#)

PYTHON_DISABLE_REMOTE_DEBUG, [10](#)

PYTHON_FROZEN_MODULES, [10](#)

PYTHON_GIL, [11](#), [105](#)

PYTHON_JIT, [29](#)

PYTHON_MANAGER_DEFAULT, [48](#)

PYTHON_PERF_JIT_SUPPORT, [10](#)

PYTHON_PRESITE, [10](#)

PYTHON_THREAD_INHERIT_CONTEXT, [11](#)

PYTHON_TLBC, [11](#)

PYTHONCOERCECLOCALE, [27](#)

PYTHONDEBUG, [6](#), [34](#)

PYTHONDEVMODE, [9](#)

PYTHONDONTWRITEBYTECODE, [6](#)

PYTHONDUMPREFS, [35](#)

PYTHONFAULTHANDLER, [9](#)

PYTHONHASHSEED, [7](#), [13](#)

PYTHONHOME, [6](#), [11](#), [12](#), [60](#), [93](#)

Pythónico, [111](#)

PYTHONINSPECT, [7](#)

PYTHONINTMAXSTRDIGITS, [9](#)

PYTHONIOENCODING, [16](#)

PYTHONLEGACYWINDOWSSTDIO, [13](#)

PYTHONMALLOC, [15](#), [33](#)

PYTHONNODEBUGRANGES, [10](#)

PYTHONNOUSERSITE, [8](#)

PYTHONOPTIMIZE, [7](#)

PYTHONPATH, [6](#), [12](#), [60](#), [93](#), [94](#)

PYTHONPERFSUPPORT, [10](#)

PYTHONPROFILEIMPORTTIME, [9](#)

PYTHONPYCACHEPREFIX, [10](#)

PYTHONSAFEPATH, [7](#)

PYTHONSTARTUP, [7](#), [13](#)

PYTHONTRACEMALLOC, [9](#)

PYTHONUNBUFFERED, [8](#)

PYTHONUTF8, [10](#), [16](#), [59](#)

PYTHONVERBOSE, [8](#)

PYTHONWARNDEFAULTENCODING, [10](#)

PYTHONWARNINGS, [9](#)

Q

-q

opción de línea de comando, [7](#)

R

-R
 opción de línea de comando, 7
rebanada, 112
recolección de basura, 104
referencia fuerte, 113
referencia prestada, 99
REPL, 112
retrollamada, 100
ruta de importación, 105

S

-s
 opción de línea de comando, 7
-S
 opción de línea de comando, 8
saltos de líneas universales, 114
secuencia, 112
sentencia, 112
soft deprecated, 112
special
 método, 112
standard library, 112
static type checker, 113
stdlib, 113

T

t-string, 113
t-strings, 113
TCLTK_CFLAGS
 opción de línea de comando, 31
TCLTK_LIBS
 opción de línea de comando, 31
thread state, 113
tipado de pato, 102
tipo, 114
tipos genéricos, 104
token, 113
tupla nombrada, 108

U

-u
 opción de línea de comando, 8

V

-v
 opción de línea de comando, 8
-V
 opción de línea de comando, 6
variable de clase, 100
variable de contexto, 100
variables de entorno
 BASECFLAGS, 43
 BASECPPFLAGS, 42
 BLDSHARED, 44
 CC, 42
 CCSHARED, 43

CFLAGS, 32, 4244
CFLAGS_ALIASING, 43
CFLAGS_NODIST, 4244
CFLAGSFORSHARED, 43
COMPILEALL_OPTS, 43
CONFIGURE_CFLAGS, 43
CONFIGURE_CFLAGS_NODIST, 43
CONFIGURE_CPPFLAGS, 42
CONFIGURE_LDFLAGS, 44
CONFIGURE_LDFLAGS_NODIST, 44
CPPFLAGS, 42, 44
CXX, 42
EXTRA_CFLAGS, 43
LDFLAGS, 42, 44
LDFLAGS_NODIST, 44
LDSSHARED, 44
LIBS, 44
LINKCC, 44
OPT, 35, 43
PATH, 12, 23, 48, 52, 53, 55, 56, 6264, 67, 69
PATHEXT, 64
PROFILE_TASK, 32
PURIFY, 44
PY_BUILTIN_MODULE_CFLAGS, 44
PY_CFLAGS, 43
PY_CFLAGS_NODIST, 43
PY_CORE_CFLAGS, 43
PY_CORE_LDFLAGS, 45
PY_CPPFLAGS, 42
PY_LDFLAGS, 45
PY_LDFLAGS_NODIST, 45
PY_PYTHON, 70
PY_STDMODULE_CFLAGS, 43
PYLAUNCHER_ALLOW_INSTALL, 71
PYLAUNCHER_ALWAYS_INSTALL, 71
PYLAUNCHER_DEBUG, 71
PYLAUNCHER_DRYRUN, 71
PYLAUNCHER_NO_SEARCH_PATH, 69
PYTHON_BASIC_REPL, 17
PYTHON_COLORS, 11, 17
PYTHON_CONTEXT_AWARE_WARNINGS, 11, 18
PYTHON_CPU_COUNT, 10, 17
PYTHON_DISABLE_REMOTE_DEBUG, 10, 17
PYTHON_FROZEN_MODULES, 10, 17
PYTHON_GIL, 11, 18, 105
PYTHON_HISTORY, 17
PYTHON_JIT, 18, 29
PYTHON_MANAGER_DEFAULT, 48
PYTHON_PERF_JIT_SUPPORT, 10, 17
PYTHON_PRESITE, 10, 18
PYTHON_THREAD_INHERIT_CONTEXT, 11, 18
PYTHON_TLBC, 11, 18
PYTHONASYNCIODEBUG, 14
PYTHONBREAKPOINT, 12
PYTHONCASEOK, 13
PYTHONCOERCECLOCALE, 15, 27
PYTHONDEBUG, 6, 12, 34
PYTHONDEVMODE, 9, 16

- PYTHONDONTWRITEBYTECODE, 6, 13
- PYTHONDUMPPREFS, 18, 35
- PYTHONDUMPPREFSFILE, 18
- PYTHONEXECUTABLE, 14
- PYTHONFAULTHANDLER, 9, 14
- PYTHONHASHSEED, 7, 13
- PYTHONHOME, 6, 11, 12, 60, 93
- PYTHONINSPECT, 7, 12
- PYTHONINTMAXSTRDIGITS, 9, 13
- PYTHONIOENCODING, 13, 16
- PYTHONLEGACYWINDOWSFSENCODING, 15
- PYTHONLEGACYWINDOWSSSTDIO, 13, 15
- PYTHONMALLOC, 15, 33
- PYTHONMALLOCSTATS, 15
- PYTHONNODEBUGRANGES, 10, 17
- PYTHONNOUSERSITE, 8, 13
- PYTHONOPTIMIZE, 7, 12
- PYTHONPATH, 6, 11, 12, 60, 93, 94
- PYTHONPERFSUPPORT, 10, 17
- PYTHONPLATLIBDIR, 12
- PYTHONPROFILEIMPORTTIME, 9, 14
- PYTHONPYCACHEPREFIX, 10, 13
- PYTHONSAFEPAATH, 7, 12
- PYTHONSTARTUP, 7, 12, 13
- PYTHONTRACEMALLOC, 9, 14
- PYTHONUNBUFFERED, 8, 13
- PYTHONUSERBASE, 14
- PYTHONUTF8, 10, 16, 59
- PYTHONVERBOSE, 8, 13
- PYTHONWARNDEFAULTENCODING, 10, 16
- PYTHONWARNINGS, 9, 14
- version
 - opción de línea de comando, 6
- vista de diccionario, 102
- W**
- W
 - opción de línea de comando, 8
- walrus operator, 115
- with-address-sanitizer
 - opción de línea de comando, 35
- with-app-store-compliance
 - opción de línea de comando, 38
- with-assertions
 - opción de línea de comando, 35
- with-build-python
 - opción de línea de comando, 39
- with-builtin-hashlib-hashes
 - opción de línea de comando, 37
- with-computed-gotos
 - opción de línea de comando, 33
- with-dbmliborder
 - opción de línea de comando, 27
- with-dtrace
 - opción de línea de comando, 35
- with-ensurepip
 - opción de línea de comando, 32
- with-framework-name
 - opción de línea de comando, 38, 39
- with-hash-algorithm
 - opción de línea de comando, 37
- with-libc
 - opción de línea de comando, 36
- with-libm
 - opción de línea de comando, 36
- with-libs
 - opción de línea de comando, 36
- with-lto
 - opción de línea de comando, 32
- with-memory-sanitizer
 - opción de línea de comando, 35
- with-openssl
 - opción de línea de comando, 36
- with-openssl-rpath
 - opción de línea de comando, 37
- without-c-locale-coercion
 - opción de línea de comando, 27
- without-decimal-contextvar
 - opción de línea de comando, 27
- without-doc-strings
 - opción de línea de comando, 33
- without-mimalloc
 - opción de línea de comando, 33
- without-pymalloc
 - opción de línea de comando, 33
- without-readline
 - opción de línea de comando, 36
- without-remote-debug
 - opción de línea de comando, 34
- without-static-libpython
 - opción de línea de comando, 36
- with-pkg-config
 - opción de línea de comando, 27
- with-platlibdir
 - opción de línea de comando, 27
- with-pydebug
 - opción de línea de comando, 35
- with-readline
 - opción de línea de comando, 36
- with-ssl-default-suites
 - opción de línea de comando, 37
- with-strict-overflow
 - opción de línea de comando, 34
- with-suffix
 - opción de línea de comando, 26
- with-system-expat
 - opción de línea de comando, 36
- with-system-libmpdec
 - opción de línea de comando, 36
- with-tail-call-interp
 - opción de línea de comando, 33
- with-thread-sanitizer
 - opción de línea de comando, 35
- with-trace-refs
 - opción de línea de comando, 35
- with-tzpath

- opción de línea de comando, [27](#)
- with-undefined-behavior-sanitizer
 - opción de línea de comando, [35](#)
- with-universal-archs
 - opción de línea de comando, [38](#)
- with-valgrind
 - opción de línea de comando, [35](#)
- with-wheel-pkg-dir
 - opción de línea de comando, [27](#)

X

- x
 - opción de línea de comando, [9](#)
- X
 - opción de línea de comando, [9](#)

Z

- Zen de Python, [115](#)
- ZLIB_CFLAGS
 - opción de línea de comando, [31](#)
- ZLIB_LIBS
 - opción de línea de comando, [31](#)