Python Setup and Usage

Versión 3.8.20

Guido van Rossum
and the Python development team

septiembre 08, 2024

Python Software Foundation
Email: docs@python.org

Índice general

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.

CAPÍTULO 1

Línea de comandos y entorno

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

CPython implementation detail: 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 [-bBdEhiIOqsSuvVWx?] [-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:

- Cuando se llama con entrada estándar conectada a un dispositivo tty, solicita comandos y los ejecuta hasta que se lea un EOF (un carácter de fin de archivo, puede producirlo con `Ctrl-D` en UNIX o `Ctrl-Z, Enter` en Windows).
- 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`.

-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 se puede utilizar para ejecutar el script en modo aislado donde `sys.path` no contiene ni el directorio actual ni el directorio site-packages del usuario. También se omiten todas las variables de entorno `PYTHON*`.

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`.

Genera 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` se puede utilizar para ejecutar el script en modo aislado donde `sys.path` no contiene ni el directorio del script ni el directorio site-packages del usuario. También se omiten todas las variables de entorno `PYTHON*`.

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

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

-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]
```

Nuevo en la versión 3.6: La opción `-VV`.

1.1.3 Opciones diversas

-b

Emitir una advertencia al comparar `bytes` o `bytearray` con `str` o `bytes` con `int`. Emitir un error cuando la opción se da dos veces (`-bb`).

Distinto en la versión 3.5: Afecta a las comparaciones de `bytes` con `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

Active la salida de depuración del analizador (solo para expertos, dependiendo de las opciones de compilación). Véase también *PYTHONDEBUG*.

-E

Ignore todas las variables de entorno `PYTHON*`, por ejemplo *PYTHONPATH* y *PYTHONHOME*, que podrían establecerse.

-i

Cuando se pasa un script como primer argumento o se utiliza la opción *-c*, entre en modo interactivo después de ejecutar el script o el comando, incluso cuando `sys.stdin` no parece ser un terminal. El archivo *PYTHONSTARTUP* no se lee.

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

-I

Ejecute Python en modo aislado. Esto también implica -E y -s. En modo aislado `sys.path` no contiene ni el directorio del script ni el directorio site-packages del usuario. También se omiten todas las variables de entorno `PYTHON*`. Se pueden imponer restricciones adicionales para evitar que el usuario inyecte código malicioso.

Nuevo en la versión 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**.

-q

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

Nuevo en la versión 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.

En versiones anteriores de Python, esta opción activa la aleatorización de hash, de modo que los valores `__hash__()` de los objetos str y bytes son «saladas» con un valor aleatorio impredecible. Aunque permanecen constantes dentro de un proceso de Python individual, no son predecibles entre invocaciones repetidas de Python.

La aleatorización de hash está diseñada para proporcionar protección contra una denegación de servicio causada por entradas cuidadosamente elegidas que aprovechan el peor rendimiento de una construcción de dictado, la complejidad de O(n-2). Consulte http://www.ocert.org/advisories/ocert-2011-003.html para obtener más información.

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

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

Nuevo en la versión 3.2.3.

-s

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

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

Imprima 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 da dos veces (`-vv`), imprima un mensaje para cada archivo que se comprueba al buscar un módulo. También proporciona información sobre la limpieza del módulo en la salida. Véase también *PYTHONVERBOSE*.

-W `arg`

Control de advertencia. La maquinaria de advertencia de Python por defecto imprime mensajes de advertencia en `sys.stderr`. Un mensaje de advertencia típico tiene el siguiente formulario:

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

De forma predeterminada, cada advertencia se imprime una vez para cada línea de origen donde se produce. Esta opción controla la frecuencia con la que se imprimen las advertencias.

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. No se omiten las opciones *-W* (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`.

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
-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 (por ejemplo, `-Wi`, `-Wd`, `-Wa`, `-We`) y el intérprete los resolverá con el nombre de acción adecuado.

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`;
- `-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.
- `-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()` para obtener más información.
- `-X showalloccount` para generar el recuento total de objetos asignados para cada tipo cuando finalice el programa. Esto sólo funciona cuando Python se creó con `COUNT_ALLOCS` definido.
- `-X int_max_str_digits` configures the integer string conversion length limitation. See also *PYTHONINTMAXSTRDIGITS*.
- `-X importtime` para mostrar cuánto tiempo tarda cada importación. Muestra el nombre del módulo, el tiempo acumulado (incluidas las importaciones anidadas) y el tiempo de autoestima (excluyendo las importaciones anidadas). Tenga en cuenta que su salida puede romperse en aplicaciones multiproceso. El uso típico es `python3 -X importtime -c 'import asyncio'`. Véase también *PYTHONPROFILEIMPORTTIME*.
- `-X dev`: habilite el «modo de desarrollo» de CPython, introduciendo comprobaciones de tiempo de ejecución adicionales que son demasiado costosas para habilitarse de forma predeterminada. No debe ser más detallado que el valor predeterminado si el código es correcto: las nuevas advertencias solo se emiten cuando se detecta un problema. Efecto del modo de desarrollador:
  - Agregue el filtro de advertencia `default`, como *-W* `default`.
  - Instale los enlaces de depuración en los asignadores de memoria: vea la función `PyMem_SetupDebugHooks()` C.
  - Habilite el módulo `faulthandler` para volcar el rastreo de Python en un bloqueo.
  - Habilite asyncio debug mode.
  - Establezca el atributo `sys.flags.dev_mode` de `sys.flags` en `True`.
  - `io.IOBase` destructor registra las excepciones `close()`.

- `-X utf8` habilita el modo UTF-8 para las interfaces del sistema operativo, reemplazando el modo predeterminado compatible con la configuración regional. `-X utf8=0` desactiva explícitamente el modo UTF-8 (incluso cuando de lo contrario se activaría automáticamente). Consulte *PYTHONUTF8* para obtener más detalles.
- `-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*.

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

Distinto en la versión 3.2: Se ha añadido la opción *-X*.

Nuevo en la versión 3.3: La opción `-X faulhandler`.

Nuevo en la versión 3.4: Las opciones `-X showrefcount` y `-X tracemalloc`.

Nuevo en la versión 3.6: La opción `-X showalloccount`.

Nuevo en la versión 3.7: Las opciones `-X importtime`, `-X dev` y `-X utf8`.

Nuevo en la versión 3.8: La opción `-X pycache_prefix`. La opción `-X dev` ahora registra las excepciones `close()` en el destructor `io.IOBase`.

Nuevo en la versión 3.8.14: The `-X int_max_str_digits` option.

1.1.4 Opciones que no debe usar

-J

Reservado para su uso por Jython.

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/python`*version* y *exec_prefix*`/lib/python`*version*, donde *prefix* y *exec_prefix* son directorios dependientes de la instalación, ambos predeterminados: file:*/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/python`*version* (consulte *PYTHONHOME* arriba). Es *always* anexado 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`.

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 argumento `filename`.

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.

Nuevo en la versión 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.

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.

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

Distinto en la versión 3.8.20: 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 OS X.

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`.

Nuevo en la versión 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 autocomprobaciónes 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.

Nuevo en la versión 3.2.3.

PYTHONINTMAXSTRDIGITS

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

Nuevo en la versión 3.8.14.

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 Distutils installation paths para `python setup.py 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 Mac OS X.

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
```

(continué en la próxima página)

(proviene de la página anterior)

```
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

Si esta variable de entorno se establece en una cadena no vacía, se llama a `faulthandler.enable()` al inicio: instale un controlador para `SIGSEGV`, `SIGFPE`, `SIGABRT`, `SIGBUS` y `SIGILL` para volcar el seguimiento de Python. Esto es equivalente a la opción *-X* `faulthandler`.

Nuevo en la versión 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 el `tracemalloc.start()` para obtener más información.

Nuevo en la versión 3.4.

PYTHONPROFILEIMPORTTIME

Si esta variable de entorno se establece en una cadena no vacía, Python mostrará cuánto tiempo tarda cada importación. Esto equivale exactamente a establecer `-X importtime` en la línea de comandos.

Nuevo en la versión 3.7.

PYTHONASYNCIODEBUG

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

Nuevo en la versión 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 los dominios pymalloc allocator para `PYMEM_DOMAIN_MEM` y `PYMEM_DOMAIN_OBJ` y utilice la función `malloc()` para el dominio `PYMEM_DOMAIN_RAW`.

Instale los ganchos del debug:

- `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.

Consulte default memory allocators y la función `PyMem_SetupDebugHooks()` (instalar enlaces de depuración en los asignadores de memoria de Python).

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

Nuevo en la versión 3.6.

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 modo de codificación y errores del sistema de archivos 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*.

Esto también se puede habilitar en tiempo de ejecución con `sys._enablelegacywindowsfsencoding()`.

Availability: Windows.

Nuevo en la versión 3.6: Consulte **PEP 529** para obtener más detalles.

PYTHONLEGACYWINDOWSSTDIO

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.

Nuevo en la versión 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 subprocesos (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.

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

Availability: *nix.

Nuevo en la versión 3.7: Consulte **PEP 538** para obtener más detalles.

PYTHONDEVMODE
Si esta variable de entorno se establece en una cadena no vacía, habilite el «modo de desarrollo» de CPython. Consulte la opción *-X* `dev`.

Nuevo en la versión 3.7.

PYTHONUTF8
Si se establece en `1`, habilita el modo UTF-8 del intérprete, donde `UTF-8` se utiliza como codificación de texto para las interfaces del sistema, independientemente de la configuración regional actual.

Esto significa que:

- `sys.getfilesystemencoding()` devuelve `'UTF-8'` (se omite la codificación de configuración local).
- `locale.getpreferredencoding()` devuelve `'UTF-8'` (se omite la codificación de configuración regional y el parámetro `do_setlocale` de la función no tiene ningún efecto).
- `sys.stdin`, `sys.stdout`, y `sys.stderr` todos usan UTF-8 como su codificación de texto, con el `surrogateescape` error handler que se habilita para `sys.stdin` y `sys.stdout` (`sys.stderr` continúa utilizando `backslashreplace` como lo hace en el modo predeterminado de configuración local)

Como consecuencia de los cambios en esas API de nivel inferior, otras API de nivel superior también presentan diferentes comportamientos predeterminados:

- Los argumentos de línea de comandos, las variables de entorno y los nombres de archivo se descodifican en texto mediante la codificación UTF-8.
- `os.fsdecode()` y `os.fsencode()` utilizan la codificación UTF-8.
- `open()`, `io.open()`, y `codecs.open()` utilizan la codificación UTF-8 de forma predeterminada. Sin embargo, siguen usando el controlador de errores estricto de forma predeterminada para que intentar abrir un archivo binario en modo de texto sea probable que genere una excepción en lugar de producir datos sin sentido.

Tenga en cuenta que la configuración de secuencia estándar en modo UTF-8 se puede invalidar por *PYTHONIOENCODING* (igual que pueden estar en el modo predeterminado de configuración local).

Si se establece en `0`, el intérprete se ejecuta en su modo predeterminado compatible con la configuración local.

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

Si esta variable de entorno no se establece en absoluto, el intérprete utiliza de forma predeterminada la configuración regional actual, *a menos que* la configuración regional actual se identifique como una configuración regional basada en ASCII heredada (como se describe para *PYTHONCOERCECLOCALE*) y la coerción de configuración regional está deshabilitada o se produce un error. En estas configuraciones regionales heredadas, el intérprete habilitará de forma predeterminada el modo UTF-8 a menos que se indique explícitamente que no lo haga.

También disponible como la opción *-X* `utf8`.

Nuevo en la versión 3.7: Consulte **PEP 540** para obtener más detalles.

1.2.1 Variables de modo de depuración

Establecer estas variables solo tiene un efecto en una compilación de depuración de Python.

PYTHONTHREADDEBUG
Si se establece, Python imprimirá información de depuración de subprocesos.

Necesita configurar Python con la opción de compilación `--with-pydebug`.

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

Necesita Python configurado con la opción de compilación `--with-trace-refs`.

CAPÍTULO 2

Uso de Python en plataformas Unix

2.1 Obteniendo e instalando la última versión de Python

2.1.1 En Linux

Python viene preinstalado en la mayoría de distribuciones Linux, y también está disponible como paquete en el resto. Sin embargo, hay determinadas características que puede que quiera usar y no están disponibles en tu paquete de distribución. Puedes compilar fácilmente la última versión de Python de la fuente.

En caso de que Python no venga preinstalado y tampoco se encuentre en los repositorios, puede crear fácilmente paquetes para su propia distribución. Eche un vistazo a los siguientes enlaces:

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-old.fedoraproject.org/en-US/Fedora_Draft_Documentation/0.1/html/RPM_Guide/ch-creating-rpms.html
para los usuarios de Fedora

http://www.slackbook.org/html/package-management-making-packages.html para los usuarios de Slackware

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
```

(continué en la próxima página)

(proviene de la página anterior)

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
```

2.1.3 En OpenSolaris

Puede obtener Python de OpenCSW. Varias versiones de Python están disponibles y pueden ser instaladas, por ejemplo: `pkgutil -i python27`.

2.2 Construyendo Python

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

El proceso de construcción consta de los comandos habituales:

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

Las opciones de configuración y las advertencias para plataformas Unix específicas están ampliamente documentadas en el fichero README.rst del árbol de origen.

Advertencia: `make install` puede sobreescribir 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/python`*`version`*.

2.3 Rutas y archivos relacionados con Python

Estos están sujetos a diferencias según las convenciones de instalación locales; `prefix` (`${prefix}`) y `exec_prefix` (`${exec_prefix}`) son dependientes de la instalación y deben interpretarse como software GNU; deben ser iguales.

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

Archivo/directorio	Significado
`exec_prefix``/bin/python3`	Ubicación recomendada del intérprete.
`prefix``/lib/python`*`version`*, *`exec_prefix`*`/lib/python`*`version`*	Ubicaciones recomendadas de los directorios que contienen los módulos estándar.
`prefix``/include/python`*`version`*, *`exec_prefix`*`/include/python`*`version`*	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 Unices 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`.

CAPÍTULO 3

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.

A diferencia de la mayoría de sistemas y servicios Unix, Windows no incluye una instalación de Python soportada por el sistema. Para hacer que Python esté disponible, el equipo de CPython ha compilado instaladores de Windows (paquetes MSI) con cada lanzamiento durante muchos años. Estos instaladores están destinados principalmente a agregar una instalación de Python para cada usuario, con el intérprete principal y la biblioteca para ser usados por un solo usuario. El instalador también es capaz de hacer la instalación para todos los usuarios de una única máquina, y un archivo ZIP separado está disponible para distribuciones locales (específicas) para cada aplicación.

Como se especifica en **PEP 11**, una versión de Python solo soporta una plataforma Windows mientras Microsoft considere dicha plataforma bajo soporte extendido. Esto significa que Python 3.8 es compatible con Windows Vista y versiones posteriores. Si necesita compatibilidad con Windows XP entonces instale Python 3.4.

Hay varios instaladores diferentes disponibles para Windows, cada uno con determinados beneficios y desventajas.

El instalador completo contiene todos los componentes y es la mejor opción para desarrolladores que usan Python para cualquier clase de proyecto.

El paquete Microsoft Store es una instalación simple de Python que es adecuada para ejecutar scripts y paquetes, y para usar IDLE u otros entornos de desarrollo. Requiere Windows 10, pero la instalación puede hacerse de forma segura sin corromper otros programas. También proporciona muchos comandos convenientes para lanzar Python y sus herramientas.

El paquete de nuget.org son instalaciones ligeras destinadas a sistemas de integración continua. Puede ser usada para crear paquetes de Python o para ejecutar scripts, pero no es actualizable y no posee herramientas de interfaz de usuario.

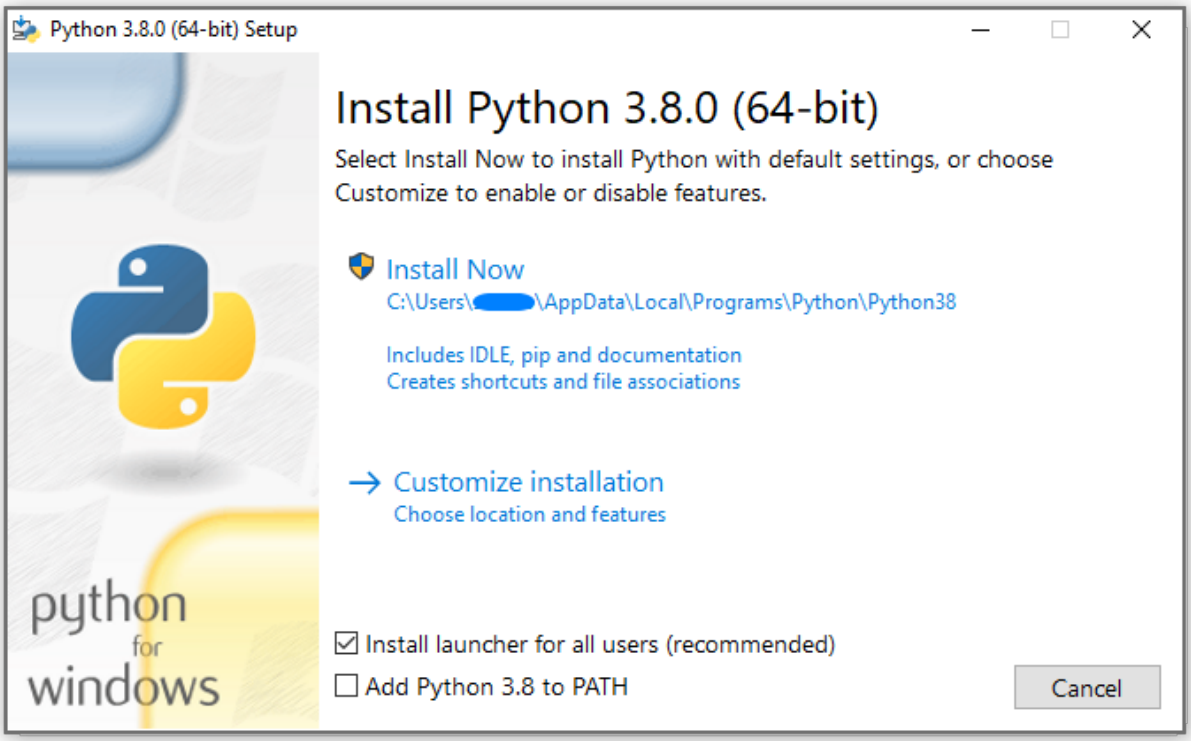
El paquete incrustable es un paquete de Python mínimo que es adecuado para incrustar en una aplicación más grande.

3.1 El instalador completo

3.1.1 Pasos para la instalación

Cuatro instaladores de Python 3.8 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 *Lanzador de Python para Windows* para todos los usuarios)
- Python será instalado en su directorio de usuario
- El *Lanzador de Python para Windows* 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 *Lanzador de Python para Windows* 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

3.1.2 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.

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

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

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.

3.1.3 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.

Para ocultar completamente la interfaz de usuario del instalador e instalar Python de forma silenciosa, use la opción `/quiet`. Para omitir la interacción con el usuario pero aún así mostrar el progreso y los errores, use la opción `/passive`. La opción `/uninstall` puede ser usada para comenzar a desinstalar Python inmediatamente - no se mostrará ninguna advertencia.

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\PythonXY` o `%LocalAppData%\Programs\PythonXY-32` o `%LocalAppData%\Programs\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	Agregar directorios de instalación y Scripts a `PATH` y `.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	Instalar encabezados y bibliotecas de desarrollo	1
Include_exe	Instalar `python.exe` y archivos relacionados	1
Include_launcher	Instalar *Lanzador de Python para Windows*.	1
InstallLauncherAllUsers	Instalar *Lanzador de Python para Windows* para todos los usuarios.	1
Include_lib	Instalar la biblioteca estándar y los módulos de extensión	1
Include_pip	Instalar los paquetes pip y setuptools	1
Include_symbols	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
SimpleInstallDescription	Un mensaje personalizado para mostrar cuando se use la versión simplificada de la interfaz de usuario de instalación.	(vacío)

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-3.8.0.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-3.8.0.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>
```

3.1.4 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 posibles archivos requeridos. Recuerde reemplazar `Python-3.8.0.exe` por el nombre real del instalador y crear una estructura de directorios propia para evitar colisiones entre archivos del mismo nombre.

```
python-3.8.0.exe /layout [optional target directory]
```

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

3.1.5 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 *Lanzador de Python para Windows*, el cual posee su propia entrada en Programas y características.

3.2 El paquete Microsoft Store

Nuevo en la versión 3.7.2.

El paquete de Microsoft Store es un intérprete de Python fácilmente instalable destinado principalmente al uso interactivo, por ejemplo, por estudiantes.

Para instalar el paquete, asegúrate de tener las últimas actualizaciones de Windows 10 y busca «Python 3.8» en Microsoft Store. Comprueba que la aplicación que seleccionas es una publicación de la Python Software Foundation e instálala.

Advertencia: Python siempre estará disponible de forma gratuita en Microsoft Store. Si se te solicita que pagues por él, entonces el paquete seleccionado no es el correcto.

Luego de la instalación, Python puede iniciarse a través del menú de Inicio. Como alternativa, también estará disponible desde cualquier símbolo del sistema o sesión de PowerShell al escribir `python`. Además, pip e IDLE pueden ser usados escribiendo `pip` o `idle`. IDLE también puede ser encontrado en el Inicio.

Los tres comandos también están disponibles con el número de versión como sufijo, por ejemplo, como `python3.exe` y `python3.x.exe` así como también `python.exe` (donde `3.x` es la versión específica que se quiera iniciar, como 3.8). Abrir «Administrar alias de ejecución de aplicaciones» a través del menú de Inicio para seleccionar cuál versión de Python se asocia con cada comando. Se recomienda asegurarse de que `pip` e `idle` sean consistentes con la versión de `python` que esté seleccionada.

Los entornos virtuales se pueden crear con `python -m venv` y se activan y usan normalmente.

Si ha instalado otra versión de Python que se haya agregado a la variable"PATH", estará disponible como `python.exe` en lugar de la de Microsoft Store. Para acceder a la nueva instalación, use `python3.exe` o `python3.x.exe`.

El lanzador `py.exe` detectará esta instalación de Python, pero priorizará instalaciones realizadas con el instalador tradicional.

Para eliminar Python, abra Configuración y utilice Aplicaciones y características, o encuentre Python en el Inicio y mediante click derecho seleccione Desinstalar. La desinstalación eliminará todos los paquetes instalados directamente en esta instalación de Python, pero no eliminará ningún entorno virtual

3.2.1 Problemas conocidos

Debido a restricciones en las aplicaciones de Microsoft Store, los scripts de Python podrían no tener acceso completo de escritura en ubicaciones compartidas como `TEMP` o el registro. En su lugar, se escribirá en una copia privada. Si sus scripts deben modificar las ubicaciones compartidas, necesitará instalar el instalador completo.

For more detail on the technical basis for these limitations, please consult Microsoft's documentation on packaged full-trust apps, currently available at docs.microsoft.com/en-us/windows/msix/desktop/desktop-to-uwp-behind-the-scenes

3.3 El paquete de nuget.org

Nuevo en la versión 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 específica, agregue `-Version 3.x.y`. El directorio de salida se puede cambiar desde `.`, y el paquete se instalará en un subdirectorio. Por defecto, el subdirectorio es nombrado con el mismo nombre del paquete, y sin la opción `-ExcludeVersion` este nombre incluirá la versión de instalación especificada. 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.

Las páginas de información del paquete en nuget.org son www.nuget.org/packages/python para la versión de 64 bit y www.nuget.org/packages/pythonx86 para la versión de 32 bit.

3.4 El paquete incrustable

Nuevo en la versión 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.

Al ser extraída, la distribución incrustable está (casi) completamente aislada del sistema del usuario, incluyendo variables de entorno, configuraciones del registro del sistema y paquetes instalados. La biblioteca estándar se incluye como archivos `.pyc` precompilados y optimizados dentro de un ZIP, y `python3.dll`, `python37.dll`, `python.exe` y `pythonw.exe` están todos proporcionados. Tcl/tk (incluidos sus dependientes, como Idle), pip y la documentación de Python no están incluidos.

Nota: La distribución incrustable no incluye el Microsoft C Runtime y la responsabilidad de proporcionarlo recae sobre el instalador de la aplicación. El runtime puede haber sido previamente instalado en el sistema de un usuario, o automáticamente vía Windows Update, y puede ser detectado encontrando `ucrtbase.dll` en el directorio del sistema.

Nota: When running on Windows 7, Python 3.8 requires the KB2533623 update to be installed. The embeddable distribution does not detect this update, and may fail at runtime. Later versions of Windows include this update.

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.

3.4.1 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.

3.4.2 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.

3.5 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`.

Canopy Un «entorno de análisis integral de Python» con editores y otras herramientas de desarrollo.

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.

3.6 Configuración de Python

Para ejecutar Python convenientemente desde el símbolo del sistema, puede considerar cambiar algunas variables de entorno predeterminadas de Windows. Si bien el instalador proporciona una opción para configurar las variables PATH y PATHEXT, esto solo es confiable para una única instalación en todo el sistema. Si utiliza varias versiones de Python con regularidad, considere usar el *Lanzador de Python para Windows*.

3.6.1 Excurso: configurar variables de entorno

Windows permite configurar las variables de entorno de forma permanente a nivel de usuario y a nivel de sistema, o temporalmente en el símbolo del sistema.

Para configurar una variable de entorno temporal, abra el símbolo del sistema y utilice el comando **`set`**:

```
C:\>set PATH=C:\Program Files\Python 3.8;%PATH%
C:\>set PYTHONPATH=%PYTHONPATH%;C:\My_python_lib
C:\>python
```

Estos cambios serán aplicados a cualquier comando que de aquí en más se ejecute en esa consola, y serán heredados por cualquier aplicación iniciada desde esa consola.

Si se incluye el nombre de la variable entre signos de porcentaje, esta se expande al valor existente, permitiendo agregar un nuevo valor tanto al principio como al final. Modificar `PATH` agregando el directorio que contiene **`python.exe`** al comienzo es una forma común de asegurar que se ejecuta la versión correcta de Python.

Para modificar permanentemente las variables de entorno predeterminadas, haga click en Inicio y busque "editar las variables de entorno", o abra Sistema, *Configuración avanzada del sistema* y haga click en el botón *Variables de entorno*. En este diálogo, se pueden agregar o modificar variables del usuario o del sistema. Para cambiar variables del sistema, se necesita acceso no restringido al equipo (por ej. con credenciales de administrador).

Nota: Windows concatenará las variables de usuario *luego* de las variables del sistema, lo cual puede causar resultados inesperados cuando se modifica `PATH`.

La variable *`PYTHONPATH`* es utilizada por todas las versiones de Python 2 y Python 3, por lo que no se debería configurar de forma permanente a menos que sólo incluya código que sea compatible con todas las versiones de Python instaladas.

Ver también:

https://www.microsoft.com/en-us/wdsi/help/folder-variables Variables de entorno en Windows NT

https://technet.microsoft.com/en-us/library/cc754250.aspx El comando SET, para modificar temporalmente variables de entorno

https://technet.microsoft.com/en-us/library/cc755104.aspx El comando SETX, para modificar permanentemente variables de entorno

https://support.microsoft.com/en-us/help/310519/how-to-manage-environment-variables-in-windows-xp Cómo gestionar variables de entorno en Windows XP

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

3.6.2 Encontrar el ejecutable de Python

Distinto en la versión 3.5.

Además de utilizar la entrada para el intérprete de Python creada automáticamente en el menú de Inicio, es posible que desee iniciar Python desde el símbolo del sistema. El instalador posee una opción para realizar esa configuración.

En la primera página del instalador, la opción llamada «Add Python to PATH» puede ser seleccionada para que el instalador agregue la ubicación de instalación a `PATH`. La ubicación del directorio `Scripts\` también es agregada. Esto permite escribir **`python`** para iniciar el intérprete, y **`pip`** para el instalador de paquetes. De esta manera los scripts también pueden ser ejecutados con opciones de línea de comandos, consulte la documentación de *Línea de comando*.

Si no se activa esta opción durante la instalación, en cualquier momento se puede ejecutar nuevamente el instalador, seleccionar Modify, y activarla. Otra alternativa es modificar `PATH` manualmente siguiendo las instrucciones en *Excurso: configurar variables de entorno*. Se necesita configurar la variable de entorno `PATH` para que incluya el directorio de instalación de Python, separándolo con punto y coma (;) de las otras entradas. Una variable de ejemplo pude verse así (suponiendo que las dos primeras entradas ya existían):

```
C:\WINDOWS\system32;C:\WINDOWS;C:\Program Files\Python 3.8
```

3.7 Modo UTF-8

Nuevo en la versión 3.7.

Windows aún utiliza codificación heredada para la codificación del sistema (la página de códigos ANSI). Python la utiliza para la codificación por defecto de archivos de texto (por ej. `locale.getpreferredencoding()`).

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 *Excurso: configurar variables de entorno* para saber cómo modificar las variables de entorno.

Cuando el modo UTF-8 está activado:

- `locale.getpreferredencoding()` retorna `'UTF-8'` en lugar de la codificación del sistema. Esta función es utilizada como la codificación de texto predeterminada en muchos lugares, incluidos `open()`, `Popen`, `Path.read_text()`, etc.
- `sys.stdin`, `sys.stdout`, y `sys.stderr` utilizan la codificación de texto UTF-8.
- Siempre se puede utilizar la codificación del sistema mediante el 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).
- Codificación del sistema de archivos (ver **PEP 529** para más detalles).

3.8 Lanzador de Python para Windows

Nuevo en la versión 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**.

3.8.1 Comenzar

Desde la línea de comandos

Distinto en la versión 3.6.

Las instalaciones en todo el sistema de Python 3.3 y posteriores agregarán la ubicación del lanzador a `PATH`. El lanzador es compatible con todas las versiones de Python disponibles, por lo que no importa cuál es la versión que está instalada. Para verificar 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 hay múltiples versiones de Python instaladas (por ej. 2.7 y 3.8) habrá notado que se inició Python 3.8 - para iniciar Python 2.7, ejecute el comando:

```
py -2.7
```

Si se quiere la última versión instalada de Python 2.x, ejecute el comando:

```
py -2
```

La última versión de Python 2.x debería iniciarse.

Si ve el siguiente error es porque el lanzador no está instalado:

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

Las instalaciones por usuario de Python no agregan la ubicación del lanzador a `PATH` a menos que la opción haya sido seleccionada durante la instalación.

Entornos virtuales

Nuevo en la versión 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 donde se encuentra 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
```

Al ejecutar nuevamente el comando se debería imprimir la información del último Python 3.x. Al igual que en los ejemplos de línea de comandos anteriores, se puede especificar un calificador de versión más explícito. Suponiendo que tiene instalado Python 2.6, pruebe cambiar la primera línea a `#! python2.6` y debería ver que se imprime la información de la versión 2.6.

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.

3.8.2 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 python`
- `/usr/bin/python`
- `/usr/local/bin/python`
- `python`

Por ejemplo, si la primera línea del script comienza con

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

La versión de Python predeterminada será ubicada y utilizada. Como muchos scripts de Python escritos para funcionar en Unix tienen esta línea, debería suceder que estos scripts pueden ser utilizados por el lanzador sin modificaciones. Si está escribiendo un nuevo script en Windows que espera que sea útil en Unix, debería utilizar una de las líneas shebang que comienza con `/usr`.

A cualquiera de los mencionados comandos virtuales se le puede agregar la versión explícita como sufijo (ya sea solo la versión mayor, o la versión mayor y menor). Además se puede solicitar la versión de 32 bit agregando «-32» detrás de la versión menor. Por ej. `/usr/bin/python2.7-32` solicitará el uso de la versión de 32 bit de Python 2.7.

Nuevo en la versión 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`).

La forma `/usr/bin/env` de la línea shebang tiene un significado especial más. Antes de buscar intérpretes de Python instalados, esta forma buscará el ejecutable de Python en `PATH`. Esto se corresponde con el comportamiento en Unix del programa `env`, el cual realiza una búsqueda en `PATH`.

3.8.3 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`

3.8.4 Personalización

Personalización con archivos INI

El lanzador buscará dos archivos .ini - `py.ini` en el directorio de «datos de aplicación» del usuario actual (esto es el directorio retornado por el llamado a la función de Windows `SHGetFolderPath` con `CSIDL_LOCAL_APPDATA`) y `py.ini` en el directorio del lanzador. Los mismos archivos .ini son usados por la versión "consola" del lanzador (py.exe) y por la versión "ventana" (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 pude 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.
- Los comandos `python3.1` y `python2.7` no consultarán ninguna opción ya que las versiones se encuentran completamente especificadas.
- Si `PY_PYTHON=3`, los comandos `python` y `python3` utilizarán ambos la última versión instalada de Python 3.
- Si `PY_PYTHON=3.1-32`, el comando `python` utilizará la implementación de 32 bit de la versión 3.1 mientras que el comando `python3` utilizará el último Python instalado (PY_PYTHON no se consideró para nada ya que se especificó una versión mayor).
- Si `PY_PYTHON=3` y `PY_PYTHON3=3.1`, los comandos `python` y `python3` utilizarán ambos 3.1 específicamente

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:

- Configurar `PY_PYTHON=3.1` es equivalente a un archivo INI con el contenido:

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

- Configurar `PY_PYTHON=3` y `PY_PYTHON3=3.1` es equivalente a un archivo INI con el contenido:

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

3.8.5 Diagnóstico

Si se configura la variable de entorno `PYLAUNCH_DEBUG` (con cualquier valor), el lanzador imprimirá información de diagnóstico a stderr (en la consola). Aunque esta información es a la vez detallada y concisa, debería permitirle ver qué versiones de Python fueron encontradas, por qué se eligió una versión particular y la línea de comandos exacta que fue utilizada para ejecutar el Python escogido.

3.9 Encontrar módulos

Python generalmente almacena su biblioteca (y por lo tanto el directorio site-packages) en el directorio de instalación. Por lo tanto si Python fue instalado en `C:\Python\`, la biblioteca predeterminada residirá en `C:\Python\Lib\` y los módulos de terceros deberían almacenarse en `C:\Python\Lib\site-packages\`.

Para sobrescribir `sys.path` completamente, crear un archivo `._pth` con el mismo nombre que la DLL (`python37._pth`) o el ejecutable (`python._pth`) y especificar una línea por cada ruta a agregar a `sys.path`. El archivo basado en el nombre de la DLL tiene precedencia sobre el basado en el ejecutable, lo que permite restringir las rutas para cualquier programa que cargue el tiempo de ejecución si se desea.

Cuando el archivo existe, se ignoran todas las variables de entorno y del registro, se activa el modo aislado, y no se importa `site` a menos que una línea en el archivo especifique `import site`. Rutas en blanco y líneas que comiencen con `#` son ignoradas. Cada ruta puede ser absoluta o relativa a la ubicación del archivo. No se permiten declaraciones de importación más que la de `site`, y no se puede especificar código arbitrario.

Tenga en cuenta que los archivos `.pth` (sin guion bajo al inicio) serán procesados normalmente por el módulo `site` cuando `import site` haya sido especificado.

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`.
- Si se carga `python3.dll` o `python37.dll` desde un ejecutable propio, invocar explícitamente `Py_SetPath()` o (al menos) `Py_SetProgramName()` antes de `Py_Initialize()`.
- 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:

- Agrega soporte para archivos `._pth` y elimina la opción `applocal` de `pyvenv.cfg`.
- Agrega `pythonXX.zip` como un potencial archivo de referencia cuando se encuentra junto al ejecutable.

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.

3.10 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.

3.10.1 PyWin32

El módulo PyWin32 de Mark Hammond es una colección de módulos para soporte avanzado específico para Windows. Este incluye utilidades para:

- Component Object Model (COM)
- Invocación de la API Win32
- Registro
- Registro de eventos
- Interfaces de usuario para Microsoft Foundation Classes (MFC)

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

3.10.2 cx_Freeze

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

3.10.3 WConio

Dado que la capa de manejo avanzado de terminales de Python, `curses`, se encuentra restringida a sistemas tipo Unix, también hay una biblioteca exclusiva para Windows: Windows Console I/O para Python.

WConio es un contenedor para `CONIO.H` de Turbo-C, utilizado para crear interfaces de usuario de texto.

3.11 Compilar Python en Windows

Si desea compilar CPython por su cuenta, lo primero que debe hacer es obtener el código fuente. Puede descargar el código fuente de la última versión o simplemente obtener una nueva copia.

El árbol del código fuente contiene una solución de compilación y archivos del proyecto para Microsoft Visual Studio 2015, el cual es el compilador utilizado para compilar las versiones oficiales de Python. Estos archivos se encuentran 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.

Ver también:

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

3.12 Otras plataformas

Con el continuo desarrollo de Python, algunas plataformas que solían ser compatibles ya no lo son (debido a la falta de usuarios o desarrolladores). Consulte **PEP 11** para detalles sobre las plataformas no soportadas.

- Windows CE es aún soportado.
- El instalador de Cygwin también ofrece instalar el intérprete de Python (consulte Cygwin package source, Maintainer releases)

Para obtener información detallada acerca de las plataformas con instaladores precompilados consulte Python for Windows.

CAPÍTULO 4

Usando Python en una Macintosh

Autor Bob Savage <bobsavage@mac.com>

Python en una Macintosh con Mac OS X es, en principio, muy similar a Python en cualquier otra plataforma Unix, pero hay una serie de características adicionales como el IDE y el Administrador de paquetes que vale la pena resaltar.

4.1 Obteniendo e instalando MacPython

Mac OS X 10.8 viene con Python 2.7 preinstalado por Apple. Si lo desea, está invitado a instalar la versión más reciente de Python 3 desde el sitio web de Python (https://www.python.org). Allí está disponible una versión actual "binaria universal" de Python, que se ejecuta de forma nativa en las nuevas CPU Intel y PPC heredadas de Mac.

Lo que obtienes después de instalar es una serie de cosas:

- A `Python 3.8` folder in your `Applications` folder. In here you find IDLE, the development environment that is a standard part of official Python distributions; and PythonLauncher, which handles double-clicking Python scripts from the Finder.
- Un *framework* `/Library/Frameworks/Python.framework`, el cual incluye los ejecutables y librerías de Python. El instalador añade esta ubicación a su variable de entorno. Para desinstalar MacPython, usted puede simplemente remover estas tres cosas. Un enlace simbólico al ejecutable de Python es colocado en /usr/local/bin/.

La compilación proporcionada por Apple de Python se instala en `/System/Library/Frameworks/Python.framework` y `/usr/bin/python`, respectivamente. Nunca debe modificarlos ni eliminarlos, ya que están controlados por Apple y son utilizados por software de Apple o de terceros. Recuerde que si elige instalar una versión más reciente de Python desde python.org, tendrá dos instalaciones de Python diferentes pero funcionales en su computadora, por lo que será importante que sus rutas y usos sean consistentes con lo que desea hacer.

IDLE incluye un menú de ayuda que le permite acceder a la documentación de Python. Si es completamente nuevo en Python, debe comenzar a leer la introducción del tutorial en ese documento.

Si está familiarizado con Python en otras plataformas Unix, debe leer la sección sobre cómo ejecutar scripts Python desde el *shell* de Unix.

4.1.1 Cómo ejecutar un *script* de Python

Su mejor manera de comenzar a usar Python en Mac OS X es a través del entorno de desarrollo integrado IDLE, consulte la sección *El IDE* y use el menú Ayuda cuando se ejecute el IDE.

Si desea ejecutar scripts de Python desde la línea de comandos de la ventana de Terminal o desde Finder, primero necesita un editor para crear su script. Mac OS X viene con una serie de editores de línea de comandos estándar de Unix, **vim** y **emacs** entre ellos. Si desea un editor más parecido a Mac, **BBEdit** o **TextWrangler** de Bare Bones Software (consulte http://www.barebones.com/products/bbedit/index.html) son buenas opciones , ya que son **TextMate** (consulte https://macromates.com/). Otros editores incluyen **Gvim** (http://macvim-dev.github.io/macvim/) y **Aquamacs** (http://aquamacs.org/).

Para ejecutar su *script* desde la ventana Terminal, debe asegurarse de que: `/usr/local/bin` esté en su ruta de búsqueda de *shell*.

Para ejecutar su *script* desde el Finder, tiene dos opciones:

- Arrástrelo a **PythonLauncher**
- Seleccione **PythonLauncher** como aplicación predeterminada para abrir su *script* (o cualquier *script* .py) a través de la ventana de información del buscador y haga doble clic en ella. **PythonLauncher** tiene varias preferencias para controlar cómo se inicia su secuencia de comandos. La opción de arrastrar le permite cambiarlos para una invocación, o usar su menú de Preferencias para cambiar las cosas globalmente.

4.1.2 Ejecutar scripts con una GUI

Con versiones anteriores de Python, hay una peculiaridad de Mac OS X que debe conocer: los programas que hablan con el administrador de ventanas Aqua (en otras palabras, cualquier cosa que tenga una GUI) deben ejecutarse de una manera especial. Use **pythonw** en vez de **python** para comenzar tales scripts.

Con Python 3.8, usted podrá utilizar ya sea **python** o **pythonw**.

4.1.3 Configuración

Python en OS X respeta todas las variables de entorno estándar de Unix como *PYTHONPATH*, pero configurar estas variables para programas iniciados desde el Finder no es estándar ya que Finder no lee su `.profile` o `.cshrc` al arranque. Usted necesita crear un nuevo archivo `~/.MacOSX/environment.plist`. Consulte el documento técnico de Apple QA1067 para más detalles.

Para obtener más información sobre la instalación de paquetes de Python en MacPython, consulte la sección *Instalación de paquetes adicionales de Python*.

4.2 El IDE

MacPython se entrega con el entorno de desarrollo IDLE estándar. Se puede encontrar una buena introducción al uso de IDLE en http://www.hashcollision.org/hkn/python/idle_intro/index.html.

4.3 Instalación de paquetes adicionales de Python

Existen varios métodos para instalar paquetes Python adicionales:

- Los paquetes se pueden instalar a través del modo *distutils* estándar de Python (`python setup.py install`).
- Muchos paquetes también se pueden instalar a través de la extensión **`setuptools`** o el *wrapper* **`pip`**, consulte https://pip.pypa.io/.

4.4 Programación de GUI en Mac

Hay varias opciones para crear aplicaciones GUI en Mac con Python.

PyObjC es un enlace de Python al *framework* Objective-C/Cocoa de Apple, que es la base del desarrollo más moderno de Mac. La información sobre PyObjC está disponible en https://pypi.org/project/pyobjc/.

El kit de herramientas estándar de Python GUI es `tkinter`, basado en el kit de herramientas Tk multiplataforma (https://www.tcl.tk). Apple incluye una versión nativa de Aqua de Tk, y la última versión puede ser descargada e instalada desde https://www.activestate.com; También se puede incorporar desde la fuente.

wxPython es otro kit de herramientas GUI multiplataforma popular que se ejecuta de forma nativa en Mac OS X. Los paquetes y la documentación están disponibles en https://www.wxpython.org.

PyQt es otro kit de herramientas GUI multiplataforma popular que se ejecuta de forma nativa en Mac OS X. Más información podrá ser encontrada en https://riverbankcomputing.com/software/pyqt/intro.

4.5 Distribuyendo aplicaciones de Python en la Mac

La herramienta estándar para implementar aplicaciones independientes de Python en Mac es **`py2app`**. Puede encontrar más información sobre la instalación y el uso de py2app en http://undefined.org/python/#py2app.

4.6 Otros recursos

La lista de correo de MacPython es un excelente recurso de soporte para usuarios y desarrolladores de Python en Mac:

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

Otro recurso útil es el wiki de MacPython:

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

CAPÍTULO 5

Editores e IDEs

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**.

Diríjase a Python Editors y a Integrated Development Environments para obtener una lista completa.

APÉNDICE A

Glosario

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

. . . Puede referirse a:

- El prompt en el shell interactivo de Python por omisión cuando se ingresa código para un bloque indentado de código, y cuando se encuentra entre dos delimitadores que emparejan (paréntesis, corchetes, llaves o comillas triples), o después de especificar un decorador.
- La constante incorporada `Ellipsis`.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

a la espera Es un objeto a la espera (*awaitable*) que puede ser usado en una expresión `await`. Puede ser una *coroutine* o un objeto con un método `__await__()`. Vea también **PEP 492**.

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

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

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

objetos tipo binarios Un objeto que soporta bufferobjects y puede exportar un 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 corren en una *virtual machine* que ejecute el código de máquina correspondiente a cada *bytecode*. Note que los *bytecodes* no tienen como requisito trabajar en las diversas máquina virtuales de Python, ni de ser estable entre versiones Python.

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

callback A subroutine function which is passed as an argument to be executed at some point in the future.

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

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

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

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

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

variable de contexto Una variable que puede tener diferentes valores dependiendo del contexto. Esto es similar a un almacenamiento de hilo local *Thread-Local Storage* en el cual cada hilo de ejecución puede tener valores diferentes para una variable. Sin embargo, con las variables de contexto, podría haber varios contextos en un hilo de ejecución y el uso principal de las variables de contexto es mantener registro de las variables en tareas concurrentes asíncronas. Vea `contextvars`.

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 vería 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 se iniciadas, finalizadas y reanudadas en muchos puntos diferentes. Pueden ser implementadas con la sentencia `async def`. Vea además **PEP 492**.

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

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

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

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

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

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

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

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

Para mayor información sobre los métodos de los descriptores vea descriptors.

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

dictionary comprehension A compact way to process all or part of the elements in an iterable and return a dictionary with the results. `results = {n: n ** 2 for n in range(10)}` generates a dictionary containing key `n` mapped to value `n ** 2`. See 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 Una cadena de caracteres literal que aparece como la primera expresión en una clase, función o módulo. Aunque es ignorada cuando se ejecuta, es reconocida por el compilador y puesta en el atributo `__doc__` de la clase, función o módulo comprendida. Como está disponible mediante introspección, es el lugar canónico para ubicar la documentación del objeto.

tipado de pato Un estilo de programación que no revisa el tipo del objeto para determinar si tiene la interfaz correcta; en vez de ello, el método o atributo es simplemente llamado o usado («Si se ve como un pato y grazna como un

pato, debe ser un pato»). Enfatizando las interfaces en vez de hacerlo con los tipos específicos, un código bien diseñado pues tener mayor flexibilidad permitiendo la sustitución polimórfica. El tipado de pato *duck-typing* evita usar pruebas llamando a `type()` o `isinstance()`. (Nota: si embargo, el tipado de pato puede ser complementado con *abstract base classes*. En su lugar, generalmente pregunta con `hasattr()` o *EAFP*.

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

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

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

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

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

Existen tres categorías de objetos archivo: crudos *raw archivos binarios*, con 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*.

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

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

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

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

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

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

Las anotaciones de funciones son usadas frecuentemente para *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.

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

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

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

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

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

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

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

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

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

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

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

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

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

GIL Vea *global interpreter lock*.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

iterable 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.

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

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

Puede encontrar más información en typeiter.

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

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

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

argumento nombrado Vea *argument*.

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

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

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

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

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

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

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

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

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

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

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

Más información hallará en metaclasses.

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

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

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

Vea también *package*.

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

MRO Vea *method resolution order*.

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

tupla nombrada 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
```

Algunas tuplas nombradas con tipos incorporados (como en los ejemplo precedentes). También puede ser creada con una definición regular de clase que hereda de la clase `tuple` y que define campos nombrados. Una clase como esta puede ser hechas personalizadamente o puede ser creada con la función factoría `collections.namedtuple()`. Esta última técnica automáticamente brinda métodos adicionales que pueden no estar presentes en las tuplas nombradas personalizadas o incorporadas.

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

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

Vea también *module*.

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

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

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

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

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

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

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

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

- *sólo posicional*: especifica un argumento que puede ser pasado sólo por posición. 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 Un invocable en la lista `sys.path_hook` que retorna un *path entry finder* si éste sabe cómo encontrar módulos en un *path entry* específico.

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

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

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

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

Vea **PEP 1**.

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

argumento posicional Vea *argument*.

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

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

Este proceso permite que la biblioteca estándar continúe evolucionando con el tiempo, sin bloquearse por errores de diseño problemáticos por períodos extensos de tiempo. Vea **PEP 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 El número de referencias a un objeto. Cuando el contador de referencias de un objeto cae hasta cero, éste es desalojable. En conteo de referencias no suele ser visible en el código de Python, pero es un elemento clave para la implementación de *CPython*. El módulo `sys` define la `getrefcount()` que los programadores pueden emplear para retornar el conteo de referencias de un objeto en particular.

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

Vea también *namespace package*.

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

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()`.

set comprehension A compact way to process all or part of the elements in an iterable and return a set with the results. `results = {c for c in 'abracadabra' if c not in 'abc'}` generates the set of strings `{'r', 'd'}`. See 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`.

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

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

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

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

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

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

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

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

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

```
from typing import List, Tuple

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

podría ser más legible así:

```
from typing import List, Tuple

Color = Tuple[int, int, int]

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

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

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

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

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

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

saltos de líneas universales Una manera de interpretar flujos de texto en la cual son reconocidos como finales de línea todas siguientes formas: la convención de Unix para fin de línea `'\n'`, la convención de Windows `'\r\n'`, y

la vieja convención de Macintosh `'\r'`. Vea **PEP 278** y **PEP 3116**, además de `bytes.splitlines()` para usos adicionales.

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

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

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

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

```
count: int = 0
```

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

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

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

Vea también `venv`.

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

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

APÉNDICE B

Acerca de estos documentos

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

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

Agradecemos a:

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

B.1 Contribuidores de la documentación de Python

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

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

APÉNDICE C

History and License

C.1 History of the software

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

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

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

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

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

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

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

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

Python software and documentation are licensed under the *PSF License Agreement*.

Starting with Python 3.8.6, examples, recipes, and other code in the documentation are dual licensed under the PSF License Agreement and the *Zero-Clause BSD license*.

Some software incorporated into Python is under different licenses. The licenses are listed with code falling under that license. See *Licenses and Acknowledgements for Incorporated Software* for an incomplete list of these licenses.

C.2.1 PSF LICENSE AGREEMENT FOR PYTHON 3.8.20

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

2. Subject to the terms and conditions of this License Agreement, PSF hereby
   grants Licensee a nonexclusive, royalty-free, world-wide license to␣
↪reproduce,
   analyze, test, perform and/or display publicly, prepare derivative works,
   distribute, and otherwise use Python 3.8.20 alone or in any derivative
   version, provided, however, that PSF's License Agreement and PSF's notice␣
↪of
   copyright, i.e., "Copyright © 2001-2023 Python Software Foundation; All␣
↪Rights
   Reserved" are retained in Python 3.8.20 alone or in any derivative version
   prepared by Licensee.

3. In the event Licensee prepares a derivative work that is based on or
   incorporates Python 3.8.20 or any part thereof, and wants to make the
   derivative work available to others as provided herein, then Licensee␣
↪hereby
   agrees to include in any such work a brief summary of the changes made to␣
↪Python
   3.8.20.

4. PSF is making Python 3.8.20 available to Licensee on an "AS IS" basis.
   PSF MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED.  BY WAY OF
   EXAMPLE, BUT NOT LIMITATION, PSF MAKES NO AND DISCLAIMS ANY REPRESENTATION␣
↪OR
   WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT␣
↪THE
   USE OF PYTHON 3.8.20 WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
```

```
5. PSF SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF PYTHON 3.8.20
   FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT␣
↪OF
   MODIFYING, DISTRIBUTING, OR OTHERWISE USING PYTHON 3.8.20, OR ANY␣
↪DERIVATIVE
   THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

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

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

8. By copying, installing or otherwise using Python 3.8.20, Licensee agrees
   to be bound by the terms and conditions of this License Agreement.
```

C.2.2 BEOPEN.COM LICENSE AGREEMENT FOR PYTHON 2.0

BEOPEN PYTHON OPEN SOURCE LICENSE AGREEMENT VERSION 1

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

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

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

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

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

C.2.3 CNRI LICENSE AGREEMENT FOR PYTHON 1.6.1

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

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

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

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

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

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

C.2.4 CWI LICENSE AGREEMENT FOR PYTHON 0.9.0 THROUGH 1.2

```
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 3.8.20 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
```

(continué en la próxima página)

(proviene de la página anterior)

```
OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

C.3 Licenses and Acknowledgements for Incorporated Software

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

C.3.1 Mersenne Twister

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

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

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

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

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

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

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

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

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


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

C.3.2 Sockets

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

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

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

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

C.3.3 Asynchronous socket services

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

```
Copyright 1996 by Sam Rushing

                        All Rights Reserved

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

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

C.3.4 Cookie management

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

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

               All Rights Reserved

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

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

C.3.5 Execution tracing

The `trace` module contains the following notice:

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

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

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

Copyright 1995-1997, Automatrix, Inc., all rights reserved.
Author: Skip Montanaro

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


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

C.3.6 UUencode and UUdecode functions

The `uu` module contains the following notice:

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

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

C.3.7 XML Remote Procedure Calls

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

```
    The XML-RPC client interface is

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

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

```
ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE
OF THIS SOFTWARE.
```

C.3.8 test_epoll

The `test_epoll` module contains the following notice:

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

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

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

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

C.3.9 Select kqueue

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

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

```
OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
SUCH DAMAGE.
```

C.3.10 SipHash24

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

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

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

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

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

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

C.3.11 strtod and dtoa

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

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

(continué en la próxima página)

(proviene de la página anterior)

```
*
*****************************************************************/
```

C.3.12 OpenSSL

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

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

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

```
    *    being used are not cryptographic related :-).
    * 4. If you include any Windows specific code (or a derivative thereof) from
    *    the apps directory (application code) you must include an acknowledgement:
    *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
    *
    * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG ``AS IS'' AND
    * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
    * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
    * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
    * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
    * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
    * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
    * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
    * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
    * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
    * SUCH DAMAGE.
    *
    * The licence and distribution terms for any publically available version or
    * derivative of this code cannot be changed.  i.e. this code cannot simply be
    * copied and put under another distribution licence
    * [including the GNU Public Licence.]
    */
```

C.3.13 expat

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

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

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

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

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

C.3.14 libffi

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

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

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

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

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

C.3.15 zlib

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

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

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

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

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

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

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

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

C.3.16 cfuhash

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

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

This code is released under the BSD license:

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

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

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

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

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

C.3.17 libmpdec

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

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

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

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

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

(continué en la próxima página)

(proviene de la página anterior)

```
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 W3C C14N test suite

The C14N 2.0 test suite in the `test` package (`Lib/test/xmltestdata/c14n-20/`) was retrieved from the W3C website at https://www.w3.org/TR/xml-c14n2-testcases/ and is distributed under the 3-clause BSD license:

> 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 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.

APÉNDICE D

Copyright

Python and this documentation is:

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

Índice

No alfabético

A

B

C

D

E

F

G

H

I

J

L

M

N