What's New in Python

Versión 3.8.20

A. M. Kuchling

septiembre 08, 2024

Python Software Foundation
Email: docs@python.org

Índice general

Editor Raymond Hettinger

Este artículo explica las nuevas características de Python 3.8, en comparación con 3.7. Para obtener los detalles completos, consultar el registro de cambios.

Python 3.8 fue lanzado el 14 de octubre de 2019.

1 Resumen – Aspectos destacados de la versión

2 Nuevas características

2.1 Expresiones de asignación

La nueva sintaxis `:=` asigna valores a variables como parte de una expresión más grande. Se le conoce cariñosamente como «el operador morsa» debido a su parecido con los ojos y colmillos de una morsa.

En el siguiente ejemplo, la expresión de asignación ayuda evitando que se tenga que llamar a `len()` dos veces:

```
if (n := len(a)) > 10:
    print(f"List is too long ({n} elements, expected <= 10)")
```

Un beneficio similar surge durante la búsqueda de coincidencias mediante expresiones regulares donde los objetos de coincidencias se necesitan dos veces, una para comprobar si se produjo una coincidencia y otra para extraer un subgrupo:

```
discount = 0.0
if (mo := re.search(r'(\d+)% discount', advertisement)):
    discount = float(mo.group(1)) / 100.0
```

El operador también es útil en bucles while que calculan un valor para comprobar la terminación del bucle y posteriormente necesitan nuevamente ese mismo valor en el cuerpo del bucle:

```
# Loop over fixed length blocks
while (block := f.read(256)) != '':
    process(block)
```

Otra motivación de uso surge en las listas por comprensión en las que un valor calculado en una condición de filtrado también se necesita en el cuerpo de la expresión:

```
[clean_name.title() for name in names
 if (clean_name := normalize('NFC', name)) in allowed_names]
```

Intenta limitar el uso del «operador morsa» a aquellos casos en los que reduce la complejidad y mejora la legibilidad del código.

Consultar **PEP 572** para obtener una descripción completa.

(Contribución de Emily Morehouse en bpo-35224.)

2.2 Parámetros solo posicionales

Hay una nueva sintaxis para establecer parámetro de función, `/`, para indicar que algunos parámetros de función deben especificarse solo posicionalmente y no pueden usarse como argumentos por palabra clave. Esta es la misma notación que muestra `help()` para las funciones de C anotadas con la herramienta Argument Clinic de Larry Hastings.

En el siguiente ejemplo, los parámetros *a* y *b* son solo posicionales, mientras que *c* o *d* pueden ser posicionales o por palabra clave y *e* o *f* deben proporcionarse por palabra clave exclusivamente:

```
def f(a, b, /, c, d, *, e, f):
    print(a, b, c, d, e, f)
```

La siguiente es una invocación válida:

```
f(10, 20, 30, d=40, e=50, f=60)
```

Sin embargo, estas son invocaciones inválidas:

```
f(10, b=20, c=30, d=40, e=50, f=60)   # b cannot be a keyword argument
f(10, 20, 30, 40, 50, f=60)           # e must be a keyword argument
```

Un caso de uso de esta notación es permitir que las funciones puras de Python emulen completamente los comportamientos de las funciones codificadas en C existentes. Por ejemplo, la función incorporada `divmod()` no acepta argumentos por palabra clave:

```
def divmod(a, b, /):
    "Emulate the built in divmod() function"
    return (a // b, a % b)
```

Otro caso de uso es excluir los argumentos por palabra clave cuando el nombre del parámetro no es útil. Por ejemplo, la función incorporada `len()` tiene la firma `len(obj, /)`. Esto excluye llamadas inoportunas como:

```
len(obj='hello')  # The "obj" keyword argument impairs readability
```

Un beneficio adicional de marcar un parámetro como solo posicional es que permite cambiar el nombre del parámetro en el futuro sin riesgo de romper el código del cliente. Por ejemplo, en el módulo `statistics`, el nombre del parámetro *dist* puede cambiarse en el futuro. Esto es posible gracias a la siguiente especificación de función:

```
def quantiles(dist, /, *, n=4, method='exclusive')
    ...
```

Dado que los parámetros a la izquierda de / no se exponen como posibles palabras clave, los nombres de los parámetros permanecen disponibles para su uso en **kwargs:

```
>>> def f(a, b, /, **kwargs):
...     print(a, b, kwargs)
...
>>> f(10, 20, a=1, b=2, c=3)         # a and b are used in two ways
10 20 {'a': 1, 'b': 2, 'c': 3}
```

Esto simplifica enormemente la implementación de funciones y métodos que necesitan aceptar argumentos por palabra clave arbitrarios. Por ejemplo, aquí hay un extracto del código del módulo collections:

```
class Counter(dict):

    def __init__(self, iterable=None, /, **kwds):
        # Note "iterable" is a possible keyword argument
```

Consultar **PEP 570** para obtener una descripción completa.

(Contribución de Pablo Galindo en bpo-36540.)

2.3 Caché del sistema de archivos paralelo para archivos de bytecode compilados

La nueva configuración PYTHONPYCACHEPREFIX (también disponible mediante la opción -X pycache_prefix) configura la caché implícita de bytecode para que use un árbol del sistema de archivos paralelo separado, en lugar de los subdirectorios __pycache__ predeterminados dentro cada directorio de origen.

La ubicación de la caché se define en sys.pycache_prefix (None indica la ubicación predeterminada en los subdirectorios __pycache__).

(Contribución de Carl Meyer en bpo-33499.)

2.4 La compilación de depuración usa la misma ABI que la compilación de lanzamiento

Python ahora usa la misma ABI, independientemente de que esté compilado en modo de lanzamiento o de depuración. En Unix, cuando Python se compila en modo de depuración, ahora es posible cargar extensiones C compiladas en modo de lanzamiento y extensiones C compiladas usando la ABI estable.

Las versiones de lanzamiento y las versiones de depuración ahora son ABI compatibles: definir la macro Py_DEBUG ya no implica la macro Py_TRACE_REFS, que introducía la única incompatibilidad de la ABI. La macro Py_TRACE_REFS, que agrega la función sys.getobjects() y la variable de entorno PYTHONDUMPREFS, se puede establecer usando la nueva opción de compilación ./configure --with-trace-refs. (Contribución de Victor Stinner en bpo-36465.)

En Unix, las extensiones en C ya no están enlazadas a libpython, excepto en Android y Cygwin. Ahora es posible que un intérprete Python enlazado estáticamente cargue una extensión de C creada con una biblioteca dinámica de Python. (Contribución de Victor Stinner en bpo-21536.)

En Unix, cuando Python se compila en modo de depuración, la importación ahora también busca extensiones C compiladas en modo de lanzamiento y extensiones C compiladas con la ABI estable. (Contribución de Victor Stinner en bpo-36722.)

Para integrar Python en una aplicación, se debe pasar una nueva opción `--embed` a `python3-config --libs --embed` para obtener `-lpython3.8` (enlaza la aplicación a libpython). Para ser compatible con 3.8 y versiones anteriores, prueba primero con `python3-config --libs --embed` y vuelve a usar `python3-config --libs` (sin `--embed`) si falla el comando anterior.

Agrega un módulo pkg-config `python-3.8-embed` para integrar Python en una aplicación: `pkg-config python-3.8-embed --libs` incluye `-lpython3.8`. Para que sea compatible con 3.8 y versiones anteriores, primero intenta usar `pkg-config python-X.Y-embed --libs` y vuelve a `pkg-config python-X.Y --libs` (sin `--embed`) si el comando anterior falla (reemplaza `XY` con la versión de Python).

Por otro lado, `pkg-config python3.8 --libs` ya no contiene `-lpython3.8`. Las extensiones en C no deben estar enlazadas a libpython (excepto en Android y Cygwin, cuyos casos son manejados por el script); este cambio no es retrocompatible a propósito. (Contribución de Victor Stinner en bpo-36721.)

2.5 Los f-strings soportan = para expresiones autodocumentadas y depuración

Se ha agregado un especificador = a los f-strings. Un f-string como `f'{expr=}'` se expandirá al texto de la expresión, seguido de un signo igual y luego la representación de la expresión evaluada. Por ejemplo:

```
>>> user = 'eric_idle'
>>> member_since = date(1975, 7, 31)
>>> f'{user=} {member_since=}'
"user='eric_idle' member_since=datetime.date(1975, 7, 31)"
```

Los especificadores de formato de los f-string usuales permiten un mayor control sobre como se muestra el resultado de la expresión:

```
>>> delta = date.today() - member_since
>>> f'{user=!s}  {delta.days=:,d}'
'user=eric_idle  delta.days=16,075'
```

El especificador = mostrará la expresión completa para que se puedan mostrar los cálculos:

```
>>> print(f'{theta=}  {cos(radians(theta))=:.3f}')
theta=30  cos(radians(theta))=0.866
```

(Contribución de Eric V. Smith y Larry Hastings en bpo-36817.)

2.6 PEP 578: Ganchos de auditoría en tiempo de ejecución de Python

Este PEP agrega un gancho de auditoría y un gancho abierto verificado. Ambos están disponibles desde Python y desde el código nativo, lo que permite que las aplicaciones y los frameworks escritos en código Python puro aprovechen las notificaciones adicionales, al tiempo que permiten a los integradores o administradores de sistemas implementar compilaciones de Python donde la auditoría siempre está habilitada.

Consultar **PEP 578** para obtener más detalles.

2.7 PEP 587: Configuración de inicialización de Python

El **PEP 587** agrega una nueva API de C para configurar la inicialización de Python, proporcionando un control más preciso de toda la configuración y mejores informes de errores.

Nuevas estructuras:

- `PyConfig`
- `PyPreConfig`
- `PyStatus`
- `PyWideStringList`

Nuevas funciones:

- `PyConfig_Clear()`
- `PyConfig_InitIsolatedConfig()`
- `PyConfig_InitPythonConfig()`
- `PyConfig_Read()`
- `PyConfig_SetArgv()`
- `PyConfig_SetBytesArgv()`
- `PyConfig_SetBytesString()`
- `PyConfig_SetString()`
- `PyPreConfig_InitIsolatedConfig()`
- `PyPreConfig_InitPythonConfig()`
- `PyStatus_Error()`
- `PyStatus_Exception()`
- `PyStatus_Exit()`
- `PyStatus_IsError()`
- `PyStatus_IsExit()`
- `PyStatus_NoMemory()`
- `PyStatus_Ok()`
- `PyWideStringList_Append()`
- `PyWideStringList_Insert()`
- `Py_BytesMain()`
- `Py_ExitStatusException()`
- `Py_InitializeFromConfig()`
- `Py_PreInitialize()`
- `Py_PreInitializeFromArgs()`
- `Py_PreInitializeFromBytesArgs()`
- `Py_RunMain()`

Este PEP también agrega los campos `_PyRuntimeState.preconfig` (tipo `PyPreConfig`) y `PyInterpreterState.config` (tipo `PyConfig`) a estas estructuras internas. `PyInterpreterState.config` se convierte en la nueva configuración de referencia, reemplazando las variables de configuración globales y otras variables privadas.

Consultar Configuración de inicialización de Python para la documentación.

Consultar **PEP 587** para obtener una descripción completa.

(Contribución de Victor Stinner en bpo-36763.)

2.8 Vectorcall: un protocolo de invocación rápida para CPython

El protocolo «vectorcall» se ha agregado a la API de Python/C. Tiene como objetivo formalizar las optimizaciones existentes que ya se realizaron para varias clases. Cualquier tipo de extensión que implemente un invocable puede utilizar este protocolo.

Actualmente es provisional. El objetivo es hacerlo completamente público en Python 3.9.

Consultar **PEP 590** para obtener una descripción completa.

(Contribución de Jeroen Demeyer y Mark Shannon en bpo-36974.)

2.9 Protocolo 5 de Pickle con búferes de datos fuera de banda

Cuando `pickle` se usa para transferir grandes cantidades de datos entre procesos de Python, con la finalidad de aprovechar el procesamiento de múltiples núcleos o máquinas, es importante optimizar la transferencia reduciendo las copias en memoria y posiblemente aplicando técnicas personalizadas, como la compresión dependiente de datos.

El protocolo 5 de `pickle` introduce soporte para búferes fuera de banda, donde los datos compatibles con **PEP 3118** se pueden transmitir separados del flujo principal de pickle, a discreción de la capa de comunicación.

Consultar **PEP 574** para obtener una descripción completa.

(Contribución de Antoine Pitrou en bpo-36785.)

3 Otros cambios en el lenguaje

- La declaración `continue` era ilegal en la cláusula `finally` debido a un problema con la implementación. En Python 3.8 se ha eliminado esta restricción. (Contribución de Serhiy Storchaka en bpo-32489.)
- Los tipos `bool`, `int` y `fractions.Fraction` ahora tienen un método `as_integer_ratio()` como el que se encuentra en `float` y `decimal.Decimal`. Esta extensión menor de la API hace posible escribir `numerator, denominator = x.as_integer_ratio()` y hacer que funcione con múltiples tipos numéricos. (Contribución de Lisa Roach en bpo-33073 y de Raymond Hettinger en bpo-37819.)
- Los constructores de `int`, `float` y `complex` ahora usarán el método especial `__index__()`, si está disponible y el método correspondiente `__int__()`, `__float__()` o `__complex__()` no está disponible. (Contribución de Serhiy Storchaka en bpo-20092.)
- Agregado soporte para escapes `\N{name}` en `expresiones regulares`:

```
>>> notice = 'Copyright © 2019'
>>> copyright_year_pattern = re.compile(r'\N{copyright sign}\s*(\d{4})')
>>> int(copyright_year_pattern.search(notice).group(1))
2019
```

(Contribución de Jonathan Eunice y Serhiy Storchaka in bpo-30688.)

- Los diccionarios y sus vistas ahora se pueden iterar en orden inverso de inserción usando `reversed()`. (Contribución de Rémi Lapeyre en bpo-33462.)
- La sintaxis permitida para los nombres por palabra clave en las llamadas a funciones se ha restringido aún más. En particular, `f((keyword)=arg)` ya no está permitido. Nunca se tuvo intención de permitir algo más que un simple nombre en el lado izquierdo de un término de asignación de argumento por palabra clave. (Contribución de Benjamin Peterson en bpo-34641.)
- El desempaquetado de iterables generalizado en declaraciones `yield` y `return` ya no requiere ser encerrado entre paréntesis. Esto hace que la sintaxis de *yield* y *return* se asemeje más a la sintaxis normal de asignación:

```
>>> def parse(family):
        lastname, *members = family.split()
        return lastname.upper(), *members

>>> parse('simpsons homer marge bart lisa sally')
('SIMPSONS', 'homer', 'marge', 'bart', 'lisa', 'sally')
```

(Contribución de David Cuthbert y Jordan Chapman en bpo-32117.)

- Cuando falta una coma en el código, como en `[(10, 20) (30, 40)]`, el compilador muestra un `SyntaxWarning` con una sugerencia útil. Esto representa una mejora con respecto a la implementación previa en la que solo se mostraba un `TypeError` indicando que la primera tupla no era invocable. (Contribución de Serhiy Storchaka en bpo-15248.)
- Las operaciones aritméticas entre subclases de `datetime.date` o `datetime.datetime` y `datetime.timedelta` ahora retornan una instancia de la subclase, en lugar de la clase base. Esto también afecta al tipo de retorno de las operaciones cuya implementación (directa o indirectamente) usa la aritmética de `datetime.timedelta`, como `astimezone()`. (Contribución de Paul Ganssle en bpo-32417.)
- Cuando el intérprete de Python es interrumpido por Ctrl-C (SIGINT) y la excepción `KeyboardInterrupt` resultante no se detecta, el proceso de Python ahora termina su ejecución a través de una señal SIGINT o con el código de salida correcto, de modo que el proceso que lo invocó puede detectar que murió debido a Ctrl-C. Los shells en POSIX y Windows usan esto para terminar la ejecución de los scripts en sesiones interactivas de forma correcta. (Contribución de Google a través de Gregory P. Smith en bpo-1054041.)
- Algunos estilos de programación avanzados requieren actualizar el objeto `types.CodeType` para una función ya existente. Dado que los objetos de código son inmutables, es necesario crear un nuevo objeto de código, que sea modelado a partir del objeto de código existente. Con 19 parámetros, esto era algo tedioso. Ahora, el nuevo método `replace()` hace posible crear un clon con algunos parámetros alterados.

  Aquí hay un ejemplo que modifica la función `statistics.mean()` para evitar que el parámetro *data* se use como un argumento por palabra clave:

```
>>> from statistics import mean
>>> mean(data=[10, 20, 90])
40
>>> mean.__code__ = mean.__code__.replace(co_posonlyargcount=1)
>>> mean(data=[10, 20, 90])
Traceback (most recent call last):
  ...
TypeError: mean() got some positional-only arguments passed as keyword arguments:
↪'data'
```

(Contribución de Victor Stinner en bpo-37032.)

- Para enteros, la forma de tres argumentos de la función `pow()` ahora permite que el exponente sea negativo en el caso de que la base y el módulo sean primos relativos (coprimos). Luego calcula un inverso modular a la base

cuando el exponente es `-1` y una potencia adecuada de ese inverso en el caso de otros exponentes negativos. Por ejemplo, para calcular el inverso multiplicativo de 38 módulo 137, escribe:

```
>>> pow(38, -1, 137)
119
>>> 119 * 38 % 137
1
```

Los inversos modulares surgen de la solución de ecuaciones diofánticas lineales. Por ejemplo, para encontrar soluciones enteras para $4258x + 147y = 369$, primero debes reescribirla como $4258x \equiv 369 \pmod{147}$ y luego resolver:

```
>>> x = 369 * pow(4258, -1, 147) % 147
>>> y = (4258 * x - 369) // -147
>>> 4258 * x + 147 * y
369
```

(Contribución de Mark Dickinson en bpo-36027.)

- Las compresiones de diccionarios se han sincronizado con los literales de diccionario para que primero se calcule la clave y posteriormente el valor:

```
>>> # Dict comprehension
>>> cast = {input('role? '): input('actor? ') for i in range(2)}
role? King Arthur
actor? Chapman
role? Black Knight
actor? Cleese

>>> # Dict literal
>>> cast = {input('role? '): input('actor? ')}
role? Sir Robin
actor? Eric Idle
```

Este orden de ejecución garantizado es especialmente útil en las expresiones de asignación porque las variables asignadas en la expresión de la clave estarán disponibles en la expresión del valor:

```
>>> names = ['Martin von Löwis', 'Łukasz Langa', 'Walter Dörwald']
>>> {(n := normalize('NFC', name)).casefold() : n for name in names}
{'martin von löwis': 'Martin von Löwis',
 'łukasz langa': 'Łukasz Langa',
 'walter dörwald': 'Walter Dörwald'}
```

(Contribución de Jörn Heissler en bpo-35224.)

- El método `object.__reduce__()` ahora puede retornar una tupla con una longitud que va desde los dos a los seis elementos. Anteriormente, el límite era cinco. El nuevo sexto elemento opcional es un invocable con una firma `(obj, state)`. Esto permite el control directo sobre el comportamiento de actualización de estado de un objeto específico. Si no es *None*, este invocable tendrá prioridad sobre el método `__setstate__()` del objeto. (Contribución de Pierre Glaser y Olivier Grisel en bpo-35900.)

4 Nuevos módulos

- El nuevo módulo `importlib.metadata` proporciona soporte (provisional) para leer metadatos de paquetes de terceros. Por ejemplo, puede extraer el número de versión de un paquete instalado, la lista de puntos de entrada y más:

```
>>> # Note following example requires that the popular "requests"
>>> # package has been installed.
>>>
>>> from importlib.metadata import version, requires, files
>>> version('requests')
'2.22.0'
>>> list(requires('requests'))
['chardet (<3.1.0,>=3.0.2)']
>>> list(files('requests'))[:5]
[PackagePath('requests-2.22.0.dist-info/INSTALLER'),
 PackagePath('requests-2.22.0.dist-info/LICENSE'),
 PackagePath('requests-2.22.0.dist-info/METADATA'),
 PackagePath('requests-2.22.0.dist-info/RECORD'),
 PackagePath('requests-2.22.0.dist-info/WHEEL')]
```

(Contribución de Barry Warsaw y Jason R. Coombs en bpo-34632.)

5 Módulos mejorados

5.1 ast

Los nodos de AST ahora disponen de los atributos `end_lineno` y `end_col_offset`, que proporcionan la localización precisa del final del nodo. (Esto solo se aplica a los nodos que tienen los atributos `lineno` y `col_offset`.)

La nueva función `ast.get_source_segment()` retorna el código fuente de un nodo AST específico.

(Contribución de Ivan Levkivskyi en bpo-33416.)

La función `ast.parse()` tiene algunos flags nuevos:

- `type_comments=True` causa que la función retorne el texto de los comentarios de tipo especificados en **PEP 484** y **PEP 526** asociados con ciertos nodos AST;
- `mode='func_type'` puede usarse para realizar un análisis sintáctico de los «comentarios de tipo de firma» de **PEP 484** (retornados por los nodos AST de definición de funciones);
- `feature_version=(3, N)` permite especificar una versión de Python 3 previa. Por ejemplo, `feature_version=(3, 4)` hará que se trate a `async` y `await` como palabras no reservadas.

(Contribución de Guido van Rossum en bpo-35766.)

5.2 asyncio

`asyncio.run()` ha pasado de la API provisional a la estable. Esta función se puede utilizar para ejecutar una coroutine y retornar el resultado mientras se gestiona automáticamente el bucle de eventos. Por ejemplo:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

asyncio.run(main())
```

Esto es *aproximadamente* equivalente a:

```
import asyncio

async def main():
    await asyncio.sleep(0)
    return 42

loop = asyncio.new_event_loop()
asyncio.set_event_loop(loop)
try:
    loop.run_until_complete(main())
finally:
    asyncio.set_event_loop(None)
    loop.close()
```

La implementación real es significativamente más compleja. Por lo tanto, `asyncio.run()` debería ser la forma preferida de ejecutar programas asyncio.

(Contribución de Yury Selivanov en bpo-32314.)

La ejecución de `python -m asyncio` lanza un REPL asincrónico de forma nativa. Esto permite una rápida experimentación con código que tiene un nivel `await` superior. Ya no es necesario llamar directamente a `asyncio.run()`, lo que generaría un nuevo ciclo de eventos en cada invocación:

```
$ python -m asyncio
asyncio REPL 3.8.0
Use "await" directly instead of "asyncio.run()".
Type "help", "copyright", "credits" or "license" for more information.
>>> import asyncio
>>> await asyncio.sleep(10, result='hello')
hello
```

(Contribución de Yury Selivanov en bpo-37028.)

The exception `asyncio.CancelledError` now inherits from `BaseException` rather than `Exception` and no longer inherits from `concurrent.futures.CancelledError`. (Contributed by Yury Selivanov in bpo-32528.)

En Windows, el ciclo de eventos predeterminado ahora es `ProactorEventLoop`. (Contribución de Victor Stinner en bpo-34687.)

`ProactorEventLoop` ahora también es compatible con UDP. (Contribución de Adam Meily y Andrew Svetlov en bpo-29883.)

`ProactorEventLoop` ahora puede ser interrumpido por `KeyboardInterrupt` («CTRL+C»). (Contribución de Vladimir Matveev en bpo-23057.)

Se ha agregado `asyncio.Task.get_coro()` para obtener la corrutina envuelta dentro de `asyncio.Task`. (Contribución de Alex Grönholm en bpo-36999.)

Las tareas de Asyncio ahora se pueden nombrar, ya sea pasando el argumento por palabra clave `name` a `asyncio.create_task()` o al método `create_task()` del bucle de eventos, o invocando al método `set_name()` en el objeto de tarea. El nombre de la tarea es visible en la salida de `repr()` de `asyncio.Task` y también se puede recuperar usando el método `get_name()`. (Contribución de Alex Grönholm en bpo-34270.)

Se agregó soporte para Happy Eyeballs a `asyncio.loop.create_connection()`. Para especificar el comportamiento, se han agregado dos nuevos parámetros: *happy_eyeballs_delay* e *interleave*. El algoritmo Happy Eyeballs mejora la capacidad de respuesta en aplicaciones que admiten IPv4 e IPv6 al intentar conectarse simultáneamente utilizando ambos. (Contribución de twisteroid ambassador en bpo-33530.)

5.3 builtins

La función incorporada `compile()` se ha mejorado para que acepte el flag `ast.PyCF_ALLOW_TOP_LEVEL_AWAIT`. Si se pasa este nuevo flag, `compile()` permitirá construcciones de nivel superior `await`, `async for` y `async with`, que normalmente se consideran sintaxis inválida. El objeto de código asíncrono marcado con el flag `CO_COROUTINE` puede ser retornado. (Contribución de Matthias Bussonnier en bpo-34616.)

5.4 collections

El método `_asdict()` para `collections.namedtuple()` ahora retorna una instancia de `dict` en lugar de una de `collections.OrderedDict`. Esto funciona porque se garantiza que los diccionarios regulares mantienen el orden de inserción desde Python 3.7. Si se requieren las características adicionales de `OrderedDict`, la solución sugerida es realizar una conversión del resultado al tipo deseado: `OrderedDict(nt._asdict())`. (Contribución de Raymond Hettinger en bpo-35864.)

5.5 cProfile

La clase `cProfile.Profile` ahora puede usarse como gestor de contexto. Ahora se puede perfilar un bloque de código ejecutando:

```
import cProfile

with cProfile.Profile() as profiler:
      # code to be profiled
      ...
```

(Contribución de Scott Sanderson en bpo-29235.)

5.6 csv

`csv.DictReader` ahora retorna instancias de `dict` en lugar de `collections.OrderedDict`. La herramienta ahora es más rápida y usa menos memoria, mientras conserva el orden de los campos. (Contribución de Michael Selik en bpo-34003.)

5.7 curses

Se agregó una nueva variable que contiene información de versión estructurada para la biblioteca ncurses subyacente: `ncurses_version`. (Contribución de Serhiy Storchaka en bpo-31680.)

5.8 ctypes

En Windows, `CDLL` y sus subclases ahora aceptan un parámetro *winmode* para especificar flags para la invocación subyacente de `LoadLibraryEx`. Los flags predeterminados están establecidos para cargar solo las dependencias de DLL desde ubicaciones confiables, incluida la ruta donde se almacena la DLL (si se usa una ruta completa o parcial para cargar la DLL inicial) y las rutas agregadas por `add_dll_directory()` . (Contribución de Steve Dower en bpo-36085.)

5.9 datetime

Se agregaron nuevos constructores alternativos `datetime.date.fromisocalendar()` y `datetime.datetime.fromisocalendar()`, que construyen objetos `date` y `datetime` respectivamente con el año, número de semana y día de la semana de la fecha del calendario ISO. Estos son el inverso del método `isocalendar` de cada clase. (Contribución de Paul Ganssle en bpo-36004.)

5.10 functools

`functools.lru_cache()` ahora se puede usar como un decorador directo en lugar de como una función que retorna un decorador. De forma que ambos son compatibles ahora:

```
@lru_cache
def f(x):
    ...

@lru_cache(maxsize=256)
def f(x):
    ...
```

(Contribución de Raymond Hettinger en bpo-36772.)

Se ha agregado un nuevo decorador `functools.cached_property()`, para propiedades calculadas almacenadas en caché durante toda la vida útil de la instancia.

```
import functools
import statistics

class Dataset:
   def __init__(self, sequence_of_numbers):
      self.data = sequence_of_numbers

   @functools.cached_property
   def variance(self):
      return statistics.variance(self.data)
```

(Contribución de Carl Meyer en bpo-21145.)

Se ha agregado un nuevo decorador `functools.singledispatchmethod()` que convierte métodos en funciones genéricas usando single dispatch:

```
from functools import singledispatchmethod
from contextlib import suppress

class TaskManager:

    def __init__(self, tasks):
        self.tasks = list(tasks)

    @singledispatchmethod
    def discard(self, value):
        with suppress(ValueError):
            self.tasks.remove(value)

    @discard.register(list)
    def _(self, tasks):
        targets = set(tasks)
        self.tasks = [x for x in self.tasks if x not in targets]
```

(Contribución de Ethan Smith en bpo-32380)

5.11 gc

`get_objects()` ahora puede recibir un parámetro opcional *generation* que indica la generación de la que recolectar objetos. (Contribución de Pablo Galindo en bpo-36016.)

5.12 gettext

Agregado `pgettext()` y sus variantes. (Contribución de Franz Glasner, Éric Araujo y Cheryl Sabella en bpo-2504.)

5.13 gzip

Se ha agregó el parámetro *mtime* a `gzip.compress()` para una salida reproducible. (Contribución de Guo Ci Teo en bpo-34898.)

Una excepción `BadGzipFile` es lanzada ahora, en lugar de `OSError`, para ciertos tipos de archivos gzip no válidos o corruptos. (Contribución de Filip Gruszczyński, Michele Orrù y Zackery Spytz en bpo-6584.)

5.14 IDLE e idlelib

Las salidas superiores a N líneas (50 por defecto) se pliegan en un botón. N se puede cambiar en la sección PyShell de la página General del cuadro de diálogo Settings. Se pueden plegar menos líneas, pero posiblemente más largas, haciendo clic derecho en la salida. La salida plegada se puede expandir en su lugar haciendo doble clic en el botón o en el portapapeles o en una ventana separada haciendo clic derecho en el botón. (Contribución de Tal Einat en bpo-1529353.)

Se ha agregado «Run Customized» al menú Run para ejecutar un módulo con configuraciones personalizadas. Cualquier argumento de la línea de comandos ingresado se agrega a sys.argv. Además, vuelven a aparecer en el cuadro para la próxima ejecución personalizada. También se puede suprimir el reinicio normal del módulo principal de la Shell. (Contribución de Cheryl Sabella, Terry Jan Reedy y otros en bpo-5680 y bpo-37627.)

Se agregaron números de línea opcionales para las ventanas del editor IDLE. Las ventanas se abren sin números de línea, a menos que se establezca lo contrario en la pestaña General del cuadro de diálogo de configuración. Los números de línea

de una ventana existente se muestran y ocultan en el menú Options. (Contribución de Tal Einat y Saimadhav Heblikar en bpo-17535.)

La codificación nativa del sistema operativo ahora se usa para convertir entre cadenas de Python y objetos Tcl. Esto permite que el IDLE funcione con emoji y otros caracteres que no son BMP. Estos caracteres se pueden mostrar o copiar y pegar en ,o desde, el portapapeles. Convertir cadenas de Tcl a Python y viceversa ahora nunca falla. (Mucha gente trabajó en esto durante ocho años, pero el problema finalmente lo resolvió Serhiy Storchaka en bpo-13153.)

New in 3.8.1:

Add option to toggle cursor blink off. (Contributed by Zackery Spytz in bpo-4603.)

Escape key now closes IDLE completion windows. (Contributed by Johnny Najera in bpo-38944.)

Los cambios anteriores se han portado a las versiones de mantenimiento de Python 3.7.

Add keywords to module name completion list. (Contributed by Terry J. Reedy in bpo-37765.)

5.15 inspect

La función `inspect.getdoc()` puede ahora encontrar cadenas de documentación para `__slots__` si el este atributo es un `dict` cuyos valores son las cadenas de documentación. Esto proporciona opciones de documentación similares a las que ya tenemos para `property()`, `classmethod()` y `staticmethod()`:

```
class AudioClip:
    __slots__ = {'bit_rate': 'expressed in kilohertz to one decimal place',
                 'duration': 'in seconds, rounded up to an integer'}
    def __init__(self, bit_rate, duration):
        self.bit_rate = round(bit_rate / 1000.0, 1)
        self.duration = ceil(duration)
```

(Contribución de Raymond Hettinger en bpo-36326.)

5.16 io

En el modo de desarrollo (`-X env`) y en la compilación de depuración, el finalizador `io.IOBase` ahora registra la excepción si falla el método `close()`. La excepción se ignora silenciosamente de forma predeterminada en la compilación de lanzamiento. (Contribución de Victor Stinner en bpo-18748.)

5.17 itertools

Se ha agregado un argumento por palabra clave opcional *initial* a la función `itertools.accumulate()` para permitir especificar un valor inicial:

```
>>> from itertools import accumulate
>>> list(accumulate([10, 5, 30, 15], initial=1000))
[1000, 1010, 1015, 1045, 1060]
```

(Contribución de Lisa Roach en bpo-34659.)

5.18 json.tool

Agregadas las opciones `--json-lines` para analizar sintácticamente cada línea de entrada como un objeto JSON separado. (Contribución de Weipeng Hong en bpo-31553.)

5.19 logging

Agregado un argumento por palabra clave *force* a la función `logging.basicConfig()`. Cuando se establece en verdadero, cualquier controlador existente adjunto al registrador (logger) raíz se elimina y se cierra antes de realizar la configuración especificada por los otros argumentos.

Esto resuelve un problema de larga data. Una vez que se había invocado a un registrador o a *basicConfig()*, las invocaciones posteriores a *basicConfig()* se ignoraban en silencio. Esto dificultaba la actualización, la experimentación o la instrucción de las diversas opciones de configuración de registro mediante el interprete interactivo o el bloc de notas de Jupyter.

(Sugerencia de Raymond Hettinger, implementación de Dong-hee Na y revisión de Vinay Sajip en bpo-33897.)

5.20 math

Se ha agregado la nueva función `math.dist()` para calcular la distancia euclidiana entre dos puntos. (Contribución de Raymond Hettinger en bpo-33089.)

Se ha expandido la función `math.hypot()` para manejar múltiples dimensiones. Anteriormente, solo admitía dos dimensiones. (Contribución de Raymond Hettinger en bpo-33089.)

Agregada una nueva función, `math.prod()`, como función análoga a `sum()`, que retorna el producto de todos los elementos de un iterable de números partiendo de un valor de inicio (*start*) (por defecto: 1):

```
>>> prior = 0.8
>>> likelihoods = [0.625, 0.84, 0.30]
>>> math.prod(likelihoods, start=prior)
0.126
```

(Contribución de Pablo Galindo en bpo-35606.)

Agregadas dos nuevas funciones combinatorias, `math.perm()` y `math.comb()`:

```
>>> math.perm(10, 3)     # Permutations of 10 things taken 3 at a time
720
>>> math.comb(10, 3)     # Combinations of 10 things taken 3 at a time
120
```

(Contribución de Yash Aggarwal, Keller Fuchs, Serhiy Storchaka y Raymond Hettinger en bpo-37128, bpo-37178 y bpo-35431.)

Se ha agregada una nueva función `math.isqrt()` para calcular raíces cuadradas enteras precisas sin conversión a coma flotante. La nueva función admite números enteros arbitrariamente grandes. Es más rápida que `floor(sqrt(n))` pero más lenta que `math.sqrt()`:

```
>>> r = 650320427
>>> s = r ** 2
>>> isqrt(s - 1)         # correct
650320426
>>> floor(sqrt(s - 1))   # incorrect
650320427
```

(Contribución de Mark Dickinson en bpo-36887.)

La función `math.factorial()` ya no acepta argumentos que no sean similares a enteros. (Contribución de Pablo Galindo en bpo-33083.)

5.21 mmap

La clase `mmap.mmap` ahora tiene un método `madvise()` para acceder a la llamada al sistema `madvise()`. (Contribución de Zackery Spytz en bpo-32941.)

5.22 multiprocessing

Agregado el nuevo módulo `multiprocessing.shared_memory`. (Contribución de Davin Potts en bpo-35813.)

En macOS, el método de inicio *spawn* se usa ahora por defecto. (Contribución de Victor Stinner en bpo-33725.)

5.23 os

Se agregó una nueva función `add_dll_directory()` en Windows para proporcionar rutas de búsqueda adicionales para las dependencias nativas al importar módulos de extensión o al cargar archivos DLL utilizando `ctypes`. (Contribución de Steve Dower en bpo-36085.)

Se agregó una nueva función `os.memfd_create()` para envolver la llamada al sistema `memfd_create()`. (Contribución de Zackery Spytz y Christian Heimes en bpo-26836.)

En Windows, gran parte de la lógica manual para manejar los puntos de reinterpretación (incluidos los enlaces simbólicos y las uniones de directorios) se ha delegado al sistema operativo. Específicamente, `os.stat()` ahora se encargará de todo lo que sea compatible con el sistema operativo, mientras que `os.lstat()` solo abrirá puntos de reinterpretación que se identifican como «sustitutos de nombre» y el resto se abrirán mediante `os.stat()`. En todos los casos, `stat_result.st_mode` solo tendrá establecido `S_IFLNK` para enlaces simbólicos y no para otros tipos de puntos de reinterpretación. Para identificar otros tipos de puntos de reinterpretación, verifica el nuevo atributo `stat_result.st_reparse_tag`.

En Windows, `os.readlink()` ahora puede leer uniones de directorio. Ten en cuenta que `islink()` retornará `False` para las uniones de directorios, por lo que el código que comprueba en primer lugar `islink` continuará tratando las uniones como directorios, mientras que el código que maneja los errores de `os.readlink()` ahora puede tratar las uniones como enlaces.

(Contribución de Steve Dower en bpo-37834.)

As of 3.8.20, `os.mkdir()` and `os.makedirs()` on Windows now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects `tempfile.mkdtemp()` and is a mitigation for CVE-2024-4030. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in gh-118486.)

5.24 os.path

Las funciones de `os.path` que retornan un resultado booleano como `exists()`, `lexists()`, `isdir()`, `isfile()`, `islink()` e `ismount()` ahora retornan `False` en lugar de lanzar una excepción `ValueError`, o sus subclases `UnicodeEncodeError` y `UnicodeDecodeError`, para rutas que contienen caracteres o bytes irrepresentables a nivel del sistema operativo. (Contribución de Serhiy Storchaka en bpo-33721.)

`expanduser()` en Windows ahora usa preferentemente la variable de entorno `USERPROFILE` y no usa `HOME`, que normalmente no está establecido para cuentas de usuario normales. (Contribución de Anthony Sottile en bpo-36264.)

`isdir()` en Windows ya no retorna `True` para un enlace a un directorio no existente.

`realpath()` en Windows ahora resuelve puntos de reinterpretación (reparse points), incluidos enlaces simbólicos y uniones de directorio.

(Contribución de Steve Dower en bpo-37834.)

5.25 pathlib

Los métodos del módulo `pathlib.Path` que retornan un resultado booleano, como `exists()`, `is_dir()`, `is_file()`, `is_mount()`, `is_symlink()`, `is_block_device()`, `is_char_device()`, `is_fifo()` o `is_socket()`, ahora retornan `False` en vez de lanzar una excepción `ValueError` o su subclase `UnicodeEncodeError` para rutas que contienen caracteres no representables a nivel del sistema operativo. (Contribución de Serhiy Storchaka en bpo-33721.)

Agregado `pathlib.Path.link_to()` que crea un enlace duro apuntando a una ruta. (Contribución de Joannah Nanjekye en bpo-26978)

5.26 pickle

Las extensiones de `pickle` que subclasifican la clase optimizada de C `Pickler` ahora pueden anular la lógica de pickling de funciones y clases definiendo el método especial `reducer_override()`. (Contribución de Pierre Glaser y Olivier Grisel en bpo-35900.)

5.27 plistlib

Se ha agregado la nueva clase `plistlib.UID` y se ha habilitado el soporte para leer y escribir plists binarios codificados por NSKeyedArchiver. (Contribución de Jon Janzen en bpo-26707.)

5.28 pprint

Se ha agregado el parámetro *sort_dicts* a varias funciones del módulo `pprint`. De forma predeterminada, esas funciones continúan ordenando los diccionarios antes de procesarlos o imprimirlos. Sin embargo, si *sort_dicts* se establece en falso, los diccionarios conservan el orden en que se insertaron las claves. Esto puede resultar útil para la comparación con entradas JSON durante la depuración.

Además, hay una nueva función de conveniencia, `pprint.pp()`, que es igual que `pprint.pprint()` pero con *sort_dicts* establecido en `False` por defecto:

```
>>> from pprint import pprint, pp
>>> d = dict(source='input.txt', operation='filter', destination='output.txt')
>>> pp(d, width=40)                  # Original order
{'source': 'input.txt',
 'operation': 'filter',
 'destination': 'output.txt'}
>>> pprint(d, width=40)              # Keys sorted alphabetically
{'destination': 'output.txt',
 'operation': 'filter',
 'source': 'input.txt'}
```

(Contribución de Rémi Lapeyre en bpo-30670.)

5.29 py_compile

`py_compile.compile()` ahora admite el modo silencioso. (Contribución de Joannah Nanjekye en bpo-22640.)

5.30 shlex

La nueva función `shlex.join()` actúa a la inversa de `shlex.split()`. (Contribución de Bo Bayles en bpo-32102.)

5.31 shutil

`shutil.copytree()` ahora acepta el nuevo argumento por palabra clave `dirs_exist_ok`. (Contribución de Josh Bronson en bpo-20849.)

`shutil.make_archive()` ahora usa por defecto el formato pax moderno (POSIX.1-2001) para nuevos archivos para mejorar la portabilidad y la conformidad con los estándares, heredado el cambio correspondiente del módulo `tarfile`. (Contribución de C.A.M. Gerlach en bpo-30661.)

`shutil.rmtree()` en Windows ahora elimina las uniones de directorio sin eliminar recursivamente su contenido primero. (Contribución de Steve Dower en bpo-37834.)

5.32 socket

Se han agregado las funciones de conveniencia `create_server()` y `has_dualstack_ipv6()` para automatizar las tareas necesarias involucradas al crear un socket servidor, incluida la aceptación de conexiones IPv4 e IPv6 en el mismo socket . (Contribución de Giampaolo Rodolà en bpo-17561.)

Las funciones `socket.if_nameindex()`, `socket.if_nametoindex()` y `socket.if_indextoname()` se han implementado en Windows. (Contribución de Zackery Spytz en bpo-37007.)

5.33 ssl

Se ha agregado `post_handshake_auth` para habilitar y `verify_client_post_handshake()` para iniciar la autenticación tras el establecimiento de la comunicación en TLS 1.3. (Contribución de Christian Heimes en bpo-34670.)

5.34 statistics

Se ha agregado `statistics.fmean()` como una variante de punto flotante más rápida de `statistics.mean()`. (Contribución de Raymond Hettinger y Steven D'Aprano en bpo-35904.)

Se ha agregado `statistics.geometric_mean()` (Contribución de Raymond Hettinger en bpo-27181.)

Se ha agregado `statistics.multimode()` que retorna una lista con los valores más comunes. (Contribución de Raymond Hettinger en bpo-35892.)

Se ha agregado `statistics.quantiles()` que divide datos o una distribución en intervalos equiprobables (por ejemplo, cuartiles, deciles o percentiles). (Contribución de Raymond Hettinger en bpo-36546.)

Se ha agregado `statistics.NormalDist`, una herramienta para crear y manipular distribuciones normales de una variable aleatoria. (Contribución de Raymond Hettinger en bpo-36018.)

```
>>> temperature_feb = NormalDist.from_samples([4, 12, -3, 2, 7, 14])
>>> temperature_feb.mean
6.0
>>> temperature_feb.stdev
6.356099432828281

>>> temperature_feb.cdf(3)           # Chance of being under 3 degrees
0.3184678262814532
>>> # Relative chance of being 7 degrees versus 10 degrees
>>> temperature_feb.pdf(7) / temperature_feb.pdf(10)
1.2039930378537762

>>> el_niño = NormalDist(4, 2.5)
>>> temperature_feb += el_niño       # Add in a climate effect
>>> temperature_feb
NormalDist(mu=10.0, sigma=6.830080526611674)

>>> temperature_feb * (9/5) + 32     # Convert to Fahrenheit
NormalDist(mu=50.0, sigma=12.294144947901014)
>>> temperature_feb.samples(3)       # Generate random samples
[7.672102882379219, 12.000027119750287, 4.647488369766392]
```

5.35 sys

Se ha agregado la nueva función `sys.unraisablehook()` que se puede anular para controlar cómo se manejan las «excepciones no lanzables». Se llama cuando se ha producido una excepción, pero Python no tiene forma de manejarla. Por ejemplo, cuando un destructor genera una excepción o durante la recolección de basura (`gc.collect()`). (Contribución de Victor Stinner en bpo-36829.)

5.36 tarfile

El módulo `tarfile` ahora tiene por defecto el formato pax moderno (POSIX.1-2001) para nuevos archivos, en lugar del anterior específico de GNU. Esto mejora la portabilidad multiplataforma con una codificación consistente (UTF-8) en un formato estandarizado y extensible, y ofrece otros varios beneficios. (Contribución de C.A.M. Gerlach en bpo-36268.)

5.37 tempfile

As of 3.8.20 on Windows, the default mode `0o700` used by `tempfile.mkdtemp()` now limits access to the new directory due to changes to `os.mkdir()`. This is a mitigation for CVE-2024-4030. (Contributed by Steve Dower in gh-118486.)

5.38 threading

Se ha agregado una nueva función `threading.excepthook()` que maneja las excepciones `threading.Thread.run()` no capturadas. Se puede anular para controlar cómo se manejan las excepciones `threading.Thread.run()` no capturadas. (Contribución de Victor Stinner en bpo-1230540.)

Se han agregado una nueva función `threading.get_native_id()` y un atributo `native_id` a la clase `threading.Thread`. Estos retornan el Thread ID nativo integral del hilo actual asignado por el kernel. Esta función solo está disponible en determinadas plataformas, consulta `get_native_id` para obtener más información. (Contribución de Jake Tesler en bpo-36084.)

5.39 tokenize

El módulo `tokenize` ahora emite implícitamente un token `NEWLINE` cuando se le proporciona una entrada sin una nueva línea al final. Este comportamiento ahora coincide con lo que hace internamente el tokenizador de C. (Contribución de Ammar Askar en bpo-33899.)

5.40 tkinter

Se han agregado los métodos `selection_from()`, `selection_present()`, `selection_range()` y `selection_to()` a la clase `tkinter.Spinbox`. (Contribución de Juliette Monsel en bpo-34829.)

Se ha agregado el método `moveto()` a la clase `tkinter.Canvas`. (Contribución de Juliette Monsel en bpo-23831.)

La clase `tkinter.PhotoImage` ahora dispone de los métodos `transparency_get()` y `transparency_set()`. (Contribución de Zackery Spytz en bpo-25451.)

5.41 time

Se ha agregado el nuevo reloj `CLOCK_UPTIME_RAW` para macOS 10.12. (Contribución de Joannah Nanjekye en bpo-35702.)

5.42 typing

Se han incorporado varias características al módulo `typing`:

- Un tipo de diccionario con tipos para cada clave. Consultar **PEP 589** y `typing.TypedDict`. TypedDict usa solo claves de cadenas de caracteres. De forma predeterminada, se requiere que todas las claves estén presentes. Especifica «total=False» para permitir que las claves sean opcionales:

```
class Location(TypedDict, total=False):
    lat_long: tuple
    grid_square: str
    xy_coordinate: tuple
```

- Tipos literales. Consultar **PEP 586** y `typing.Literal`. Los tipos literales indican que un parámetro o valor de retorno está restringido a uno o más valores literales específicos:

```
def get_status(port: int) -> Literal['connected', 'disconnected']:
    ...
```

- Variables, funciones, métodos y clases «finales». Consultar **PEP 591**, `typing.Final` y `typing.final()`. El clasificador final instruye a un validador estático de tipos para restringir la subclasificación, anulación o reasignación:

```
pi: Final[float] = 3.1415926536
```

- Definiciones de protocolo. Consultar **PEP 544**, `typing.Protocol` y `typing.runtime_checkable()`. ABCs simples como `typing.SupportsInt` ahora son subclases de `Protocol` .
- Nueva clase protocolo `typing.SupportsIndex`.
- Nuevas funciones `typing.get_origin()` y `typing.get_args()`.

5.43 unicodedata

El módulo `unicodedata` ha sido actualizado para usar Unicode 12.1.0.

La nueva función `is_normalized()` puede usarse para verificar que una cadena está en una forma normal específica, lo que es a menudo mucho más rápido que normalizar la cadena. (Contribución de Max Belanger, David Euresti y Greg Price en bpo-32285 y bpo-37966).

5.44 unittest

Se ha agregado `AsyncMock` para admitir una versión asincrónica de `Mock`. También se han agregado nuevas funciones de aserción apropiadas para las pruebas. (Contribución de Lisa Roach en bpo-26467).

Se ha agregado `addModuleCleanup()` y `addClassCleanup()` a unittest para admitir limpiezas para `setUpModule()` y `setUpClass()`. (Contribución de Lisa Roach en bpo-24412.)

Varias funciones de aserción simulada ahora también imprimen una lista de llamadas reales en caso de fallo. (Contribución de Petter Strandmark en bpo-35047.)

El módulo `unittest` ha obtenido soporte para corrutinas que se utilizarán como casos de prueba con `unittest.IsolatedAsyncioTestCase`. (Contribución de Andrew Svetlov en bpo-32972.)

Ejemplo:

```
import unittest


class TestRequest(unittest.IsolatedAsyncioTestCase):

    async def asyncSetUp(self):
        self.connection = await AsyncConnection()

    async def test_get(self):
        response = await self.connection.get("https://example.com")
        self.assertEqual(response.status_code, 200)

    async def asyncTearDown(self):
        await self.connection.close()


if __name__ == "__main__":
    unittest.main()
```

5.45 venv

`venv` ahora incluye un script `Activate.ps1` en todas las plataformas para activar entornos virtuales en PowerShell Core 6.1. (Contribución de Brett Cannon en bpo-32718.)

5.46 weakref

Los objetos proxy retornados por `debilref.proxy()` ahora admiten los operadores de multiplicación de matrices `@` y `@=`, además de los otros operadores numéricos. (Contribución de Mark Dickinson en bpo-36669.)

5.47 xml

Como mitigación contra DTD y recuperación de entidades externas, los módulos `xml.dom.minidom` y `xml.sax` ya no procesan entidades externas de forma predeterminada. (Contribución de Christian Heimes en bpo-17239.)

Los métodos `.find*()` del módulo `xml.etree.ElementTree` admiten búsquedas con comodines, como `{*}tag`, que ignora el espacio de nombres, y `{namespace}*`, que retorna todas las etiquetas en el espacio de nombres dado. (Contribución de Stefan Behnel en bpo-28238.)

El módulo `xml.etree.ElementTree` proporciona una nueva función `–xml.etree.ElementTree.canonicalize()` que implementa C14N 2.0. (Contribución de Stefan Behnel en bpo-13611.)

El objeto de destino de `xml.etree.ElementTree.XMLParser` puede recibir eventos de declaración de espacio de nombres a través de los nuevos métodos de retrollamada `start_ns()` y `end_ns()`. Además, el destino `xml.etree.ElementTree.TreeBuilder` se puede configurar para procesar eventos sobre comentarios e instrucciones de procesamiento para incluirlos en el árbol generado. (Contribución de Stefan Behnel en bpo-36676 y bpo-36673.)

5.48 xmlrpc

`xmlrpc.client.ServerProxy` ahora admite un argumento por palabra clave *headers* opcional para una secuencia de encabezados HTTP que se enviarán con cada solicitud. Entre otras cosas, esto permite actualizar desde la autenticación básica predeterminada a una autenticación de sesión más rápida. (Contribución de Cédric Krier en bpo-35153.)

6 Optimizaciones

- El módulo `subprocess` ahora puede usar en ciertos casos la función `os.posix_spawn()` para mejorar el rendimiento. Actualmente, solo se usa en macOS y Linux (usando glibc 2.24 o una versión más reciente) y siempre que se cumplan todas estas condiciones:
  - *close_fds* es falso;
  - los parámetros *preexec_fn*, *pass_fds*, *cwd* y *start_new_session* no están establecidos;
  - la ruta *executable* contiene un directorio.

  (Contribución de Joannah Nanjekye y Victor Stinner en bpo-35537.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` y `shutil.move()` usan llamadas al sistema de «copia-rápida» específicas de la plataforma en Linux y macOS, con la finalidad de copiar ficheros más eficientemente. «copia-rápida» significa que la operación de copiado tiene lugar en el propio kernel, evitando que Python haga uso de búferes en el espacio de usuario como «`outfd.write(infd.read())`». En Windows, `shutil.copyfile()` usa un tamaño de búfer predeterminado más grande (1 MiB en lugar de 16 KiB) y se usa una variante de `shutil.copyfileobj()` basada en `memoryview()`. La aceleración al copiar un archivo de 512 MiB dentro de la misma partición es de aproximadamente +26% en Linux, +50% en macOS y +40% en Windows. Además, se consumen muchos menos ciclos de CPU. Consultar la sección shutil-platform-dependent-efficient-copy-operations. (Contribución de Giampaolo Rodolà en bpo-33671.)
- Ahora `shutil.copytree()` usa la función `os.scandir()` y todas las funciones de copia que dependen de ella usan el valor en caché de `os.stat()`. Al copiar un directorio con 8000 archivos, la mejora de velocidad es de +9% en Linux, +20% en Windows y +30% en recursos compartidos de Windows SMB. Además, el número

de llamadas al sistema de `os.stat()` se reduce en un 38%, lo que hace que `shutil.copytree()` sea más rápida especialmente en sistemas de archivos de red. (Contribución de Giampaolo Rodolà en bpo-33695.)

- El protocolo por defecto del módulo `pickle` es ahora el Protocolo 4, introducido por primera vez en Python 3.4. Ofrece un mejor desempeño y un menor tamaño, en comparación con el Protocolo 3 disponible desde Python 3.0.
- Eliminado un miembro `Py_ssize_t` de `PyGC_Head`. El tamaño de todos los objetos seguidos por el GC (por ejemplo tuple, list y dict) se ha reducido 4 u 8 bytes. (Contribución de Inada Naoki en bpo-33597.)
- La clase `uuid.UUID` ahora usa `__slots__` para reducir su impacto en la memoria. (Contribución de Wouter Bolsterlee y Tal Einat en bpo-30977.)
- Mejorado el rendimiento de `operator.itemgetter()` en un 33%. Se ha optimizado el manejo de argumentos y se ha agregado una ruta rápida al caso común de índices enteros no negativos únicos en tuplas (que constituye un caso de uso común en la biblioteca estándar). (Contribución de Raymond Hettinger en bpo-35664.)
- Se han acelerado las búsquedas de campos en `collections.namedtuple()`. Ahora son más del doble de rápidas, lo que las convierte en la forma más rápida de búsqueda de variables de instancia en Python. (Contribución de Raymond Hettinger, Pablo Galindo, Joe Jevnik y Serhiy Storchaka en bpo-32492.)
- El constructor de `list` no sobre-asignará el búfer del elemento interno si se conoce la longitud de la entrada iterable (es decir, si la entrada implementa `__len__`). Esto hace que la lista generada sea, en promedio, un 12% más pequeña. (Contribución de Raymond Hettinger y Pablo Galindo en bpo-33234.)
- De ha duplicado la velocidad de escritura de variables de clase. Antes, cuando se actualizaba un atributo non-dunder, había una llamada innecesaria para actualizar slots. (Contribución de Stefan Behnel, Pablo Galindo Salgado, Raymond Hettinger, Neil Schemenauer y Serhiy Storchaka en bpo-36012.)
- Se ha reducido la sobrecarga de conversión de argumentos pasados a muchas funciones y métodos integrados. Esto acelera la llamada a algunas funciones y métodos incorporados simples hasta un 20-50%. (Contribución de Serhiy Storchaka en bpo-23867, bpo-35582 y bpo-36127.)
- La instrucción `LOAD_GLOBAL` ahora utiliza el nuevo mecanismo «per opcode cache». Ahora es aproximadamente un 40% más rápida. (Contribución de Yury Selivanov e Inada Naoki en bpo-26219.)

7 Cambios en la compilación y la API de C

- El valor predeterminado para `sys.abiflags` es ahora una cadena vacía: el flag `m` para pymalloc se ha vuelto innecesario (las compilaciones con y sin pymalloc son ABI compatibles) y por lo tanto se ha eliminado. (Contribución de Victor Stinner en bpo-36707.)

  Ejemplos del cambio:

  - Solo el programa `python3.8` es instalado, el programa `python3.8m` se ha eliminado.
  - Solo el script `python3.8-config` es instalado, el script `python3.8m-config` se ha eliminado.
  - El flag `m` se ha eliminado del sufijo de los nombres de archivo de las bibliotecas dinámicas: los módulos de extensión de la biblioteca estándar, así como los producidos e instalados por paquetes de terceros, como los descargados desde PyPI. En Linux, por ejemplo, el sufijo `.cpython-37m-x86_64-linux-gnu.so` en Python 3.7 se ha convertido en `.cpython-38-x86_64-linux-gnu.so` en Python 3.8.
- Los archivos de cabeceras se han reorganizado para separar mejor los diferentes tipos de APIs:
  - `Include/*.h` debe ser la API de C portable, pública y estable.
  - `Include/cpython/*.h` debe ser la API de C inestable específica de CPython. Una API pública, con alguna API privada marcada con los prefijos `_Py` o `_PY`.

  - `Include/internal/*.h` es la API de C interna privada, muy específica de CPython. Esta API no tiene retro-compatibilidad garantizada con versiones anteriores y no debe ser usada fuera de CPython. Solo está expuesta para cubrir necesidades muy específicas, como es el caso de depuradores y perfiladores, que necesitan acceder a los componentes internos de CPython sin llamar directamente a las funciones. Esta API es ahora instalada por `make install`.

  (Contribución de Victor Stinner en bpo-35134 y bpo-35081, trabajo iniciado por Eric Snow en Python 3.7.)

- Algunas macros se han convertido a funciones inline estáticas: los tipos de los parámetros y el tipo de retorno están bien definidos, no entrañan cuestiones que precisen el uso específico de macros y las variables tienen ámbito local. Algunos ejemplos:

  - `Py_INCREF()`, `Py_DECREF()`
  - `Py_XINCREF()`, `Py_XDECREF()`
  - `PyObject_INIT()`, `PyObject_INIT_VAR()`
  - Funciones privadas: `_PyObject_GC_TRACK()`, `_PyObject_GC_UNTRACK()`, `_Py_Dealloc()`

  (Contribución de Victor Stinner en bpo-35059.)

- Las funciones `PyByteArray_Init()` y `PyByteArray_Fini()` se han eliminado. No eran de utilidad desde Python 2.7.4 y Python 3.2.0, cuando fueron excluidas de la API limitada (ABI estable) y dejaron de estar documentadas. (Contribución de Victor Stinner en bpo-35713.)

- El resultado de `PyExceptionClass_Name()` es ahora de tipo `const char *` en vez de `char *`. (Contribución de Serhiy Storchaka en bpo-33818.)

- La dualidad conformada por `Modules/Setup.dist` y `Modules/Setup` ha sido eliminada. Anteriormente, al actualizar el árbol de fuentes de CPython, se tenía que copiar manualmente `Modules/Setup.dist` (dentro del árbol de fuentes) a `Modules/Setup` (dentro del árbol de compilación) para reflejar cualquier cambio en sentido ascendente. Esto suponía un pequeño beneficio para los empaquetadores, a expensas de una frecuente molestia para los desarrolladores de CPython, ya que olvidarse de copiar el archivo podía ocasionar fallos de compilación.

  Ahora el sistema de compilación siempre lee desde `Modules/Setup` dentro del árbol de fuentes. Se recomienda a las personas que deseen personalizar este archivo que mantengan sus cambios en un fork de git de CPython o como archivos de parche, como harían con cualquier otro cambio en el árbol de fuentes.

  (Contribución de Antoine Pitrou en bpo-32430.)

- Las funciones que convierten un número de Python a un entero de C, como `PyLong_AsLong()`, y las funciones de análisis de argumentos como `PyArg_ParseTuple()` con unidades de formato de conversión de enteros como `'i'`, ahora usarán el método especial `__index__()` si está disponible, en lugar de `__int__()`. Una advertencia de deprecación se emitirá para aquellos objetos que tengan el método `__int__()` pero no el método `__index__()` (como `Decimal` y `Fraction`). `PyNumber_Check()` ahora retornará `1` para los objetos que implementen `__index__()`. `PyNumber_Long()`, `PyNumber_Float()` y `PyFloat_AsDouble()` ahora también usan el método `__index__()` si está disponible. (Contribución de Serhiy Storchaka en bpo-36048 y bpo-20092.)

- Los objetos de tipo asignados al montículo ahora aumentarán su recuento de referencias en `PyObject_Init()` (y en su macro paralela `PyObject_INIT`) en lugar de en `PyType_GenericAlloc()`. Es posible que deban ajustarse los tipos que modifican la asignación o desasignación de instancias. (Contribución de Eddie Elizondo en bpo-35810.)

- La nueva función `PyCode_NewWithPosOnlyArgs()` permite crear objetos de código, al igual que `PyCode_New()`, pero con un parámetro *posonlyargcount* extra, que permite indicar el número de argumentos solo posicionales.

- `Py_SetPath()` ahora establece `sys.executable` en la ruta completa del programa (`Py_GetProgramFullPath()`), en vez de en el nombre del programa (`Py_GetProgramName()`). (Contribución de Victor Stinner en bpo-38234.)

8 Obsoleto

- El comando `bdist_wininst` de distutils está ahora obsoleto, usar `bdist_wheel` (paquetes wheel) en su lugar. (Contribución de Victor Stinner en bpo-37481.)
- Los métodos `getchildren()` y `getiterator()` del módulo `ElementTree` ahora emiten una advertencia `DeprecationWarning`, en lugar de `PendingDeprecationWarning`. Serán eliminados en Python 3.9. (Contribución de Serhiy Storchaka en bpo-29209.)
- Pasar un objeto a `loop.set_default_executor()` que no sea una instancia de `concurrent.futures.ThreadPoolExecutor` está obsoleto y será prohibido en Python 3.9. (Contribución de Elvis Pranskevichus en bpo-34075.)
- Los métodos `__getitem__()` pertenecientes a las clases `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` y `fileinput.FileInput` están obsoletos a partir de ahora.

  Las implementaciones de estos métodos han estado ignorando su parámetro *index* y retornando el siguiente item en su lugar. (Contribución de Berker Peksag en bpo-9372.)
- El atributo `_field_types` de la clase `typing.NamedTuple` está ahora obsoleto en favor del atributo `__annotations__`, que contiene la misma información. (Contribución de Raymond Hettinger en bpo-36320.)
- Las clases `Num`, `Str`, `Bytes`, `NameConstant` y `Ellipsis` del módulo `ast` están consideradas obsoletas y serán eliminadas en versiones futuras de Python. La clase `Constant` debe ser usada en su lugar. (Contribución de Serhiy Storchaka en bpo-32892.)
- Los métodos `visit_Num()`, `visit_Str()`, `visit_Bytes()`, `visit_NameConstant()` y `visit_Ellipsis()` de la clase `ast.NodeVisitor` están obsoletos ahora y serán invocados en versiones futuras de Python. Agregar el método `visit_Constant()` para manejar todos los nodos constantes. (Contribución de Serhiy Storchaka en bpo-36917.)
- El decorator `asyncio.coroutine()` está obsoleto y será eliminado en Python 3.10. En lugar de `@asyncio.coroutine`, se debe usar `async def`. (Contribución de Andrew Svetlov en bpo-36921.)
- En `asyncio`, pasar un argumento *loop* de forma explícita está ahora obsoleto y será eliminado en Python 3.10 para las siguientes funciones y constructores: `asyncio.sleep()`, `asyncio.gather()`, `asyncio.shield()`, `asyncio.wait_for()`, `asyncio.wait()`, `asyncio.as_completed()`, `asyncio.Task`, `asyncio.Lock`, `asyncio.Event`, `asyncio.Condition`, `asyncio.Semaphore`, `asyncio.BoundedSemaphore`, `asyncio.Queue`, `asyncio.create_subprocess_exec()` y `asyncio.create_subprocess_shell()`.
- El paso explícito de objetos corrutina a `asyncio.wait()` está ahora obsoleto y será eliminado en Python 3.11. (Contribución de Yury Selivanov en bpo-34790.)
- Las siguientes funciones y métodos del módulo `gettext` están obsoletos para: `lgettext()`, `ldgettext()`, `lngettext()` y `ldngettext()`. Retornan bytes codificados y es posible obtener excepciones inesperadas relacionadas con Unicode si hay problemas de codificación con las cadenas traducidas. En Python 3 es mucho mejor usar alternativas que retornen cadenas Unicode. Estas funciones han estado rotas durante mucho tiempo.

  Function `bind_textdomain_codeset()`, methods `output_charset()` and `set_output_charset()`, and the *codeset* parameter of functions `translation()` and `install()` are also deprecated, since they are only used for the `l*gettext()` functions. (Contributed by Serhiy Storchaka in bpo-33710.)

- El método `isAlive()` de la clase `threading.Thread` está ahora obsoleto. (Contribución de Dong-hee Na en bpo-35283.)
- Muchas funciones incorporadas y de extensión que toman argumentos enteros ahora emitirán una advertencia de deprecación para `Decimal`s, `Fraction`s y cualquier otro objeto que se pueda convertir a enteros solamente con pérdida (por ejemplo, aquellos que tienen el método `__int__()` pero no el método `__index__()`). En una versión futura, esto constituirá un error. (Contribución de Serhiy Storchaka en bpo-36048.)
- El paso de los siguientes argumentos como argumentos por palabra clave está ahora obsoleto:
  - *func* en `functools.partialmethod()`, `weakref.finalize()`, `profile.Profile.runcall()`, `cProfile.Profile.runcall()`, `bdb.Bdb.runcall()`, `trace.Trace.runfunc()` y `curses.wrapper()`.
  - *function* en `unittest.TestCase.addCleanup()`.
  - *fn* en el método `submit()` de las clases `concurrent.futures.ThreadPoolExecutor` y `concurrent.futures.ProcessPoolExecutor`.
  - *callback* en `contextlib.ExitStack.callback()`, `contextlib.AsyncExitStack.callback()` y `contextlib.AsyncExitStack.push_async_callback()`.
  - *c* y *typeid* en el método `create()` de las clases `multiprocessing.managers.Server` y `multiprocessing.managers.SharedMemoryServer`.
  - *obj* en `weakref.finalize()`.

  En futuros lanzamientos de Python, todos ellos serán argumentos solo posicionales. (Contribución de Serhiy Storchaka en bpo-36492.)

9 APIs y características eliminadas

Las siguientes características y APIs se han eliminado de Python 3.8:

- A partir de Python 3.3, la importación de ABC desde el módulo `collections` quedó obsoleta y la importación debe realizarse desde el módulo `collections.abc`. La posibilidad de importar desde collections se marcó para su eliminación en Python 3.8, pero se ha retrasado a Python 3.9. (Consultar bpo-36952.)
- El módulo `macpath`, obsoleto desde Python 3.7, ha sido eliminado. (Contribución de Victor Stinner en bpo-35471.)
- La función `platform.popen()` ha sido eliminada, después de haber estado obsoleta desde Python 3.3: usa `os.popen()` en su lugar. (Contribución de Victor Stinner en bpo-35345.)
- La función `time.clock()` ha sido eliminada, después de haber quedado obsoleta desde Python 3.3: usa `time.perf_counter()` o `time.process_time()` en su lugar, dependiendo de tus requisitos, para tener un comportamiento bien definido. (Contribución de Matthias Bussonnier en bpo-36895.)
- El script `pyvenv` se ha eliminado, en favor de `python3.8 -m venv`, para ayudar a eliminar la confusión sobre a qué intérprete de Python está vinculado el script `pyvenv`. (Contribución de Brett Cannon en bpo-25427.)
- Las funciones `parse_qs`, `parse_qsl` y `escape` se han eliminado del módulo `cgi`. Estaban obsoletas desde Python 3.2 o versiones anteriores. En su lugar, deberían ser importadas desde los módulos `urllib.parse` y `html`.
- La función `filemode` se ha eliminado del módulo `tarfile`. Estaba indocumentada y obsoleta desde Python 3.3.
- El constructor de `XMLParser` ya no acepta el argumento *html*. Nunca tuvo efecto y quedó obsoleto en Python 3.4. Todos los demás parámetros son ahora parámetros solo nombrados. (Contribución de Serhiy Storchaka en bpo-29209.)

- Se ha eliminado el método `doctype()` de `XMLParser`. (Contribución de Serhiy Storchaka en bpo-29209.)
- Se ha elimina el códec «unicode_internal». (Contribución de Inada Naoki en bpo-36297.)
- Los objetos `Cache` y `Statement` del módulo `sqlite3` no estarán expuestos al usuario a partir de ahora. (Contribución de Aviv Palivoda en bpo-30262.)
- El argumento por palabra clave `bufsize` de `fileinput.input()` y `fileinput.FileInput()`, marcado como obsoleto e ignorado desde Python 3.6, ha sido eliminado. bpo-36952 (Contribución de Matthias Bussonnier.)
- Las funciones `sys.set_coroutine_wrapper()` y `sys.get_coroutine_wrapper()`, obsoletas desde Python 3.7, han sido eliminadas; bpo-36933 (Contribución de Matthias Bussonnier.)

10 Portando a Python 3.8

Esta sección enumera los cambios descritos anteriormente y otras correcciones de errores que pueden requerir cambios en tu código.

10.1 Cambios en el comportamiento de Python

- Las expresiones yield (tanto la cláusula `yield` como la cláusula `yield from`) ahora no están permitidas en comprensiones y expresiones generadoras (excepto en la expresión iterable en la cláusula `for` situada más a la izquierda). (Contribución de Serhiy Storchaka en bpo-10544.)
- El compilador ahora produce una advertencia `SyntaxWarning` cuando se utilizan comprobaciones de identidad (`is` e `is not`) con ciertos tipos de literales (por ejemplo, cadenas, números). A menudo, estas pueden funcionar accidentalmente en CPython, pero no está garantizado por las especificaciones del lenguaje. La advertencia advierte a los usuarios que utilicen pruebas de igualdad (== y `!=`) en su lugar. (Contribución de Serhiy Storchaka en bpo-34850.)
- El intérprete de CPython puede tolerar excepciones en algunas circunstancias. En Python 3.8 esto sucederá con menos frecuencia. En particular, las excepciones que se generan al obtener atributos del diccionario de tipos ya no son ignoradas. (Contribución de Serhiy Storchaka en bpo-35459.)
- Se ha eliminado las implementaciones de `__str__` para los tipos incorporados `bool`, `int`, `float`, `complex` y algunas clases de la biblioteca estándar. Ahora heredan el método `__str__()` de `object`. Como resultado, definir el método `__repr__()` en una subclase de estas clases afectará a su representación como cadena de caracteres. (Contribución de Serhiy Storchaka en bpo-36793.)
- En AIX, el atributo `sys.platform` ya no contiene la versión principal. Es decir, siempre es `'aix'`, en lugar de `'aix3'` .. `'aix7'`. Dado que las versiones anteriores de Python incluyen el número de versión, se recomienda usar siempre `sys.platform.startswith('aix')`. (Contribución de M. Felt en bpo-36588.)
- `PyEval_AcquireLock()` y `PyEval_AcquireThread()` ahora terminan el hilo actual si se llaman mientras el intérprete está finalizando, haciéndolos consistentes con `PyEval_RestoreThread()`, `Py_END_ALLOW_THREADS()` y `PyGILState_Ensure()`. Si no se desea este comportamiento, se tiene que proteger la invocación comprobando `_Py_IsFinalizing()` o `sys.is_finalizing()`. (Contribución de Joannah Nanjekye en bpo-36475.)

10.2 Cambios en la API de Python

- La función `os.getcwdb()` ahora usa la codificación UTF-8 en Windows, en lugar de la página de códigos ANSI: consultar **PEP 529** para el fundamento. La función ya no está obsoleta en Windows. (Contribución de Victor Stinner en bpo-37412.)
- `subprocess.Popen` ahora puede usar `os.posix_spawn()` en algunos casos para un mejor rendimiento. En el Subsistema de Windows para Linux y en la Emulación de usuario QEMU, el constructor `Popen` que usa `os.posix_spawn()` ya no genera una excepción como «missing program» ante errores. En cambio, el proceso hijo falla con un valor `returncode` distinto de cero. (Contribución de Joannah Nanjekye y Victor Stinner en bpo-35537.)
- El argumento *preexec_fn* de `subprocess.Popen` ya no es compatible con subintérpretes. El uso del parámetro en un subintérprete ahora lanza una excepción `RuntimeError`. (Contribución de Eric Snow en bpo-34651, modificado por Christian Heimes en bpo-37951.)
- El método `imap.IMAP4.logout()` ya no ignora silenciosamente excepciones arbitrarias. (Contribución de Victor Stinner en bpo-36348.)
- La función `platform.popen()` ha sido eliminada, después de haber estado obsoleta desde Python 3.3: usa `os.popen()` en su lugar. (Contribución de Victor Stinner en bpo-35345.)
- La función `statistics.mode()` ya no lanza una excepción cuando se proporcionan datos multimodales. Ahora, en cambio, retorna la primera moda encontrada en los datos de entrada. (Contribución de Raymond Hettinger en bpo-35892.)
- El método `selection()` de la clase `tkinter.ttk.Treeview` ya no acepta argumentos. Usarlo con argumentos para cambiar la selección quedó obsoleto en Python 3.6. Utiliza métodos especializados, como `selection_set()`, para cambiar la selección. (Contribución de Serhiy Storchaka en bpo-31508.)
- Los métodos `writexml()`, `toxml()` y `toprettyxml()` de `xml.dom.minidom` y el método `write()` de `xml.etree`, ahora conservan el orden de los atributos especificado por el usuario. (Contribución de Diego Rojas y Raymond Hettinger en bpo-34160.)
- Una base de datos `dbm.dumb` abierta con el flag `'r'` ahora es de solo lectura. `dbm.dumb.open()` con los flags `'r'` y `'w'` ya no crea una base de datos si no existe. (Contribución de Serhiy Storchaka en bpo-32749.)
- El método `doctype()` definido en una subclase de `XMLParser` ya no será invocado y emitirá una advertencia `RuntimeWarning` en lugar de `DeprecationWarning`. Define el método `doctype()` en un objetivo para manejar una declaración doctype de XML. (Contribución de Serhiy Storchaka en bpo-29209.)
- Ahora se lanza una excepción `RuntimeError` cuando la metaclase personalizada no proporciona la entrada `__classcell__` en el espacio de nombres pasado a `type.__new__`. En Python 3.6–3.7. se emitía una advertencia `DeprecationWarning` (Contribución de Serhiy Storchaka en bpo-23722.)
- La clase `cProfile.Profile` ahora se puede usar como gestor de contexto. (Contribución de Scott Sanderson en bpo-29235.)
- `shutil.copyfile()`, `shutil.copy()`, `shutil.copy2()`, `shutil.copytree()` y `shutil.move()` usan llamadas al sistema de «copia-rápida» específicas de la plataforma. (Consultar la sección shutil-platform-dependent-efficient-copy-operations).
- El tamaño predeterminado del búfer de `shutil.copyfile()` en Windows se ha cambiado de 16 KiB a 1 MiB.
- La estructura `PyGC_Head` ha cambiado por completo. Todo código que haga uso de algún miembro de la estructura debe reescribirse. (Consultar bpo-33597.)
- La estructura `PyInterpreterState` se ha movido a los archivos de cabeceras «internos» (específicamente a Include/internal/pycore_pystate.h). Un `PyInterpreterState` opaco todavía está disponible como parte de la API pública (y ABI estable). La documentación indica que ninguno de los campos de la estructura es público, por lo que esperamos que nadie los haya estado usando. Sin embargo, si utilizas uno o más de esos campos privados y

no tienes otra alternativa, abre una issue BPO. Trabajaremos para ayudarte a adaptarlo (posiblemente incluyendo funciones de acceso a la API pública). (Consultar bpo-35886.)

- El método `mmap.flush()` ahora retorna `None` en caso de éxito y lanza una excepción en caso de error, en todas las plataformas. Anteriormente, su comportamiento dependía de la plataforma: en Windows retornaba un valor distinto de cero en caso de éxito y cero ante un error. En Unix se retornaba un valor de cero en caso de éxito y se lanzaba una excepción ante un error. (Contribución de Berker Peksag en bpo-2122.)
- Los módulos `xml.dom.minidom` y `xml.sax` ya no procesan entidades externas de forma predeterminada. (Contribución de Christian Heimes en bpo-17239.)
- Eliminar una clave de una base de datos `dbm` de solo lectura (`dbm.dumb`, `dbm.gnu` o `dbm.ndbm`) lanza una excepción `error` (`dbm.dumb.error`, `dbm.gnu.error` o `dbm.ndbm.error`) en lugar de `KeyError`. (Contribución de Xiang Zhang en bpo-33106.)
- AST simplificado para literales. Todas las constantes se representarán como instancias de `ast.Constant`. La instanciación de las antiguas clases `Num`, `Str`, `Bytes`, `NameConstant` y `Ellipsis` retornará ahora una instancia de `Constant`. (Contribución de Serhiy Storchaka en bpo-32892.)
- `expanduser()` en Windows ahora usa preferentemente la variable de entorno `USERPROFILE` y no usa `HOME`, que normalmente no está establecido para cuentas de usuario normales. (Contribución de Anthony Sottile en bpo-36264.)
- The exception `asyncio.CancelledError` now inherits from `BaseException` rather than `Exception` and no longer inherits from `concurrent.futures.CancelledError`. (Contributed by Yury Selivanov in bpo-32528.)
- La función `asyncio.wait_for()` ahora espera correctamente la cancelación cuando se usa una instancia de `asyncio.Task`. Anteriormente, al alcanzar *timeout*, se cancelaba y retornaba de inmediato. (Contribución de Elvis Pranskevichus en bpo-32751.)
- La función `asyncio.BaseTransport.get_extra_info()` ahora retorna un objeto socket seguro cuando se pasa "socket" al parámetro *name*. (Contribución de Yury Selivanov en bpo-37027.)
- `asyncio.BufferedProtocol` ha pasado a formar parte de la API estable.
- Las dependencias de DLLs para módulos de extensión y DLLs cargadas con `ctypes` en Windows ahora se resuelven de forma más segura. Solo las rutas del sistema, el directorio que contiene el archivo DLL o PYD y los directorios agregados mediante `add_dll_directory()` se buscan para las dependencias en tiempo de carga. Específicamente, `PATH` y el directorio de trabajo actual ya no se utilizan, y las modificaciones de estos ya no tendrán ningún efecto en la resolución normal de la DLL. Si tu aplicación se basa en estos mecanismos, debes buscar `add_dll_directory()` y, si existe, utilizarlo para agregar tu directorio DLL mientras carga tu biblioteca. Ten en cuenta que los usuarios de Windows 7 deberán asegurarse de que se haya instalado Windows Update KB2533623 (esto también lo verifica el instalador). (Contribución de Steve Dower en bpo-36085.)
- Los archivos de cabeceras y las funciones relacionadas con pgen se han eliminado después de su reemplazo por una implementación pura de Python. (Contribución de Pablo Galindo en bpo-36623.)
- `types.CodeType` tiene un nuevo parámetro en la segunda posición del constructor (*posonlyargcount*) para admitir argumentos solo posicionales, definidos en **PEP 570**. El primer argumento (*argcount*) ahora representa el número total de argumentos posicionales (incluidos los argumentos solo posicionales). El nuevo método `replace()` de `types.CodeType` se puede utilizar para hacer que el código esté preparado para el futuro.
- The parameter `digestmod` for `hmac.new()` no longer uses the MD5 digest by default.

10.3 Cambios en la API de C

- La estructura `PyCompilerFlags` tiene un nuevo campo *cf_feature_version*. Debe inicializarse en `PY_MINOR_VERSION`. El campo se ignora de forma predeterminada y se usa, si y solo si, el flag `PyCF_ONLY_AST` está establecido en *cf_flags*. (Contribución de Guido van Rossum en bpo-35766.)

- La función `PyEval_ReInitThreads()` se ha eliminado de la API de C. No debe llamarse explícitamente: usa `PyOS_AfterFork_Child()` en su lugar. (Contribución de Victor Stinner en bpo-36728.)

- En Unix, las extensiones de C ya no están vinculadas a libpython, excepto en Android y Cygwin. Cuando Python está integrado, `libpython` no debe cargarse con `RTLD_LOCAL`, sino con `RTLD_GLOBAL` en su lugar. Anteriormente, no era posible usar `RTLD_LOCAL` para cargar extensiones de C que no estuvieran vinculadas a `libpython`, como las extensiones de C de la biblioteca estándar construida por la sección `*shared*` de `Modules/Setup`. (Contribución de Victor Stinner en bpo-21536.)

- El uso de variantes de formato `#` en el análisis o la construcción de valores (por ejemplo: `PyArg_ParseTuple()`, `Py_BuildValue()`, `PyObject_CallFunction()`, etc.) sin `PY_SSIZE_T_CLEAN` definido ahora lanza una advertencia `DeprecationWarning`. Se eliminará en Python 3.10 ó 4.0. Consultar arg-parsing para más detalles. (Contribución de Inada Naoki en bpo-36381.)

- Las instancias de tipos asignados al montón (como los creados con `PyType_FromSpec()`) contienen una referencia a su objeto de tipo. El aumento del recuento de referencias de estos objetos de tipo se ha movido de `PyType_GenericAlloc()` a las funciones de más bajo nivel, `PyObject_Init()` y `PyObject_INIT()`. Esto hace que los tipos creados mediante `PyType_FromSpec()` se comporten como otras clases en el código gestionado.

  Los tipos asignados estáticamente no se ven afectados.

  Para la gran mayoría de casos, no debería haber efectos secundarios. Sin embargo, los tipos que aumentan manualmente el recuento de referencias después de asignar una instancia (quizás para evitar el error) ahora pueden volverse inmortales. Para evitar esto, estas clases deben llamar a Py_DECREF en el objeto de tipo durante la desasignación de la instancia.

  Para portar correctamente estos tipos a Python 3.8, aplica los siguientes cambios:

  - Elimina `Py_INCREF` en el objeto de tipo después de asignar una instancia, si la hubiera. Esto puede suceder después de invocar a `PyObject_New()`, `PyObject_NewVar()`, `PyObject_GC_New()`, `PyObject_GC_NewVar()`, o cualquier otro asignador personalizado que use `PyObject_Init()` o `PyObject_INIT()`.

    Ejemplo:

    ```
    static foo_struct *
    foo_new(PyObject *type) {
        foo_struct *foo = PyObject_GC_New(foo_struct, (PyTypeObject *) type);
        if (foo == NULL)
            return NULL;
    #if PY_VERSION_HEX < 0x03080000
        // Workaround for Python issue 35810; no longer necessary in Python 3.8
        PY_INCREF(type)
    #endif
        return foo;
    }
    ```

  - Asegúrate de que todas las funciones personalizadas `tp_dealloc` de los tipos asignados al montón disminuyan el recuento de referencias del tipo.

    Ejemplo:

```
static void
foo_dealloc(foo_struct *instance) {
    PyObject *type = Py_TYPE(instance);
    PyObject_GC_Del(instance);
#if PY_VERSION_HEX >= 0x03080000
    // This was not needed before Python 3.8 (Python issue 35810)
    Py_DECREF(type);
#endif
}
```

(Contribución de Eddie Elizondo en bpo-35810.)

- La macro `Py_DEPRECATED()` ha sido implementada para MSVC. La macro ahora debe ser colocada antes del nombre del símbolo.

  Ejemplo:

```
Py_DEPRECATED(3.8) PyAPI_FUNC(int) Py_OldFunction(void);
```

  (Contribución de Zackery Spytz en bpo-33407.)

- El intérprete ya no pretende dar suporte nunca más a la compatibilidad binaria de tipos de extensión entre versiones de características. Un `PyTypeObject` exportado por un módulo de extensión de terceros se supone que tiene todas las ranuras esperadas por la versión actual de Python, incluyendo `tp_finalize` (`Py_TPFLAGS_HAVE_FINALIZE` ya no se verifica antes de leer `tp_finalize`).

  (Contribución de Antoine Pitrou en bpo-32388.)

- Las funciones `PyNode_AddChild()` y `PyParser_AddToken()` ahora aceptan dos argumentos `int` adicionales, *end_lineno* y *end_col_offset*.

- El archivo `libpython38.a` ya no se incluye en la distribución normal de Windows. Si se necesita este archivo, se puede generar con las herramientas `gendef` y `dlltool`, que son parte del paquete binutils de MinGW:

```
gendef - python38.dll > tmp.def
dlltool --dllname python38.dll --def tmp.def --output-lib libpython38.a
```

  La ubicación de un archivo `pythonXY.dll` instalado dependerá de las opciones de instalación y de la versión y el idioma de Windows. Consultar using-on-windows para obtener más información. La biblioteca resultante debe colocarse en el mismo directorio que `pythonXY.lib`, que generalmente es el directorio `libs` en tu instalación de Python.

  (Contribución de Steve Dower en bpo-37351.)

10.4 Cambios en el bytecode de CPython

- El bucle del intérprete se ha simplificado moviendo al compilador la lógica para el desenredo de la pila de bloques. El compilador ahora emite instrucciones explícitas para ajustar la pila de valores y llamar al código de limpieza para `break`, `continue` y `return`.

  Eliminados los códigos de operación `BREAK_LOOP`, `CONTINUE_LOOP`, `SETUP_LOOP` y `SETUP_EXCEPT`. Agregados los nuevos códigos de operación `ROT_FOUR`, `BEGIN_FINALLY`, `CALL_FINALLY` y `POP_FINALLY`. Modificados los códigos de operación `END_FINALLY` y `WITH_CLEANUP_START`.

  (Contribución de Mark Shannon, Antoine Pitrou y Serhiy Storchaka en bpo-17611.)

- Agregado el código de operación `END_ASYNC_FOR` para manejar excepciones lanzadas mientras se espera al siguiente elemento en un ciclo `async for`. (Contribución de Serhiy Storchaka en bpo-33041.)

- El código de operación `MAP_ADD` ahora espera el valor como primer elemento de la pila y la clave como segundo elemento. Este cambio se realizó para que la clave siempre se evalúe antes que el valor en los diccionarios por compresión, como se propone en **PEP 572**. (Contribución de Jörn Heissler en bpo-35224.)

10.5 Demos y herramientas

Se ha añadido un script de evaluación de rendimiento para cronometrar varias formas de acceder a variables: `Tools/scripts/var_access_benchmark.py`. (Contribución de Raymond Hettinger en bpo-35884.)

A continuación, se muestra un resumen de las mejoras de rendimiento desde Python 3.3:

```
Python version                       3.3     3.4     3.5     3.6     3.7     3.8
--------------                       ---     ---     ---     ---     ---     ---

Variable and attribute read access:
    read_local                       4.0     7.1     7.1     5.4     5.1     3.9
    read_nonlocal                    5.3     7.1     8.1     5.8     5.4     4.4
    read_global                     13.3    15.5    19.0    14.3    13.6     7.6
    read_builtin                    20.0    21.1    21.6    18.5    19.0     7.5
    read_classvar_from_class        20.5    25.6    26.5    20.7    19.5    18.4
    read_classvar_from_instance     18.5    22.8    23.5    18.8    17.1    16.4
    read_instancevar                26.8    32.4    33.1    28.0    26.3    25.4
    read_instancevar_slots          23.7    27.8    31.3    20.8    20.8    20.2
    read_namedtuple                 68.5    73.8    57.5    45.0    46.8    18.4
    read_boundmethod                29.8    37.6    37.9    29.6    26.9    27.7

Variable and attribute write access:
    write_local                      4.6     8.7     9.3     5.5     5.3     4.3
    write_nonlocal                   7.3    10.5    11.1     5.6     5.5     4.7
    write_global                    15.9    19.7    21.2    18.0    18.0    15.8
    write_classvar                  81.9    92.9    96.0   104.6   102.1    39.2
    write_instancevar               36.4    44.6    45.8    40.0    38.9    35.5
    write_instancevar_slots         28.7    35.6    36.1    27.3    26.6    25.7

Data structure read access:
    read_list                       19.2    24.2    24.5    20.8    20.8    19.0
    read_deque                      19.9    24.7    25.5    20.2    20.6    19.8
    read_dict                       19.7    24.3    25.7    22.3    23.0    21.0
    read_strdict                    17.9    22.6    24.3    19.5    21.2    18.9

Data structure write access:
    write_list                      21.2    27.1    28.5    22.5    21.6    20.0
    write_deque                     23.8    28.7    30.1    22.7    21.8    23.5
    write_dict                      25.9    31.4    33.3    29.3    29.2    24.7
    write_strdict                   22.9    28.4    29.9    27.5    25.2    23.1

Stack (or queue) operations:
    list_append_pop                144.2    93.4   112.7    75.4    74.2    50.8
    deque_append_pop                30.4    43.5    57.0    49.4    49.2    42.5
    deque_append_popleft            30.8    43.7    57.3    49.7    49.7    42.8

Timing loop:
    loop_overhead                    0.3     0.5     0.6     0.4     0.3     0.3
```

Las evaluaciones de rendimiento se realizaron en un procesador Intel® Core™ i7-4960HQ ejecutando las compilaciones de 64-bits para macOS disponibles en python.org. El script de evaluación de rendimiento muestra los tiempos en nanosegundos.

11 Cambios notables en Python 3.8.1

Debido a importantes preocupaciones de seguridad, el parámetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()` ya no está soportado. Esto se debe al comportamiento de la opción `SO_REUSEADDR` del socket en UDP. Para obtener más detalles, consulta la documentación de `loop.create_datagram_endpoint()`. (Contribución de Kyle Stanley, Antoine Pitrou y Yury Selivanov en bpo-37228.)

12 Cambios notables en Python 3.8.2

Se ha corregido una regresión con la retrollamada `ignore` de `shutil.copytree()`. Los tipos de los argumentos ahora son str y List[str] nuevamente. (Contribución de Manuel Barkhau y Giampaolo Rodola en bpo-39390.)

13 Cambios notables en Python 3.8.3

Se han actualizado los valores constantes de los flags futuros del módulo `__future__` para evitar colisiones con los flags del compilador. Anteriormente, `PyCF_ALLOW_TOP_LEVEL_AWAIT` estaba en conflicto con `CO_FUTURE_DIVISION`. (Contribución de Batuhan Taskaya en bpo-39562.)

14 Notable changes in Python 3.8.8

Earlier Python versions allowed using both `;` and `&` as query parameter separators in `urllib.parse.parse_qs()` and `urllib.parse.parse_qsl()`. Due to security concerns, and to conform with newer W3C recommendations, this has been changed to allow only a single separator key, with `&` as the default. This change also affects `cgi.parse()` and `cgi.parse_multipart()` as they use the affected functions internally. For more details, please see their respective documentation. (Contributed by Adam Goldschmidt, Senthil Kumaran and Ken Jin in bpo-42967.)

15 Notable changes in Python 3.8.9

A security fix alters the `ftplib.FTP` behavior to not trust the IPv4 address sent from the remote server when setting up a passive data channel. We reuse the ftp server IP address instead. For unusual code requiring the old behavior, set a `trust_server_pasv_ipv4_address` attribute on your FTP instance to `True`. (See bpo-43285)

16 Notable changes in Python 3.8.10

16.1 macOS 11.0 (Big Sur) and Apple Silicon Mac support

As of 3.8.10, Python now supports building and running on macOS 11 (Big Sur) and on Apple Silicon Macs (based on the `ARM64` architecture). A new universal build variant, `universal2`, is now available to natively support both `ARM64` and `Intel 64` in one set of executables. Note that support for «weaklinking», building binaries targeted for newer versions of macOS that will also run correctly on older versions by testing at runtime for missing features, is not included in this backport from Python 3.9; to support a range of macOS versions, continue to target for and build on the oldest version in the range.

(Originally contributed by Ronald Oussoren and Lawrence D'Anna in bpo-41100, with fixes by FX Coudert and Eli Rykoff, and backported to 3.8 by Maxime Bélanger and Ned Deily)

17 Notable changes in Python 3.8.10

17.1 urllib.parse

The presence of newline or tab characters in parts of a URL allows for some forms of attacks. Following the WHATWG specification that updates **RFC 3986**, ASCII newline `\n`, `\r` and tab `\t` characters are stripped from the URL by the parser in `urllib.parse` preventing such attacks. The removal characters are controlled by a new module level variable `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (See bpo-43882)

18 Notable changes in Python 3.8.12

18.1 Cambios en la API de Python

Starting with Python 3.8.12 the `ipaddress` module no longer accepts any leading zeros in IPv4 address strings. Leading zeros are ambiguous and interpreted as octal notation by some libraries. For example the legacy function `socket.inet_aton()` treats leading zeros as octal notation. glibc implementation of modern `inet_pton()` does not accept any leading zeros.

(Originally contributed by Christian Heimes in bpo-36384, and backported to 3.8 by Achraf Merzouki)

19 Notable security feature in 3.8.14

Converting between `int` and `str` in bases other than 2 (binary), 4, 8 (octal), 16 (hexadecimal), or 32 such as base 10 (decimal) now raises a `ValueError` if the number of digits in string form is above a limit to avoid potential denial of service attacks due to the algorithmic complexity. This is a mitigation for CVE-2020-10735. This limit can be configured or disabled by environment variable, command line flag, or `sys` APIs. See the integer string conversion length limitation documentation. The default limit is 4300 digits in string form.

20 Notable Changes in 3.8.17

20.1 tarfile

- The extraction methods in `tarfile`, and `shutil.unpack_archive()`, have a new a *filter* argument that allows limiting tar features than may be surprising or dangerous, such as creating files outside the destination directory. See tarfile-extraction-filter for details. In Python 3.12, use without the *filter* argument will show a `DeprecationWarning`. In Python 3.14, the default will switch to `'data'`. (Contributed by Petr Viktorin in **PEP 706**.)

21 Notable changes in 3.8.20

21.1 ipaddress

- Fixed `is_global` and `is_private` behavior in `IPv4Address`, `IPv6Address`, `IPv4Network` and `IPv6Network`.

21.2 email

- Headers with embedded newlines are now quoted on output.

  The `generator` will now refuse to serialize (write) headers that are improperly folded or delimited, such that they would be parsed as multiple headers or joined with adjacent data. If you need to turn this safety feature off, set `verify_generated_headers`. (Contributed by Bas Bloemsaat and Petr Viktorin in gh-121650.)

- `email.utils.getaddresses()` and `email.utils.parseaddr()` now return `('', '')` 2-tuples in more situations where invalid email addresses are encountered, instead of potentially inaccurate values. An optional *strict* parameter was added to these two functions: use `strict=False` to get the old behavior, accepting malformed inputs. `getattr(email.utils, 'supports_strict_parsing', False)` can be used to check if the *strict* paramater is available. (Contributed by Thomas Dwyer and Victor Stinner for gh-102988 to improve the CVE-2023-27043 fix.)

Índice

H

P

R

U

V