What's New in Python

Versión 3.12.3

A. M. Kuchling

mayo 06, 2024

Python Software Foundation
Email: docs@python.org

Índice general

Editor
Adam Turner

Este artículo explica las nuevas características de Python 3.12, en comparación con 3.11. Python 3.12 se lanzó el 2 de octubre de 2023. Para obtener detalles completos, consulte changelog.

Ver también:

PEP 693 - Calendario de lanzamiento de Python 3.12

1 Resumen: aspectos destacados de la versión

Python 3.12 es la última versión estable del lenguaje de programación Python, con una combinación de cambios en el lenguaje y la biblioteca estándar. Los cambios en la biblioteca se centran en limpiar las API obsoletas, su usabilidad y su corrección. Es de destacar que el paquete `distutils` se ha eliminado de la biblioteca estándar. La compatibilidad con el sistema de archivos en `os` y `pathlib` ha experimentado una serie de mejoras y varios módulos tienen un mejor rendimiento.

Los cambios de idioma se centran en la usabilidad, ya que a f-strings se le han eliminado muchas limitaciones y las sugerencias «¿Quiso decir…» continúan mejorando. La nueva declaración *type parameter syntax* y `type` mejora la ergonomía para usar generic types y type aliases con controladores de tipo estático.

Este artículo no intenta proporcionar una especificación completa de todas las características nuevas, sino que brinda una descripción general conveniente. Para obtener detalles completos, debe consultar la documentación, como Library Reference y Language Reference. Si desea comprender la implementación completa y la justificación del diseño de un cambio, consulte el PEP para conocer una característica nueva en particular; pero tenga en cuenta que los PEP generalmente no se mantienen actualizados una vez que una característica se ha implementado por completo.

Nuevas características de sintaxis:

- *PEP 695*, sintaxis de parámetro de tipo y sentencia `type`

Nuevas características gramaticales:

- *PEP 701*, f-strings en la gramática

Mejoras del intérprete:

- *PEP 684*, un GIL único por intérprete
- *PEP 669*, monitorización de bajo impacto
- *Improved "Did you mean …" suggestions* para excepciones `NameError`, `ImportError` y `SyntaxError`

Mejoras en el modelo de datos de Python:

- *PEP 688*, usando buffer protocol de Python

Mejoras significativas en la biblioteca estándar:

- La clase `pathlib.Path` ahora admite subclases
- El módulo `os` recibió varias mejoras para el soporte de Windows
- Se ha agregado un command-line interface al módulo `sqlite3`
- Los controles `isinstance()` frente a `runtime-checkable protocols` benefician de una aceleración de entre dos y 20 veces
- El paquete `asyncio` ha tenido una serie de mejoras de rendimiento, y algunos puntos de referencia muestran una aceleración del 75 %.
- Se ha agregado un command-line interface al módulo `uuid`
- Due to the changes in *PEP 701*, producing tokens via the `tokenize` module is up to 64% faster.

Mejoras de seguridad:

- Reemplaza las implementaciones `hashlib` integradas de SHA1, SHA3, SHA2-384, SHA2-512 y MD5 con código verificado formalmente del proyecto HACL*. Estas implementaciones integradas permanecen como alternativas que solo se utilizan cuando OpenSSL no las proporciona.

Mejoras de la API C:

- *PEP 697*, nivel de API C inestable
- *PEP 683*, objetos inmortales

Mejoras en la implementación de CPython:

- *PEP 709*, comprensión en línea
- CPython support para el perfilador Linux `perf`
- Implementar protección contra desbordamiento de pila en plataformas compatibles

Nuevas funciones de tipado:

- *PEP 692*, usando `TypedDict` para anotar **kwargs
- *PEP 698*, decorador `typing.override()`

Depreciaciones, eliminaciones o restricciones importantes:

- **PEP 623**: elimina `wstr` de los objetos Unicode en la API C de Python, reduciendo el tamaño de cada objeto `str` en al menos 8 bytes.
- **PEP 632**: elimina el paquete `distutils`. Consulte the migration guide para obtener consejos sobre cómo reemplazar las API que proporcionaba. El paquete Setuptools de terceros continúa proporcionando `distutils`, si aún lo necesita en Python 3.12 y versiones posteriores.
- gh-95299: No preinstala `setuptools` en entornos virtuales creados con `venv`. Esto significa que `distutils`, `setuptools`, `pkg_resources` y `easy_install` ya no estarán disponibles de forma predeterminada; para acceder a ellos, ejecute `pip install setuptools` en el entorno virtual activated.
- Se han eliminado los módulos `asynchat`, `asyncore` y `imp`, junto con varios `unittest.TestCase` *method aliases*.

2 Nuevas características

2.1 PEP 695: Sintaxis de parámetro de tipo

Las clases y funciones genéricas en **PEP 484** se declararon utilizando una sintaxis detallada que dejaba el alcance de los parámetros de tipo poco claro y requería declaraciones explícitas de variación.

PEP 695 presenta una forma nueva, más compacta y explícita de crear generic classes y functions:

```
def max[T](args: Iterable[T]) -> T:
    ...

class list[T]:
    def __getitem__(self, index: int, /) -> T:
        ...

    def append(self, element: T) -> None:
        ...
```

Además, el PEP introduce una nueva forma de declarar type aliases utilizando la declaración `type`, que crea una instancia de `TypeAliasType`:

```
type Point = tuple[float, float]
```

Los alias de tipo también pueden ser generic:

```
type Point[T] = tuple[T, T]
```

La nueva sintaxis permite declarar los parámetros `TypeVarTuple` y `ParamSpec`, así como los parámetros `TypeVar` con límites o restricciones:

```
type IntFunc[**P] = Callable[P, int]  # ParamSpec
type LabeledTuple[*Ts] = tuple[str, *Ts]  # TypeVarTuple
type HashableSequence[T: Hashable] = Sequence[T]  # TypeVar with bound
type IntOrStrSequence[T: (int, str)] = Sequence[T]  # TypeVar with constraints
```

El valor de los alias de tipo y los límites y restricciones de las variables de tipo creadas mediante esta sintaxis se evalúan solo según demanda (consulte lazy evaluation). Esto significa que los alias de tipo pueden hacer referencia a otros tipos definidos más adelante en el archivo.

Los parámetros de tipo declarados a través de una lista de parámetros de tipo son visibles dentro del alcance de la declaración y de cualquier alcance anidado, pero no en el alcance externo. Por ejemplo, se pueden utilizar en las anotaciones de tipo de los métodos de una clase genérica o en el cuerpo de la clase. Sin embargo, no se pueden utilizar en el alcance del módulo una vez definida la clase. Consulte type-params para obtener una descripción detallada de la semántica de tiempo de ejecución de los parámetros de tipo.

Para respaldar esta semántica de alcance, se introduce un nuevo tipo de alcance, el annotation scope. Los ámbitos de anotación se comportan en su mayor parte como ámbitos de función, pero interactúan de manera diferente con los ámbitos de clase adjuntos. En Python 3.13, annotations también se evaluará en ámbitos de anotación.

Consulte **PEP 695** para obtener más detalles.

(PEP escrito por Eric Traut. Implementación por Jelle Zijlstra, Eric Traut y otros en gh-103764.)

2.2 PEP 701: Formalización sintáctica de cadenas f

PEP 701 elimina algunas restricciones sobre el uso de f-strings. Los componentes de expresión dentro de cadenas f ahora pueden ser cualquier expresión Python válida, incluidas cadenas que reutilizan la misma comilla que la cadena f que las contiene, expresiones de varias líneas, comentarios, barras invertidas y secuencias de escape Unicode. Cubramos estos en detalle:

- Reutilización de comillas: en Python 3.11, reutilizar las mismas comillas que la cadena f adjunta genera un `SyntaxError`, lo que obliga al usuario a usar otras comillas disponibles (como usar comillas dobles o triples si la cadena f usa comillas simples). En Python 3.12, ahora puedes hacer cosas como esta:

```
>>> songs = ['Take me back to Eden', 'Alkaline', 'Ascensionism']
>>> f"This is the playlist: {", ".join(songs)}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

  Tenga en cuenta que antes de este cambio no había un límite explícito en cómo se pueden anidar las cadenas f, pero el hecho de que las comillas de cadena no se pueden reutilizar dentro del componente de expresión de las cadenas f hacía imposible anidar cadenas f arbitrariamente. De hecho, esta es la cadena f más anidada que podría escribirse:

```
>>> f"""{f'''{f'{f"{1+1}"}'}'''}"""
'2'
```

  Como ahora las cadenas f pueden contener cualquier expresión Python válida dentro de los componentes de expresión, ahora es posible anidar cadenas f de forma arbitraria:

```
>>> f"{f"{f"{f"{f"{f"{1+1}"}"}"}"}"}"
'2'
```

- Expresiones y comentarios de varias líneas: en Python 3.11, las expresiones de cadena f deben definirse en una sola línea, incluso si la expresión dentro de la cadena f normalmente podría abarcar varias líneas (como listas literales que se definen en varias líneas), lo que las hace más difícil de leer. En Python 3.12 ahora puedes definir cadenas f que abarquen varias líneas y agregar comentarios en línea:

```
>>> f"This is the playlist: {", ".join([
...     'Take me back to Eden',  # My, my, those eyes like fire
...     'Alkaline',              # Not acid nor alkaline
...     'Ascensionism'           # Take to the broken skies at last
... ])}"
'This is the playlist: Take me back to Eden, Alkaline, Ascensionism'
```

- Barras invertidas y caracteres Unicode: antes de Python 3.12, las expresiones de cadena f no podían contener ningún carácter `\`. Esto también afectó a Unicode escape sequences (como `\N{snowman}`), ya que contienen la parte `\N` que anteriormente no podía ser parte de los componentes de expresión de f-strings. Ahora puedes definir expresiones como esta:

```
>>> print(f"This is the playlist: {"\n".join(songs)}")
This is the playlist: Take me back to Eden
Alkaline
Ascensionism
>>> print(f"This is the playlist: {"\N{BLACK HEART SUIT}".join(songs)}")
This is the playlist: Take me back to Eden♥Alkaline♥Ascensionism
```

Consulte **PEP 701** para obtener más detalles.

As a positive side-effect of how this feature has been implemented (by parsing f-strings with **the PEG parser**), now error messages for f-strings are more precise and include the exact location of the error. For example, in Python 3.11, the following f-string raises a `SyntaxError`:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    (x z y)
     ^^^
SyntaxError: f-string: invalid syntax. Perhaps you forgot a comma?
```

pero el mensaje de error no incluye la ubicación exacta del error dentro de la línea y también tiene la expresión artificialmente entre paréntesis. En Python 3.12, a medida que las cadenas f se analizan con el analizador PEG, los mensajes de error pueden ser más precisos y mostrar la línea completa:

```
>>> my_string = f"{x z y}" + f"{1 + 1}"
  File "<stdin>", line 1
    my_string = f"{x z y}" + f"{1 + 1}"
                   ^^^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

(Aportado por Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou, Cristián Maureira-Fredes y Marta Gómez en gh-102856. PEP escrito por Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou y Marta Gómez).

2.3 PEP 684: un GIL por intérprete

PEP 684 introduce un GIL por intérprete, de modo que ahora se pueden crear subintérpretes con un GIL único por intérprete. Esto permite que los programas Python aprovechen al máximo múltiples núcleos de CPU. Actualmente, esto solo está disponible a través de C-API, aunque una API de Python es **anticipada para 3.13**.

Use the new `Py_NewInterpreterFromConfig()` function to create an interpreter with its own GIL:

```
PyInterpreterConfig config = {
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
PyThreadState *tstate = NULL;
PyStatus status = Py_NewInterpreterFromConfig(&tstate, &config);
if (PyStatus_Exception(status)) {
    return -1;
}
/* The new interpreter is now active in the current thread. */
```

Para obtener más ejemplos de cómo utilizar C-API para subintérpretes con un GIL por intérprete, consulte Modules/_xxsubinterpretersmodule.c.

(Aportado por Eric Snow en gh-104210, etc.)

2.4 PEP 669: Monitoreo de bajo impacto para CPython

PEP 669 define un nuevo `API` para perfiladores, depuradores y otras herramientas para monitorear eventos en CPython. Cubre una amplia gama de eventos, incluidas llamadas, devoluciones, líneas, excepciones, saltos y más. Esto significa que solo paga por lo que usa, brindando soporte para depuradores generales y herramientas de cobertura casi nulos. Consulte `sys.monitoring` para obtener más detalles.

(Aportado por Mark Shannon en gh-103082.)

2.5 PEP 688: Hacer accesible el protocolo de búfer en Python

PEP 688 presenta una forma de utilizar buffer protocol desde el código Python. Las clases que implementan el método `__buffer__()` ahora se pueden utilizar como tipos de búfer.

El nuevo `collections.abc.Buffer` ABC proporciona una forma estándar de representar objetos de búfer, por ejemplo en anotaciones de tipo. La nueva enumeración `inspect.BufferFlags` representa los indicadores que se pueden usar para personalizar la creación del búfer. (Aportado por Jelle Zijlstra en gh-102500.)

2.6 PEP 709: Integración de la comprensión

Las comprensiones de diccionario, lista y conjunto ahora están integradas, en lugar de crear un nuevo objeto de función de un solo uso para cada ejecución de la comprensión. Esto acelera la ejecución de una comprensión hasta dos veces. Consulte **PEP 709** para obtener más detalles.

Las variables de iteración de comprensión permanecen aisladas y no sobrescriben una variable del mismo nombre en el alcance externo, ni son visibles después de la comprensión. La incorporación da como resultado algunos cambios de comportamiento visibles:

- Ya no hay un marco separado para la comprensión en los rastreos, y el rastreo/elaboración de perfiles ya no muestra la comprensión como una llamada a función.
- El módulo `symtable` ya no producirá tablas de símbolos secundarios para cada comprensión; en cambio, los locales de comprensión se incluirán en la tabla de símbolos de la función principal.
- Llamar a `locals()` dentro de una comprensión ahora incluye variables externas a la comprensión y ya no incluye la variable sintética `.0` para el «argumento» de comprensión.
- Una comprensión que itera directamente sobre `locals()` (por ejemplo, `[k for k in locals()]`) puede ver «RuntimeError: el diccionario cambió de tamaño durante la iteración» cuando se ejecuta bajo seguimiento (por ejemplo, medición de cobertura de código). Este es el mismo comportamiento que ya se ha visto, por ejemplo, en `for k in locals():`. Para evitar el error, primero cree una lista de claves para iterar: `keys = list(locals()); [k for k in keys]`.

(Aportado por Carl Meyer y Vladimir Matveev en **PEP 709**.)

2.7 Mensajes de error mejorados

- Ahora se sugieren potencialmente módulos de la biblioteca estándar como parte de los mensajes de error que muestra el intérprete cuando un `NameError` se eleva al nivel superior. (Aportado por Pablo Galindo en gh-98254.)

```
>>> sys.version_info
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sys' is not defined. Did you forget to import 'sys'?
```

- Mejora la sugerencia de error para excepciones `NameError` para instancias. Ahora, si se genera un `NameError` en un método y la instancia tiene un atributo que es exactamente igual al nombre en la excepción, la sugerencia incluirá `self.<NAME>` en lugar de la coincidencia más cercana en el alcance del método. (Aportado por Pablo Galindo en gh-99139.)

```
>>> class A:
...     def __init__(self):
...         self.blech = 1
...
...     def foo(self):
```

(continúe en la próxima página)

(proviene de la página anterior)

```
...         somethin = blech
...
>>> A().foo()
Traceback (most recent call last):
  File "<stdin>", line 1
    somethin = blech
               ^^^^^
NameError: name 'blech' is not defined. Did you mean: 'self.blech'?
```

- Mejora el mensaje de error `SyntaxError` cuando el usuario escribe `import x from y` en lugar de `from y import x`. (Aportado por Pablo Galindo en gh-98931.)

```
>>> import a.y.z from b.y.z
Traceback (most recent call last):
  File "<stdin>", line 1
    import a.y.z from b.y.z
    ^^^^^^^^^^^^^^^^^^^^^^
SyntaxError: Did you mean to use 'from ... import ...' instead?
```

- Las excepciones `ImportError` generadas por sentencias `from <module> import <name>` fallidas ahora incluyen sugerencias para el valor de `<name>` según los nombres disponibles en `<module>`. (Aportado por Pablo Galindo en gh-91058.)

```
>>> from collections import chainmap
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'chainmap' from 'collections'. Did you mean:
↪'ChainMap'?
```

3 Nuevas funciones relacionadas con las sugerencias de escritura

Esta sección cubre los cambios principales que afectan a **type hints** y al módulo `typing`.

3.1 PEP 692: uso de `TypedDict` para una escritura `**kwargs` más precisa

Escribir `**kwargs` en una firma de función introducida por **PEP 484** permitía anotaciones válidas solo en los casos en que todos los `**kwargs` eran del mismo tipo.

PEP 692 especifica una forma más precisa de escribir `**kwargs` basándose en diccionarios escritos:

```
from typing import TypedDict, Unpack

class Movie(TypedDict):
  name: str
  year: int

def foo(**kwargs: Unpack[Movie]): ...
```

Consulte **PEP 692** para obtener más detalles.

(Aportado por Franek Magiera en gh-103629.)

3.2 PEP 698: Decorador override para escritura estática

Se ha agregado un nuevo decorador `typing.override()` al módulo `typing`. Indica a los inspectores de tipo que el método está destinado a anular un método en una superclase. Esto permite a los verificadores de tipos detectar errores cuando un método destinado a anular algo en una clase base en realidad no lo hace.

Ejemplo:

```
from typing import override

class Base:
  def get_color(self) -> str:
    return "blue"

class GoodChild(Base):
  @override  # ok: overrides Base.get_color
  def get_color(self) -> str:
    return "yellow"

class BadChild(Base):
  @override  # type checker error: does not override Base.get_color
  def get_colour(self) -> str:
    return "red"
```

Consulte **PEP 698** para obtener más detalles.

(Aportado por Steven Troxler en gh-101561.)

4 Otros cambios del lenguaje

- El analizador ahora genera `SyntaxError` al analizar el código fuente que contiene bytes nulos. (Aportado por Pablo Galindo en gh-96670.)
- Un par de caracteres de barra invertida que no es una secuencia de escape válida ahora genera un `SyntaxWarning`, en lugar de `DeprecationWarning`. Por ejemplo, `re.compile("\d+\.\d+")` ahora emite un `SyntaxWarning` (`"\d"` es una secuencia de escape no válida; utilice cadenas sin formato para la expresión regular: `re.compile(r"\d+\.\d+")`). En una versión futura de Python, eventualmente se generará `SyntaxError`, en lugar de `SyntaxWarning`. (Aportado por Victor Stinner en gh-98401.)
- Los escapes octales con un valor mayor que `0o377` (por ejemplo, `"\477"`), obsoletos en Python 3.11, ahora producen un `SyntaxWarning`, en lugar de `DeprecationWarning`. En una futura versión de Python, eventualmente serán un `SyntaxError`. (Aportado por Victor Stinner en gh-98401.)
- Las variables utilizadas en la parte de destino de las comprensiones que no están almacenadas ahora se pueden usar en expresiones de asignación (`:=`). Por ejemplo, en `[(b := 1) for a, b.prop in some_iter]`, ahora se permite la asignación a `b`. Tenga en cuenta que la asignación de variables almacenadas en la parte de destino de las comprensiones (como `a`) todavía no está permitida, según **PEP 572**. (Aportado por Nikita Sobolev en gh-100581.)
- Las excepciones generadas en el método `__set_name__` de una clase o tipo ya no están incluidas en un `RuntimeError`. La información de contexto se agrega a la excepción como una nota **PEP 678**. (Aportado por Irit Katriel en gh-77757.)
- Cuando una construcción `try-except*` maneja todo el `ExceptionGroup` y genera otra excepción, esa excepción ya no está incluida en un `ExceptionGroup`. También cambió en la versión 3.11.4. (Aportado por Irit Katriel en gh-103590.)

- El recolector de basura ahora se ejecuta solo en el mecanismo de interrupción de evaluación del ciclo de evaluación del código de bytes de Python en lugar de en las asignaciones de objetos. El GC también se puede ejecutar cuando se llama a `PyErr_CheckSignals()`, por lo que las extensiones C que necesitan ejecutarse durante un período prolongado sin ejecutar ningún código Python también tienen la posibilidad de ejecutar el GC periódicamente. (Aportado por Pablo Galindo en gh-97922.)
- Todos los invocables integrados y de extensión que esperan parámetros booleanos ahora aceptan argumentos de cualquier tipo en lugar de solo `bool` y `int`. (Contribución de Serhiy Storchaka en gh-60203.)
- `memoryview` ahora admite el tipo medio flotante (el código de formato «e»). (Contribución de Donghee Na y Antoine Pitrou en gh-90751.)
- Los objetos `slice` ahora son hasheables, lo que permite usarlos como claves de dictado y elementos de configuración. (Contribución de Will Bradshaw, Furkan Onder y Raymond Hettinger en gh-101264.)
- `sum()` ahora utiliza la suma de Neumaier para mejorar la precisión y la conmutatividad al sumar flotantes o enteros y flotantes mixtos. (Aportado por Raymond Hettinger en gh-100425.)
- `ast.parse()` ahora genera `SyntaxError` en lugar de `ValueError` al analizar el código fuente que contiene bytes nulos. (Aportado por Pablo Galindo en gh-96670.)
- Los métodos de extracción en `tarfile` y `shutil.unpack_archive()` tienen un nuevo argumento, *filter*, que permite limitar funciones tar que puedan resultar sorprendentes o peligrosas, como la creación de archivos fuera del directorio de destino. Consulte tarfile extraction filters para obtener más detalles. En Python 3.14, el valor predeterminado cambiará a `'data'`. (Aportado por Petr Viktorin en **PEP 706**.)
- Las instancias `types.MappingProxyType` ahora son hasheables si el mapeo subyacente es hasheable. (Contribución de Serhiy Storchaka en gh-87995.)
- Agregado support for the perf profiler a través de la nueva variable de entorno `PYTHONPERFSUPPORT` y la opción de línea de comandos `-X perf`, así como las nuevas funciones `sys.activate_stack_trampoline()`, `sys.deactivate_stack_trampoline()` y `sys.is_stack_trampoline_active()`. (Diseño de Pablo Galindo. Contribución de Pablo Galindo y Christian Heimes con contribuciones de Gregory P. Smith [Google] y Mark Shannon en gh-96123.)

5 Nuevos módulos

- Ninguno.

6 Módulos mejorados

6.1 array

- La clase `array.array` ahora admite subíndices, lo que la convierte en generic type. (Aportado por Jelle Zijlstra en gh-98658.)

6.2 asincio

- Se ha mejorado significativamente el rendimiento de escritura en sockets en `asyncio`. `asyncio` ahora evita copias innecesarias al escribir en sockets y usa `sendmsg()` si la plataforma lo admite. (Aportado por Kumar Aditya en gh-91166.)
- Agregado las funciones `asyncio.eager_task_factory()` y `asyncio.create_eager_task_factory()` para permitir la opción de un bucle de eventos para la ejecución de tareas entusiastas, lo que hace que algunos casos de uso sean de 2 a 5 veces más rápidos. (Contribución de Jacob Bower e Itamar Oren en gh-102853, gh-104140 y gh-104138)
- En Linux, `asyncio` usa `asyncio.PidfdChildWatcher` de forma predeterminada si `os.pidfd_open()` está disponible y es funcional en lugar de `asyncio.ThreadedChildWatcher`. (Aportado por Kumar Aditya en gh-98024.)
- El bucle de eventos ahora utiliza el mejor observador de hijos disponible para cada plataforma (`asyncio.PidfdChildWatcher` si es compatible y `asyncio.ThreadedChildWatcher` en caso contrario), por lo que no se recomienda configurar manualmente un observador de hijos. (Aportado por Kumar Aditya en gh-94597.)
- Agregado el parámetro *loop_factory* a `asyncio.run()` para permitir especificar una fábrica de bucle de eventos personalizada. (Aportado por Kumar Aditya en gh-99388.)
- Agregado la implementación C de `asyncio.current_task()` para una aceleración de 4x-6x. (Aportado por Itamar Oren y Pranav Thulasiram Bhat en gh-100344.)
- `asyncio.iscoroutine()` ahora devuelve `False` para generadores, ya que `asyncio` no admite rutinas heredadas basadas en generadores. (Aportado por Kumar Aditya en gh-102748.)
- `asyncio.wait()` y `asyncio.as_completed()` ahora aceptan generadores que generan tareas. (Aportado por Kumar Aditya en gh-78530.)

6.3 calendar

- Agregado las enumeraciones `calendar.Month` y `calendar.Day` que definen los meses del año y los días de la semana. (Aportado por el Príncipe Roshan en gh-103636.)

6.4 csv

- Agregado indicadores `csv.QUOTE_NOTNULL` y `csv.QUOTE_STRINGS` para proporcionar un control más detallado de `None` y cadenas vacías por objetos `csv.writer`.

6.5 dis

- Los códigos de operación de pseudoinstrucción (que son utilizados por el compilador pero que no aparecen en el código de bytes ejecutable) ahora están expuestos en el módulo `dis`. `HAVE_ARGUMENT` sigue siendo relevante para códigos de operación reales, pero no es útil para pseudoinstrucciones. Utilice la nueva colección `dis.hasarg` en su lugar. (Aportado por Irit Katriel en gh-94216.)
- Agregado la colección `dis.hasexc` para indicar instrucciones que establecen un controlador de excepciones. (Aportado por Irit Katriel en gh-94216.)

6.6 fractions

- Los objetos de tipo `fractions.Fraction` ahora admiten formato de estilo flotante. (Aportado por Mark Dickinson en gh-100161.)

6.7 importlib.resources

- `importlib.resources.as_file()` ahora admite directorios de recursos. (Aportado por Jason R. Coombs en gh-97930.)
- Rename first parameter of `importlib.resources.files()` to *anchor*. (Contributed by Jason R. Coombs in gh-100598.)

6.8 inspect

- Add `inspect.markcoroutinefunction()` to mark sync functions that return a coroutine for use with `inspect.iscoroutinefunction()`. (Contributed by Carlton Gibson in gh-99247.)
- Agregado `inspect.getasyncgenstate()` y `inspect.getasyncgenlocals()` para determinar el estado actual de los generadores asíncronos. (Aportado por Thomas Krennwallner en gh-79940.)
- The performance of `inspect.getattr_static()` has been considerably improved. Most calls to the function should be at least 2x faster than they were in Python 3.11. (Contributed by Alex Waygood in gh-103193.)

6.9 itertools

- Agregado `itertools.batched()` para recopilar en tuplas de tamaño par donde el último lote puede ser más corto que el resto. (Aportado por Raymond Hettinger en gh-98363.)

6.10 math

- Agregado `math.sumprod()` para calcular una suma de productos. (Aportado por Raymond Hettinger en gh-100485.)
- Extend `math.nextafter()` to include a *steps* argument for moving up or down multiple steps at a time. (Contributed by Matthias Goergens, Mark Dickinson, and Raymond Hettinger in gh-94906.)

6.11 os

- Agregado `os.PIDFD_NONBLOCK` para abrir un descriptor de archivo para un proceso con `os.pidfd_open()` en modo sin bloqueo. (Aportado por Kumar Aditya en gh-93312.)
- `os.DirEntry` ahora incluye un método `os.DirEntry.is_junction()` para verificar si la entrada es un cruce. (Aportado por Charles Machalow en gh-99547.)
- Agregado funciones `os.listdrives()`, `os.listvolumes()` y `os.listmounts()` en Windows para enumerar unidades, volúmenes y puntos de montaje. (Aportado por Steve Dower en gh-102519.)
- `os.stat()` y `os.lstat()` ahora son más precisos en Windows. El campo `st_birthtime` ahora se completará con la hora de creación del archivo, y `st_ctime` está obsoleto pero aún contiene la hora de creación (pero en el futuro devolverá el último cambio de metadatos, para mantener la coherencia con otras plataformas). `st_dev` puede tener hasta 64 bits y `st_ino` hasta 128 bits dependiendo de su sistema de archivos, y `st_rdev`

siempre está configurado en cero en lugar de valores incorrectos. Ambas funciones pueden ser significativamente más rápidas en las versiones más recientes de Windows. (Aportado por Steve Dower en gh-99726.)

6.12 os.path

- Agregado `os.path.isjunction()` para verificar si una ruta determinada es un cruce. (Aportado por Charles Machalow en gh-99547.)
- Agregado `os.path.splitroot()` para dividir una ruta en una tríada `(drive, root, tail)`. (Aportado por Barney Gale en gh-101000.)

6.13 pathlib

- Agregado soporte para subclasificar `pathlib.PurePath` y `pathlib.Path`, además de sus variantes específicas de Posix y Windows. Las subclases pueden anular el método `pathlib.PurePath.with_segments()` para pasar información entre instancias de ruta.
- Agregado `pathlib.Path.walk()` para recorrer los árboles de directorios y generar todos los nombres de archivos o directorios dentro de ellos, similar a `os.walk()`. (Aportado por Stanislav Zmiev en gh-90385.)
- Agregado el parámetro opcional *walk_up* a `pathlib.PurePath.relative_to()` para permitir la inserción de entradas `..` en el resultado; este comportamiento es más consistente con `os.path.relpath()`. (Aportado por Domenico Ragusa en gh-84538.)
- Agregado `pathlib.Path.is_junction()` como proxy a `os.path.isjunction()`. (Aportado por Charles Machalow en gh-99547.)
- Agregado el parámetro opcional *case_sensitive* a `pathlib.Path.glob()`, `pathlib.Path.rglob()` y `pathlib.PurePath.match()` para hacer coincidir la distinción entre mayúsculas y minúsculas de la ruta, lo que permite un control más preciso sobre el proceso de coincidencia.

6.14 pbd

- Agregado variables convenientes para mantener valores temporalmente para la sesión de depuración y proporcionar acceso rápido a valores como el marco actual o el valor de retorno. (Aportado por Tian Gao en gh-103693.)

6.15 random

- Agregado `random.binomialvariate()`. (Aportado por Raymond Hettinger en gh-81620.)
- Agregado un valor predeterminado de `lambd=1.0` a `random.expovariate()`. (Aportado por Raymond Hettinger en gh-100234.)

6.16 shutil

- `shutil.make_archive()` ahora pasa el argumento *root_dir* a archivadores personalizados que lo admiten. En este caso, ya no cambia temporalmente el directorio de trabajo actual del proceso a *root_dir* para realizar el archivado. (Aportado por Serhiy Storchaka en gh-74696.)
- `shutil.rmtree()` now accepts a new argument *onexc* which is an error handler like *onerror* but which expects an exception instance rather than a *(typ, val, tb)* triplet. *onerror* is deprecated. (Contributed by Irit Katriel in gh-102828.)
- `shutil.which()` ahora consulta la variable de entorno *PATHEXT* para encontrar coincidencias dentro de *PATH* en Windows incluso cuando el *cmd* dado incluye un componente de directorio. (Aportado por Charles Machalow en gh-103179.)

  `shutil.which()` llamará a `NeedCurrentDirectoryForExePathW` cuando solicite ejecutables en Windows para determinar si el directorio de trabajo actual debe anteponerse a la ruta de búsqueda. (Aportado por Charles Machalow en gh-103179.)

  `shutil.which()` devolverá una ruta que coincida con *cmd* con un componente de `PATHEXT` antes de una coincidencia directa en otra parte de la ruta de búsqueda en Windows. (Aportado por Charles Machalow en gh-103179.)

6.17 sqlite3

- Agregado un command-line interface. (Aportado por Erlend E. Aasland en gh-77617.)
- Agregado el atributo `sqlite3.Connection.autocommit` a `sqlite3.Connection` y el parámetro *autocommit* a `sqlite3.connect()` para controlar transaction handling compatible con **PEP 249**. (Aportado por Erlend E. Aasland en gh-83638.)
- Agregado el parámetro de solo palabra clave *entrypoint* a `sqlite3.Connection.load_extension()`, para anular el punto de entrada de la extensión SQLite. (Aportado por Erlend E. Aasland en gh-103015.)
- Agregado `sqlite3.Connection.getconfig()` y `sqlite3.Connection.setconfig()` a `sqlite3.Connection` para realizar cambios de configuración en una conexión de base de datos. (Aportado por Erlend E. Aasland en gh-103489.)

6.18 statistics

- Se extendió `statistics.correlation()` para incluirlo como método `ranked` para calcular la correlación de Spearman de datos clasificados. (Aportado por Raymond Hettinger en gh-95861.)

6.19 sys

- Agregado el espacio de nombres `sys.monitoring` para exponer la nueva API de supervisión *PEP 669*. (Aportado por Mark Shannon en gh-103082.)
- Agregado `sys.activate_stack_trampoline()` y `sys.deactivate_stack_trampoline()` para activar y desactivar los trampolines del perfilador de pila, y `sys.is_stack_trampoline_active()` para consultar si los trampolines del perfilador de pila están activos. (Contribuido por Pablo Galindo y Christian Heimes con contribuciones de Gregory P. Smith [Google] y Mark Shannon en gh-96123.)
- Agregado `sys.last_exc` que contiene la última excepción no controlada que se generó (para casos de uso de depuración post-mortem). Deje obsoletos los tres campos que tienen la misma información en su formato heredado: `sys.last_type`, `sys.last_value` y `sys.last_traceback`. (Aportado por Irit Katriel en gh-102778.)

- `sys._current_exceptions()` ahora devuelve una asignación de thread-id a una instancia de excepción, en lugar de a una tupla `(typ, exc, tb)`. (Aportado por Irit Katriel en gh-103176.)
- `sys.setrecursionlimit()` y `sys.getrecursionlimit()`. El límite de recursividad ahora se aplica sólo al código Python. Las funciones integradas no utilizan el límite de recursividad, pero están protegidas por un mecanismo diferente que evita que la recursividad provoque un fallo de la máquina virtual.

6.20 tempfile

- La función `tempfile.NamedTemporaryFile` tiene un nuevo parámetro opcional *delete_on_close* (Aportado por Evgeny Zorin en gh-58451.)
- `tempfile.mkdtemp()` ahora siempre devuelve una ruta absoluta, incluso si el argumento proporcionado al parámetro *dir* es una ruta relativa.

6.21 threading

- Agregado `threading.settrace_all_threads()` y `threading.setprofile_all_threads()` que permiten configurar funciones de seguimiento y creación de perfiles en todos los hilos en ejecución además del que realiza la llamada. (Aportado por Pablo Galindo en gh-93503.)

6.22 tkinter

- `tkinter.Canvas.coords()` ahora aplana sus argumentos. Ahora acepta no sólo coordenadas como argumentos separados (`x1, y1, x2, y2, ...`) y una secuencia de coordenadas (`[x1, y1, x2, y2, ...]`), sino también coordenadas agrupadas en pares (`(x1, y1), (x2, y2), ...` y `[(x1, y1), (x2, y2), ...]`), como los métodos `create_*()`. (Aportado por Serhiy Storchaka en gh-94473.)

6.23 tokenize

- El módulo `tokenize` incluye los cambios introducidos en **PEP 701**. (Aportado por Marta Gómez Macías y Pablo Galindo en gh-102856.) Consulte *Portar a Python 3.12* para obtener más información sobre los cambios en el módulo `tokenize`.

6.24 types

- Agregado `types.get_original_bases()` para permitir una mayor introspección de user-defined-generics cuando esté subclasificado. (Contribución de James Hilton-Balfe y Alex Waygood en gh-101827.)

6.25 typing

- Las comprobaciones de `isinstance()` con `runtime-checkable protocols` ahora usan `inspect.getattr_static()` en lugar de `hasattr()` para buscar si existen atributos. Esto significa que los descriptores y los métodos `__getattr__()` ya no se evalúan inesperadamente durante las comprobaciones de `isinstance()` con protocolos verificables en tiempo de ejecución. Sin embargo, también puede significar que algunos objetos que solían considerarse instancias de un protocolo verificable en tiempo de ejecución ya no se consideran instancias de ese protocolo en Python 3.12+, y viceversa. Es poco probable que la mayoría de los usuarios se vean afectados por este cambio. (Contribuido por Alex Waygood en gh-102433.)

- Los miembros de un protocolo verificable en tiempo de ejecución ahora se consideran «congelados» en tiempo de ejecución tan pronto como se crea la clase. La aplicación de parches de atributos en un protocolo verificable en tiempo de ejecución seguirá funcionando, pero no tendrá ningún impacto en las comprobaciones `isinstance()` que comparan objetos con el protocolo. Por ejemplo:

```
>>> from typing import Protocol, runtime_checkable
>>> @runtime_checkable
... class HasX(Protocol):
...     x = 1
...
>>> class Foo: ...
...
>>> f = Foo()
>>> isinstance(f, HasX)
False
>>> f.x = 1
>>> isinstance(f, HasX)
True
>>> HasX.y = 2
>>> isinstance(f, HasX)  # unchanged, even though HasX now also has a "y"␣
↪attribute
True
```

  Este cambio se realizó para acelerar las comprobaciones de `isinstance()` en protocolos verificables en tiempo de ejecución.

- The performance profile of `isinstance()` checks against `runtime-checkable protocols` has changed significantly. Most `isinstance()` checks against protocols with only a few members should be at least 2x faster than in 3.11, and some may be 20x faster or more. However, `isinstance()` checks against protocols with many members may be slower than in Python 3.11. (Contributed by Alex Waygood in gh-74690 and gh-103193.)

- Todas las clases `typing.TypedDict` y `typing.NamedTuple` ahora tienen el atributo `__orig_bases__`. (Aportado por Adrián García Badaracco en gh-103699.)

- Agregado el parámetro `frozen_default` a `typing.dataclass_transform()`. (Aportado por Erik De Bonte en gh-99957.)

6.26 unicodedata

- La base de datos Unicode se ha actualizado a la versión 15.0.0. (Aportado por Benjamin Peterson en gh-96734).

6.27 unittest

Agregado una opción de línea de comando `--durations`, que muestra los N casos de prueba más lentos:

```
python3 -m unittest --durations=3 lib.tests.test_threading
.....
Slowest test durations
----------------------------------------------------------------------
1.210s     test_timeout (Lib.test.test_threading.BarrierTests)
1.003s     test_default_timeout (Lib.test.test_threading.BarrierTests)
0.518s     test_timeout (Lib.test.test_threading.EventTests)

(0.000 durations hidden.  Use -v to show these durations.)
----------------------------------------------------------------------
```

(continúe en la próxima página)

(proviene de la página anterior)

```
Ran 158 tests in 9.869s

OK (skipped=3)
```

(Aportado por Giampaolo Rodola en gh-48330)

6.28 uuid

- Agregado un command-line interface. (Aportado por Adam Chhina en gh-88597.)

7 Optimizaciones

- Elimine los miembros `wstr` y `wstr_length` de los objetos Unicode. Reduce el tamaño del objeto en 8 o 16 bytes en una plataforma de 64 bits. (**PEP 623**) (Aportado por Inada Naoki en gh-92536.)
- Agregado soporte experimental para usar el optimizador binario BOLT en el proceso de compilación, lo que mejora el rendimiento entre un 1% y un 5%. (Contribuido por Kevin Modzelewski en gh-90536 y sintonizado por Donghee Na en gh-101525)
- Acelere la sustitución de expresiones regulares (funciones `re.sub()` y `re.subn()` y los métodos `re.Pattern` correspondientes) para cadenas de reemplazo que contienen referencias de grupo entre 2 y 3 veces. (Aportado por Serhiy Storchaka en gh-91524.)
- Acelere la creación de `asyncio.Task` aplazando el costoso formato de cadenas. (Aportado por Itamar Oren en gh-103793.)
- Las funciones `tokenize.tokenize()` y `tokenize.generate_tokens()` son hasta un 64 % más rápidas como efecto secundario de los cambios necesarios para cubrir **PEP 701** en el módulo `tokenize`. (Aportado por Marta Gómez Macías y Pablo Galindo en gh-102856.)
- Acelere las llamadas al método `super()` y las cargas de atributos mediante la nueva instrucción `LOAD_SUPER_ATTR`. (Contribución de Carl Meyer y Vladimir Matveev en gh-103497.)

8 Cambios en el código de bytes de CPython

- Elimine la instrucción `LOAD_METHOD`. Se ha fusionado con `LOAD_ATTR`. `LOAD_ATTR` ahora se comportará como la antigua instrucción `LOAD_METHOD` si el bit bajo de su oparg está configurado. (Contribuido por Ken Jin en gh-93429.)
- Elimine las instrucciones `JUMP_IF_FALSE_OR_POP` y `JUMP_IF_TRUE_OR_POP`. (Aportado por Irit Katriel en gh-102859.)
- Elimine la instrucción `PRECALL`. (Aportado por Mark Shannon en gh-92925.)
- Agregado las instrucciones `BINARY_SLICE` y `STORE_SLICE`. (Aportado por Mark Shannon en gh-94163.)
- Agregado las instrucciones `CALL_INTRINSIC_1`. (Aportado por Mark Shannon en gh-99005.)
- Agregado la instrucción `CALL_INTRINSIC_2`. (Aportado por Irit Katriel en gh-101799.)
- Agregado la instrucción `CLEANUP_THROW`. (Aportado por Brandt Bucher en gh-90997.)
- Agregado la instrucción `END_SEND`. (Aportado por Mark Shannon en gh-103082.)

- Agregada la instrucción `LOAD_FAST_AND_CLEAR` como parte de la implementación de **PEP 709**. (Aportado por Carl Meyer en gh-101441.)
- Agregado la instrucción `LOAD_FAST_CHECK`. (Aportado por Dennis Sweeney en gh-93143.)
- Agregado los códigos de operación `LOAD_FROM_DICT_OR_DEREF`, `LOAD_FROM_DICT_OR_GLOBALS` y `LOAD_LOCALS` como parte de la implementación de **PEP 695**. Elimine el código de operación `LOAD_CLASSDEREF`, que se puede reemplazar con `LOAD_LOCALS` más `LOAD_FROM_DICT_OR_DEREF`. (Aportado por Jelle Zijlstra en gh-103764.)
- Agregado la instrucción `LOAD_SUPER_ATTR`. (Contribución de Carl Meyer y Vladimir Matveev en gh-103497.)
- Agregado la instrucción `RETURN_CONST`. (Aportado por Wenyang Wang en gh-101632.)

9 Demostraciones y herramientas

- Elimine el directorio `Tools/demo/` que contenía scripts de demostración antiguos. Se puede encontrar una copia en el old-demos project. (Aportado por Victor Stinner en gh-97681.)
- Elimine los scripts de ejemplo obsoletos del directorio `Tools/scripts/`. Se puede encontrar una copia en el old-demos project. (Aportado por Victor Stinner en gh-97669.)

10 Obsoleto

- `argparse`: Los parámetros *type*, *choices* y *metavar* de `argparse.BooleanOptionalAction` están obsoletos y se eliminarán en 3.14. (Contribución de Nikita Sobolev en gh-92248.)
- `ast`: las siguientes características de `ast` han quedado obsoletas en la documentación desde Python 3.8, ahora provocan que se emita un `DeprecationWarning` en tiempo de ejecución cuando se accede a ellas o se usan, y se eliminarán en Python 3.14:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

  Utilice `ast.Constant` en su lugar. (Aportado por Serhiy Storchaka en gh-90953.)
- `asyncio`:
  - Las clases de vigilancia secundaria `asyncio.MultiLoopChildWatcher`, `asyncio.FastChildWatcher`, `asyncio.AbstractChildWatcher` y `asyncio.SafeChildWatcher` están en desuso y se eliminarán en Python 3.14. (Aportado por Kumar Aditya en gh-94597.)
  - `asyncio.set_child_watcher()`, `asyncio.get_child_watcher()`, `asyncio.AbstractEventLoopPolicy.set_child_watcher()` y `asyncio.AbstractEventLoopPolicy.get_child_watcher()` están en desuso y se eliminarán en Python 3.14. (Aportado por Kumar Aditya en gh-94597.)
  - El método `get_event_loop()` de la política de bucle de eventos predeterminada ahora emite un `DeprecationWarning` si no hay ningún bucle de eventos establecido y decide crear uno. (Contribución de Serhiy Storchaka y Guido van Rossum en gh-100160.)

- `calendar`: las constantes `calendar.January` y `calendar.February` están obsoletas y reemplazadas por `calendar.JANUARY` y `calendar.FEBRUARY`. (Aportado por el Príncipe Roshan en gh-103636.)
- `collections.abc`: `collections.abc.ByteString` en desuso. Prefiere `Sequence` o `collections.abc.Buffer`. Para usar al escribir, prefiera una unión, como `bytes | bytearray` o `collections.abc.Buffer`. (Aportado por Shantanu Jain en gh-91896.)
- `datetime`: `utcnow()` y `utcfromtimestamp()` de `datetime.datetime` están en desuso y se eliminarán en una versión futura. En su lugar, utilice objetos que tengan en cuenta la zona horaria para representar las fechas y horas en UTC: respectivamente, llame a `now()` y `fromtimestamp()` con el parámetro *tz* establecido en `datetime.UTC`. (Aportado por Paul Ganssle en gh-103857.)
- `email`: obsoleto el parámetro *isdst* en `email.utils.localtime()`. (Aportado por Alan Williams en gh-72346.)
- `importlib.abc`: Obsoletas las siguientes clases, cuya eliminación está programada en Python 3.14:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`

  Utilice clases `importlib.resources.abc` en su lugar:
  - `importlib.resources.abc.Traversable`
  - `importlib.resources.abc.TraversableResources`

  (Contribución de Jason R. Coombs y Hugo van Kemenade en gh-93963.)
- `itertools`: obsoleta la compatibilidad con operaciones de copia, copia profunda y pickle, que no está documentada, es ineficiente, históricamente tiene errores e inconsistente. Esto se eliminará en 3.14 para lograr una reducción significativa en el volumen de código y la carga de mantenimiento. (Aportado por Raymond Hettinger en gh-101588.)
- `multiprocessing`: en Python 3.14, el método de inicio predeterminado de `multiprocessing` cambiará a uno más seguro en Linux, BSD y otras plataformas POSIX que no sean macOS donde `'fork'` es actualmente el predeterminado (gh-84559). Agregar una advertencia de tiempo de ejecución sobre esto se consideró demasiado perjudicial ya que se espera que a la mayoría del código no le importe. Utilice las API `get_context()` o `set_start_method()` para especificar explícitamente cuándo su código *requires* `'fork'`. Ver contexts and start methods.
- `pkgutil`: `pkgutil.find_loader()` y `pkgutil.get_loader()` están en desuso y se eliminarán en Python 3.14; utilice `importlib.util.find_spec()` en su lugar. (Contribución de Nikita Sobolev en gh-97850.)
- `pty`: el módulo tiene dos funciones `master_open()` y `slave_open()` no documentadas que han quedado obsoletas desde Python 2 pero que solo obtuvieron un `DeprecationWarning` adecuado en 3.12. Elimínelos en 3.14. (Aportado por Soumendra Ganguly y Gregory P. Smith en gh-85984.)
- `os`:
  - Los campos `st_ctime` devueltos por `os.stat()` y `os.lstat()` en Windows están en desuso. En una versión futura, contendrán la hora del último cambio de metadatos, de forma coherente con otras plataformas. Por ahora, todavía contienen la hora de creación, que también está disponible en el nuevo campo `st_birthtime`. (Aportado por Steve Dower en gh-99726.)
  - En plataformas POSIX, `os.fork()` ahora puede generar un `DeprecationWarning` cuando detecta una llamada desde un proceso multiproceso. Siempre ha habido una incompatibilidad fundamental con la plataforma POSIX al hacerlo. Incluso si dicho código *appeared* funciona. Agregamos la advertencia para crear conciencia, ya que los problemas encontrados por el código al hacer esto son cada vez más frecuentes. Consulte la documentación de `os.fork()` para obtener más detalles junto con this discussion on fork

being incompatible with threads para *why*. Ahora estamos exponiendo a los desarrolladores este problema de compatibilidad de plataforma de larga data.

Cuando aparece esta advertencia debido al uso de `multiprocessing` o `concurrent.futures`, la solución es utilizar un método de inicio de `multiprocessing` diferente, como `"spawn"` o `"forkserver"`.

- `shutil`: The *onerror* argument of `shutil.rmtree()` is deprecated; use *onexc* instead. (Contributed by Irit Katriel in gh-102828.)
- `sqlite3`:
  - default adapters and converters ahora están en desuso. En su lugar, utilice el sqlite3-adapter-converter-recipes y adáptelo a sus necesidades. (Aportado por Erlend E. Aasland en gh-90016.)
  - En `execute()`, `DeprecationWarning` ahora se emite cuando named placeholders se utiliza junto con los parámetros proporcionados como sequence en lugar de como `dict`. A partir de Python 3.14, el uso de marcadores de posición con nombre con parámetros proporcionados como una secuencia generará un `ProgrammingError`. (Aportado por Erlend E. Aasland en gh-101698.)
- `sys`: los campos `sys.last_type`, `sys.last_value` y `sys.last_traceback` están en desuso. Utilice `sys.last_exc` en su lugar. (Aportado por Irit Katriel en gh-102778.)
- `tarfile`: la extracción de archivos tar sin especificar *filter* está en desuso hasta Python 3.14, cuando el filtro `'data'` se convertirá en el predeterminado. Consulte tarfile-extraction-filter para obtener más detalles.
- `typing`:
  - `typing.Hashable` and `typing.Sized`, aliases for `collections.abc.Hashable` and `collections.abc.Sized` respectively, are deprecated. (gh-94309.)
  - `typing.ByteString`, en desuso desde Python 3.9, ahora provoca que se emita un `DeprecationWarning` cuando se utiliza. (Aportado por Alex Waygood en gh-91896.)
- `xml.etree.ElementTree`: el módulo ahora emite `DeprecationWarning` al probar el valor de verdad de un `xml.etree.ElementTree.Element`. Antes, la implementación de Python emitía `FutureWarning` y la implementación de C no emitía nada. (Aportado por Jacob Walls en gh-83122.)
- Las firmas de 3 argumentos (tipo, valor, rastreo) de `coroutine throw()`, `generator throw()` y `async generator throw()` están en desuso y es posible que se eliminen en una versión futura de Python. Utilice en su lugar las versiones de un solo argumento de estas funciones. (Aportado por Ofey Chan en gh-89874.)
- `DeprecationWarning` ahora se genera cuando `__package__` en un módulo difiere de `__spec__.parent` (anteriormente era `ImportWarning`). (Aportado por Brett Cannon en gh-65961.)
- La configuración de `__package__` o `__cached__` en un módulo está obsoleta y el sistema de importación dejará de configurarla o tomarla en consideración en Python 3.14. (Aportado por Brett Cannon en gh-65961.)
- El operador de inversión bit a bit (`~`) en bool está en desuso. Lanzará un error en Python 3.14. Utilice `not` para la negación lógica de bools. En el raro caso de que realmente necesite la inversión bit a bit del `int` subyacente, lo convierte a int explícitamente: `~int(x)`. (Aportado por Tim Hoffmann en gh-103487.)
- Accessing `co_lnotab` on code objects was deprecated in Python 3.10 via **PEP 626**, but it only got a proper `DeprecationWarning` in 3.12, therefore it will be removed in 3.14. (Contributed by Nikita Sobolev in gh-101866.)

10.1 Eliminación pendiente en Python 3.13

Los siguientes módulos y API quedaron obsoletos en versiones anteriores de Python y se eliminarán en Python 3.13.

Módulos (ver **PEP 594**):

- `aifc`
- `audioop`
- `cgi`
- `cgitb`
- `chunk`
- `crypt`
- `imghdr`
- `mailcap`
- `msilib`
- `nis`
- `nntplib`
- `ossaudiodev`
- `pipes`
- `sndhdr`
- `spwd`
- `sunau`
- `telnetlib`
- `uu`
- `xdrlib`

Otros módulos:

- `lib2to3` y el programa **`2to3`** (gh-84540)

APIs:

- `configparser.LegacyInterpolation` (gh-90765)
- `locale.resetlocale()` (gh-90817)
- `turtle.RawTurtle.settiltangle()` (gh-50096)
- `unittest.findTestCases()` (gh-50096)
- `unittest.getTestCaseNames()` (gh-50096)
- `unittest.makeSuite()` (gh-50096)
- `unittest.TestProgram.usageExit()` (gh-67048)
- `webbrowser.MacOSX` (gh-86421)
- Encadenamiento de descriptores `classmethod` (gh-89519)
- `importlib.resources` deprecated methods:
  - `contents()`

- `is_resource()`
- `open_binary()`
- `open_text()`
- `path()`
- `read_binary()`
- `read_text()`

Use `importlib.resources.files()` instead. Refer to importlib-resources: Migrating from Legacy (gh-106531)

10.2 Eliminación pendiente en Python 3.14

Las siguientes API han quedado obsoletas y se eliminarán en Python 3.14.

- `argparse`: Los parámetros *type*, *choices* y *metavar* de `argparse.BooleanOptionalAction`
- `ast`:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`
- `asyncio`:
  - `asyncio.MultiLoopChildWatcher`
  - `asyncio.FastChildWatcher`
  - `asyncio.AbstractChildWatcher`
  - `asyncio.SafeChildWatcher`
  - `asyncio.set_child_watcher()`
  - `asyncio.get_child_watcher()`,
  - `asyncio.AbstractEventLoopPolicy.set_child_watcher()`
  - `asyncio.AbstractEventLoopPolicy.get_child_watcher()`
- `collections.abc`: `collections.abc.ByteString`.
- `email`: el parámetro *isdst* en `email.utils.localtime()`.
- `importlib.abc`:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`
- `itertools`: soporte para operaciones de copia, copia profunda y pickle.
- `pkgutil`:
  - `pkgutil.find_loader()`

  - `pkgutil.get_loader()`.
- `pty`:
  - `pty.master_open()`
  - `pty.slave_open()`
- `shutil`: El argumento *onerror* de `shutil.rmtree()`
- `typing`: `typing.ByteString`
- `xml.etree.ElementTree`: Probando el valor de verdad de un `xml.etree.ElementTree.Element`.
- Los atributos `__package__` y `__cached__` en los objetos del módulo.
- The `co_lnotab` attribute of code objects.

10.3 Eliminación pendiente en Python 3.15

The following APIs have been deprecated and will be removed in Python 3.15.

APIs:

- `locale.getdefaultlocale()` (gh-90817)

10.4 Eliminación pendiente en versiones futuras

Las siguientes API quedaron obsoletas en versiones anteriores de Python y se eliminarán, aunque actualmente no hay una fecha programada para su eliminación.

- Código de formato `'u'` de `array` (gh-57281)
- `typing.Text` (gh-92332)
- Actualmente, Python acepta literales numéricos seguidos inmediatamente de palabras clave, por ejemplo `0in x`, `1or x`, `0if 1else 2`. Permite expresiones confusas y ambiguas como `[0x1for x in y]` (que puede interpretarse como `[0x1 for x in y]` o `[0x1f or x in y]`). Se genera una advertencia de sintaxis si el literal numérico va seguido inmediatamente de una de las palabras clave `and`, `else`, `for`, `if`, `in`, `is` y `or`. En una versión futura se cambiará a un error de sintaxis. (gh-87999)

11 Eliminado

11.1 asynchat y asyncore

- Estos dos módulos se eliminaron según lo programado en **PEP 594** y quedaron obsoletos en Python 3.6. Utilice `asyncio` en su lugar. (Aportado por Nikita Sobolev en gh-96580.)

11.2 configparser

- Varios nombres obsoletos en `configparser` en 3.2 se han eliminado según gh-89336:
  - `configparser.ParsingError` ya no tiene un atributo o argumento `filename`. Utilice el atributo y el argumento `source` en su lugar.
  - `configparser` ya no tiene una clase `SafeConfigParser`. Utilice en su lugar el nombre `ConfigParser` más corto.
  - `configparser.ConfigParser` ya no tiene un método `readfp`. Utilice `read_file()` en su lugar.

11.3 distutils

- Elimine el paquete `distutils`. Quedó obsoleto en Python 3.10 por **PEP 632** «Módulo distutils obsoleto». Para proyectos que todavía usan `distutils` y no se pueden actualizar a otra cosa, se puede instalar el proyecto `setuptools`: todavía proporciona `distutils`. (Aportado por Victor Stinner en gh-92584.)

11.4 ensurepip

- Retire la rueda de herramientas de configuración incluida en `ensurepip` y deje de instalar herramientas de configuración en entornos creados por `venv`.

  `pip (>= 22.1)` no requiere la instalación de herramientas de configuración en el entorno. Los paquetes basados en `setuptools` (y `distutils`) aún se pueden usar con `pip install`, ya que pip proporcionará `setuptools` en el entorno de compilación que utiliza para crear un paquete.

  `easy_install`, `pkg_resources`, `setuptools` y `distutils` ya no se proporcionan de forma predeterminada en entornos creados con `venv` o arrancados con `ensurepip`, ya que forman parte del paquete `setuptools`. Para proyectos que dependen de estos en tiempo de ejecución, el proyecto `setuptools` debe declararse como una dependencia e instalarse por separado (generalmente, usando pip).

  (Aportado por Pradyun Gedam en gh-95299.)

11.5 enum

- Elimine `EnumMeta.__getattr__` de `enum`, que ya no es necesario para acceder al atributo de enumeración. (Aportado por Ethan Furman en gh-95083.)

11.6 ftplib

- Elimine el atributo de clase `FTP_TLS.ssl_version` de `ftplib`: utilice el parámetro *context* en su lugar. (Aportado por Victor Stinner en gh-94172.)

11.7 gzip

- Elimine el atributo `filename` del `gzip.GzipFile` de `gzip`, en desuso desde Python 2.6; utilice el atributo `name` en su lugar. En modo de escritura, el atributo `filename` agregaba la extensión de archivo `'.gz'` si no estaba presente. (Aportado por Victor Stinner en gh-94196.)

11.8 hashlib

- Elimine la implementación pura de Python de `hashlib.pbkdf2_hmac()` de `hashlib`, obsoleta en Python 3.10. Python 3.10 y versiones posteriores requieren OpenSSL 1.1.1 (**PEP 644**): esta versión de OpenSSL proporciona una implementación C de `pbkdf2_hmac()` que es más rápida. (Aportado por Victor Stinner en gh-94199.)

11.9 importlib

- Muchas limpiezas anteriormente obsoletas en `importlib` ahora se han completado:
  - Se han eliminado las referencias y la compatibilidad con `module_repr()`. (Aportado por Barry Varsovia en gh-97850.)
  - Se han eliminado `importlib.util.set_package`, `importlib.util.set_loader` y `importlib.util.module_for_loader`. (Contribución de Brett Cannon y Nikita Sobolev en gh-65961 y gh-97850.)
  - Se eliminó la compatibilidad con las API `find_loader()` y `find_module()`. (Aportado por Barry Varsovia en gh-98040.)
  - Se han eliminado `importlib.abc.Finder`, `pkgutil.ImpImporter` y `pkgutil.ImpLoader`. (Aportado por Barry Varsovia en gh-98040.)

11.10 imp

- Se ha eliminado el módulo `imp`. (Aportado por Barry Varsovia en gh-98040.)

  Para migrar consulte la siguiente tabla de correspondencias:

imp	importlib
`imp.NullImporter`	Inserta `None` en `sys.path_importer_cache`
`imp.cache_from_s`	`importlib.util.cache_from_source()`
`imp.find_module(`	`importlib.util.find_spec()`
`imp.get_magic()`	`importlib.util.MAGIC_NUMBER`
`imp.get_suffixes`	`importlib.machinery.SOURCE_SUFFIXES`, `importlib.machinery.EXTENSION_SUFFIXES` y `importlib.machinery.BYTECODE_SUFFIXES`
`imp.get_tag()`	`sys.implementation.cache_tag`
`imp.load_module(`	`importlib.import_module()`
`imp.new_module(r`	`types.ModuleType(name)`
`imp.reload()`	`importlib.reload()`
`imp.source_from_`	`importlib.util.source_from_cache()`
`imp.load_source(`	*Ver abajo*

Reemplace `imp.load_source()` con:

```
import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(modname, filename)
    spec = importlib.util.spec_from_file_location(modname, filename,␣
↪loader=loader)
    module = importlib.util.module_from_spec(spec)
    # The module is always executed and not cached in sys.modules.
    # Uncomment the following line to cache the module.
    # sys.modules[module.__name__] = module
    loader.exec_module(module)
    return module
```

- Elimine las funciones y atributos de `imp` sin reemplazos:
  - Funciones no documentadas:
    - `imp.init_builtin()`
    - `imp.load_compiled()`
    - `imp.load_dynamic()`
    - `imp.load_package()`
  - `imp.lock_held()`, `imp.acquire_lock()`, `imp.release_lock()`: el esquema de bloqueo ha cambiado en Python 3.3 a bloqueos por módulo.

- Constantes `imp.find_module()`: `SEARCH_ERROR`, `PY_SOURCE`, `PY_COMPILED`, `C_EXTENSION`, `PY_RESOURCE`, `PKG_DIRECTORY`, `C_BUILTIN`, `PY_FROZEN`, `PY_CODERESOURCE`, `IMP_HOOK`.

11.11 io

- Elimine `io.OpenWrapper` y `_pyio.OpenWrapper` de `io`, obsoletos en Python 3.10: simplemente use `open()` en su lugar. La función `open()` (`io.open()`) es una función incorporada. Desde Python 3.10, `_pyio.open()` también es un método estático. (Aportado por Victor Stinner en gh-94169.)

11.12 locale

- Elimine la función `locale.format()` de `locale`, obsoleta en Python 3.7: use `locale.format_string()` en su lugar. (Aportado por Victor Stinner en gh-94226.)

11.13 smtpd

- The `smtpd` module has been removed according to the schedule in **PEP 594**, having been deprecated in Python 3.4.7 and 3.5.4. Use the aiosmtpd PyPI module or any other `asyncio`-based server instead. (Contributed by Oleg Iarygin in gh-93243.)

11.14 sqlite3

- Las siguientes características `sqlite3` no documentadas, obsoletas en Python 3.10, ahora se eliminan:
  - `sqlite3.enable_shared_cache()`
  - `sqlite3.OptimizedUnicode`

  Si se debe utilizar una caché compartida, abra la base de datos en modo URI utilizando el parámetro de consulta `cache=shared`.

  La fábrica de textos `sqlite3.OptimizedUnicode` ha sido un alias para `str` desde Python 3.3. El código que previamente configuró la fábrica de texto en `OptimizedUnicode` puede usar `str` explícitamente o confiar en el valor predeterminado que también es `str`.

  (Aportado por Erlend E. Aasland en gh-92548.)

11.15 ssl

- Elimine la función `ssl.RAND_pseudo_bytes()` de `ssl`, obsoleta en Python 3.6: use `os.urandom()` o `ssl.RAND_bytes()` en su lugar. (Aportado por Victor Stinner en gh-94199.)
- Elimine la función `ssl.match_hostname()`. Quedó obsoleto en Python 3.7. OpenSSL realiza coincidencias de nombres de host desde Python 3.7, Python ya no usa la función `ssl.match_hostname()`. (Aportado por Victor Stinner en gh-94199.)
- Remove the `ssl.wrap_socket()` function, deprecated in Python 3.7: instead, create a `ssl.SSLContext` object and call its `ssl.SSLContext.wrap_socket` method. Any package that still uses `ssl.wrap_socket()` is broken and insecure. The function neither sends a SNI TLS extension nor validates the server hostname. Code is subject to CWE-295 (Improper Certificate Validation). (Contributed by Victor Stinner in gh-94199.)

11.16 unittest

- Elimine muchas características de `unittest` que están en desuso desde hace mucho tiempo:
  - Varios alias del método `TestCase`:

Alias obsoleto	Nombre del método	En desuso en
`failUnless`	`assertTrue()`	3.1
`failIf`	`assertFalse()`	3.1
`failUnlessEqual`	`assertEqual()`	3.1
`failIfEqual`	`assertNotEqual()`	3.1
`failUnlessAlmostEqual`	`assertAlmostEqual()`	3.1
`failIfAlmostEqual`	`assertNotAlmostEqual()`	3.1
`failUnlessRaises`	`assertRaises()`	3.1
`assert_`	`assertTrue()`	3.2
`assertEquals`	`assertEqual()`	3.2
`assertNotEquals`	`assertNotEqual()`	3.2
`assertAlmostEquals`	`assertAlmostEqual()`	3.2
`assertNotAlmostEquals`	`assertNotAlmostEqual()`	3.2
`assertRegexpMatches`	`assertRegex()`	3.2
`assertRaisesRegexp`	`assertRaisesRegex()`	3.2
`assertNotRegexpMatches`	`assertNotRegex()`	3.5

    Puede utilizar https://github.com/isidentical/teyit para modernizar automáticamente sus pruebas unitarias.
  - Método `assertDictContainsSubset` de `TestCase` indocumentado y roto (obsoleto en Python 3.2).
  - Parámetro `TestLoader.loadTestsFromModule` no documentado *use_load_tests* (obsoleto e ignorado desde Python 3.2).
  - Un alias de la clase `TextTestResult`: `_TextTestResult` (obsoleto en Python 3.2).

  (Contribución de Serhiy Storchaka en gh-89325.)

11.17 webbrowser

- Elimina la compatibilidad con navegadores obsoletos de `webbrowser`. Los navegadores eliminados incluyen: Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird y Firefox versiones 35 e inferiores (gh-102871).

11.18 xml.etree.ElementTree

- Elimine el método `ElementTree.Element.copy()` de la implementación pura de Python, obsoleto en Python 3.10, use la función `copy.copy()` en su lugar. La implementación C de `xml.etree.ElementTree` no tiene ningún método `copy()`, solo un método `__copy__()`. (Aportado por Victor Stinner en gh-94383.)

11.19 zipimport

- Elimine los métodos `find_loader()` y `find_module()` de `zipimport`, obsoletos en Python 3.10: use el método `find_spec()` en su lugar. Consulte **PEP 451** para conocer el fundamento. (Aportado por Victor Stinner en gh-94379.)

11.20 Otros

- Elimine la regla `suspicious` de la documentación `Makefile` y `Doc/tools/rstlint.py`, ambas a favor de sphinx-lint. (Contribución de Julien Palard en gh-98179.)
- Elimine los parámetros *keyfile* y *certfile* de los módulos `ftplib`, `imaplib`, `poplib` y `smtplib`, y los parámetros *key_file*, *cert_file* y *check_hostname* del módulo `http.client`, todos obsoletos desde Python 3.6. Utilice el parámetro *context* (*ssl_context* en `imaplib`) en su lugar. (Aportado por Victor Stinner en gh-94172.)
- Elimine los trucos de compatibilidad `Jython` de varios módulos y pruebas stdlib. (Aportado por Nikita Sobolev en gh-99482.)
- Elimine el indicador `_use_broken_old_ctypes_structure_semantics_` del módulo `ctypes`. (Aportado por Nikita Sobolev en gh-99285.)

12 Portar a Python 3.12

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

12.1 Cambios en la API de Python

- Ahora se aplican reglas más estrictas para las referencias numéricas de grupos y los nombres de grupos en expresiones regulares. Ahora sólo se acepta como referencia numérica la secuencia de dígitos ASCII. El nombre del grupo en patrones de bytes y cadenas de reemplazo ahora solo puede contener letras y dígitos ASCII y guiones bajos. (Contribución de Serhiy Storchaka en gh-91760.)
- Elimine la funcionalidad `randrange()` en desuso desde Python 3.10. Anteriormente, `randrange(10.0)` se convertía sin pérdidas a `randrange(10)`. Ahora, genera un `TypeError`. Además, la excepción planteada para valores no enteros como `randrange(10.5)` o `randrange('10')` se cambió de `ValueError` a `TypeError`. Esto también evita errores en los que `randrange(1e25)` seleccionaría silenciosamente de un rango mayor que `randrange(10**25)`. (Originalmente sugerido por Serhiy Storchaka gh-86388.)
- `argparse.ArgumentParser` cambió la codificación y el controlador de errores para leer argumentos de un archivo (por ejemplo, la opción `fromfile_prefix_chars`) de la codificación de texto predeterminada (por ejemplo, `locale.getpreferredencoding(False)`) a filesystem encoding and error handler. Los archivos de argumentos deben codificarse en UTF-8 en lugar de en la página de códigos ANSI en Windows.
- Remove the `asyncore`-based `smtpd` module deprecated in Python 3.4.7 and 3.5.4. A recommended replacement is the `asyncio`-based aiosmtpd PyPI module.
- `shlex.split()`: Pasar `None` por el argumento *s* ahora genera una excepción, en lugar de leer `sys.stdin`. La característica quedó obsoleta en Python 3.9. (Aportado por Victor Stinner en gh-94352.)
- El módulo `os` ya no acepta rutas tipo bytes, como los tipos `bytearray` y `memoryview`: solo se acepta el tipo exacto `bytes` para cadenas de bytes. (Aportado por Victor Stinner en gh-98393.)

- `syslog.openlog()` y `syslog.closelog()` ahora fallan si se usan en subintérpretes. `syslog.syslog()` aún se puede usar en subintérpretes, pero ahora solo si `syslog.openlog()` ya ha sido llamado en el intérprete principal. Estas nuevas restricciones no se aplican al intérprete principal, por lo que sólo un conjunto muy pequeño de usuarios podría verse afectado. Este cambio ayuda con el aislamiento del intérprete. Además, `syslog` es un contenedor de recursos globales de procesos, que se administran mejor desde el intérprete principal. (Aportado por Donghee Na en gh-99127.)

- Se elimina el comportamiento de bloqueo no documentado de `cached_property()`, porque bloqueaba todas las instancias de la clase, lo que generaba una alta contención de bloqueo. Esto significa que una función de obtención de propiedades almacenadas en caché ahora podría ejecutarse más de una vez para una sola instancia, si dos subprocesos se ejecutan. Para la mayoría de las propiedades almacenadas en caché simples (por ejemplo, aquellas que son idempotentes y simplemente calculan un valor en función de otros atributos de la instancia), esto estará bien. Si se necesita sincronización, implemente el bloqueo dentro de la función de obtención de propiedades en caché o alrededor de puntos de acceso de subprocesos múltiples.

- `sys._current_exceptions()` ahora devuelve una asignación de thread-id a una instancia de excepción, en lugar de a una tupla `(typ, exc, tb)`. (Aportado por Irit Katriel en gh-103176.)

- Al extraer archivos tar usando `tarfile` o `shutil.unpack_archive()`, pase el argumento *filter* para limitar las funciones que pueden resultar sorprendentes o peligrosas. Consulte tarfile-extraction-filter para obtener más detalles.

- La salida de las funciones `tokenize.tokenize()` y `tokenize.generate_tokens()` ahora cambia debido a los cambios introducidos en **PEP 701**. Esto significa que los tokens `STRING` ya no se emiten para cadenas f y los tokens descritos en **PEP 701** ahora se producen en su lugar: `FSTRING_START`, `FSTRING_MIDDLE` y `FSTRING_END` ahora se emiten para partes de «cadena» de cadena f además de los tokens apropiados para la tokenización. en los componentes de la expresión. Por ejemplo, para la cadena f `f"start {1+1} end"`, la versión anterior del tokenizador emitió:

```
1,0-1,18:           STRING         'f"start {1+1} end"'
```

  mientras que la nueva versión emite:

```
1,0-1,2:            FSTRING_START  'f"'
1,2-1,8:            FSTRING_MIDDLE 'start '
1,8-1,9:            OP             '{'
1,9-1,10:           NUMBER         '1'
1,10-1,11:          OP             '+'
1,11-1,12:          NUMBER         '1'
1,12-1,13:          OP             '}'
1,13-1,17:          FSTRING_MIDDLE ' end'
1,17-1,18:          FSTRING_END    '"'
```

  Además, puede haber algunos cambios de comportamiento menores como consecuencia de los cambios necesarios para admitir **PEP 701**. Algunos de estos cambios incluyen:

  - El atributo `type` de los tokens emitidos al tokenizar algunos caracteres de Python no válidos, como `!`, ha cambiado de `ERRORTOKEN` a `OP`.
  - Las cadenas incompletas de una sola línea ahora también generan `tokenize.TokenError` como lo hacen las cadenas incompletas de varias líneas.
  - Algún código Python incompleto o no válido ahora genera `tokenize.TokenError` en lugar de devolver tokens `ERRORTOKEN` arbitrarios al tokenizarlo.
  - Ya no se admite la combinación de tabulaciones y espacios como sangría en el mismo archivo y generará un `TabError`.

- The `threading` module now expects the `_thread` module to have an `_is_main_interpreter` attribute. It is a function with no arguments that returns `True` if the current interpreter is the main interpreter.

Any library or application that provides a custom `_thread` module should provide `_is_main_interpreter()`. (See gh-112826.)

13 Cambios de compilación

- Python ya no usa `setup.py` para crear módulos de extensión C compartidos. Los parámetros de compilación, como encabezados y bibliotecas, se detectan en el script `configure`. Las extensiones están construidas por `Makefile`. La mayoría de las extensiones usan `pkg-config` y recurren a la detección manual. (Aportado por Christian Heimes en gh-93939.)
- Ahora se requiere `va_start()` con dos parámetros, como `va_start(args, format),`, para compilar Python. `va_start()` ya no se llama con un solo parámetro. (Aportado por Kumar Aditya en gh-93207.)
- CPython ahora usa la opción ThinLTO como política predeterminada de optimización del tiempo de enlace si el compilador Clang acepta la marca. (Aportado por Donghee Na en gh-89536.)
- Agregado la variable `COMPILEALL_OPTS` en `Makefile` para anular las opciones de `compileall` (predeterminada: `-j0`) en `make install`. También fusionó los 3 comandos `compileall` en un solo comando para crear archivos .pyc para todos los niveles de optimización (0, 1, 2) a la vez. (Aportado por Victor Stinner en gh-99289.)
- Agregado tripletes de plataforma para LoongArch de 64 bits:
  - loongarch64-linux-gnusf
  - loongarch64-linux-gnuf32
  - loongarch64-linux-gnu

  (Aportado por Zhang Na en gh-90656.)
- `PYTHON_FOR_REGEN` ahora requiere Python 3.10 o posterior.
- Ahora se requieren Autoconf 2.71 y aclocal 1.16.4 para regenerar `!configure`. (Aportado por Christian Heimes en gh-89886.)
- Las compilaciones de Windows y los instaladores de macOS de python.org ahora usan OpenSSL 3.0.

14 Cambios en la API de C

14.1 Nuevas características

- **PEP 697**: presente Unstable C API tier, destinado a herramientas de bajo nivel como depuradores y compiladores JIT. Esta API puede cambiar en cada versión menor de CPython sin advertencias de obsolescencia. Su contenido está marcado con el prefijo `PyUnstable_` en los nombres.

  Constructores de objetos de código:

  - `PyUnstable_Code_New()` (renombrado de `PyCode_New`)
  - `PyUnstable_Code_NewWithPosOnlyArgs()` (renombrado de `PyCode_NewWithPosOnlyArgs`)

  Almacenamiento adicional para objetos de código (**PEP 523**):

  - `PyUnstable_Eval_RequestCodeExtraIndex()` (renombrado de `_PyEval_RequestCodeExtraIndex`)
  - `PyUnstable_Code_GetExtra()` (renombrado de `_PyCode_GetExtra`)

  - `PyUnstable_Code_SetExtra()` (renombrado de `_PyCode_SetExtra`)

  Los nombres originales seguirán estando disponibles hasta que cambie la API respectiva.

  (Aportado por Petr Viktorin en gh-101101.)

- **PEP 697**: agregue una API para extender tipos cuyo diseño de memoria de instancia es opaco:

  - `PyType_Spec.basicsize` puede ser cero o negativo para especificar heredar o ampliar el tamaño de la clase base.
  - Se agregaron `PyObject_GetTypeData()` y `PyType_GetTypeDataSize()` para permitir el acceso a datos de instancia específicos de subclase.
  - Se agregaron `Py_TPFLAGS_ITEMS_AT_END` y `PyObject_GetItemData()` para permitir extender de forma segura ciertos tipos de tamaño variable, incluido `PyType_Type`.
  - Se agregó `Py_RELATIVE_OFFSET` para permitir definir `members` en términos de una estructura específica de subclase.

  (Aportado por Petr Viktorin en gh-103509.)

- Agregado la nueva función limited C API `PyType_FromMetaclass()`, que generaliza el `PyType_FromModuleAndSpec()` existente utilizando un argumento de metaclase adicional. (Aportado por Wenzel Jakob en gh-93012.)

- Se agregó API para crear objetos que se pueden llamar usando the vectorcall protocol a Limited API:

  - `Py_TPFLAGS_HAVE_VECTORCALL`
  - `PyVectorcall_NARGS()`
  - `PyVectorcall_Call()`
  - `vectorcallfunc`

  El indicador `Py_TPFLAGS_HAVE_VECTORCALL` ahora se elimina de una clase cuando se reasigna el método `__call__()` de la clase. Esto hace que vectorcall sea seguro de usar con tipos mutables (es decir, tipos de montón sin el indicador inmutable, `Py_TPFLAGS_IMMUTABLETYPE`). Los tipos mutables que no anulan `tp_call` ahora heredan el indicador `Py_TPFLAGS_HAVE_VECTORCALL`. (Aportado por Petr Viktorin en gh-93274.)

  Se han agregado los indicadores `Py_TPFLAGS_MANAGED_DICT` y `Py_TPFLAGS_MANAGED_WEAKREF`. Esto permite que las clases de extensiones admitan el objeto `__dict__` y las referencias débiles con menos contabilidad, usando menos memoria y con un acceso más rápido.

- Se agregó API para realizar llamadas usando the vectorcall protocol a Limited API:

  - `PyObject_Vectorcall()`
  - `PyObject_VectorcallMethod()`
  - `PY_VECTORCALL_ARGUMENTS_OFFSET`

  Esto significa que tanto el extremo entrante como el saliente del protocolo de llamada vectorial ahora están disponibles en el Limited API. (Aportado por Wenzel Jakob en gh-98586.)

- Agregado dos nuevas funciones públicas, `PyEval_SetProfileAllThreads()` y `PyEval_SetTraceAllThreads()`, que permiten configurar funciones de seguimiento y creación de perfiles en todos los subprocesos en ejecución además del que realiza la llamada. (Aportado por Pablo Galindo en gh-93503.)

- Agregado la nueva función `PyFunction_SetVectorcall()` a la API de C que establece el campo de llamada vectorial de un `PyFunctionObject` determinado. (Aportado por Andrew Frost en gh-92257.)

- La API de C ahora permite registrar devoluciones de llamada a través de `PyDict_AddWatcher()`, `PyDict_Watch()` y API relacionadas para ser llamadas cada vez que se modifica un diccionario. Está pensado para optimizar intérpretes, compiladores JIT o depuradores. (Aportado por Carl Meyer en gh-91052.)
- Agregado las API `PyType_AddWatcher()` y `PyType_Watch()` para registrar devoluciones de llamadas y recibir notificaciones sobre cambios en un tipo. (Aportado por Carl Meyer en gh-91051.)
- Agregado las API `PyCode_AddWatcher()` y `PyCode_ClearWatcher()` para registrar devoluciones de llamadas y recibir notificaciones sobre la creación y destrucción de objetos de código. (Aportado por Itamar Oren en gh-91054.)
- Agregado las funciones `PyFrame_GetVar()` y `PyFrame_GetVarString()` para obtener una variable de marco por su nombre. (Aportado por Victor Stinner en gh-91248.)
- Agregado `PyErr_GetRaisedException()` y `PyErr_SetRaisedException()` para guardar y restaurar la excepción actual. Estas funciones devuelven y aceptan un único objeto de excepción, en lugar de los argumentos triples de los ahora obsoletos `PyErr_Fetch()` y `PyErr_Restore()`. Esto es menos propenso a errores y un poco más eficiente. (Aportado por Mark Shannon en gh-101578.)
- Agregado `_PyErr_ChainExceptions1`, que toma una instancia de excepción, para reemplazar la API heredada `_PyErr_ChainExceptions`, que ahora está en desuso. (Aportado por Mark Shannon en gh-101578.)
- Agregado `PyException_GetArgs()` y `PyException_SetArgs()` como funciones convenientes para recuperar y modificar el `args` pasado al constructor de la excepción. (Aportado por Mark Shannon en gh-101578.)
- Agregado `PyErr_DisplayException()`, que toma una instancia de excepción, para reemplazar la API heredada `PyErr_Display()`. (Aportado por Irit Katriel en gh-102755).
- **PEP 683**: presente *Immortal Objects*, que permite que los objetos omitan los recuentos de referencias y los cambios relacionados en C-API:
  - **`_Py_IMMORTAL_REFCNT`: el recuento de referencias que define un objeto** como inmortal.
  - `_Py_IsImmortal` Comprueba si un objeto tiene el recuento de referencia inmortal.
  - **`PyObject_HEAD_INIT` Esto ahora inicializará el recuento de referencias** `_Py_IMMORTAL_REFCNT` cuando se utiliza con `Py_BUILD_CORE`.
  - **`SSTATE_INTERNED_IMMORTAL` Un identificador para objetos Unicode internos** que son inmortales.
  - **`SSTATE_INTERNED_IMMORTAL_STATIC` Un identificador para Unicode interno** Objetos inmortales y estáticos.
  - **`sys.getunicodeinternedsize` Esto devuelve el número total de Unicode** objetos que han sido internados. Esto ahora es necesario para que `refleak.py` rastree correctamente los recuentos de referencia y los bloques asignados.

  (Contribuido por Eddie Elizondo en gh-84436.)
- **PEP 684**: agregue la nueva función `Py_NewInterpreterFromConfig()` y `PyInterpreterConfig`, que pueden usarse para crear subintérpretes con sus propios GIL. (Consulte *PEP 684: un GIL por intérprete* para obtener más información). (Contribución de Eric Snow en gh-104110.)
- En la versión limitada de C API 3.12, las funciones `Py_INCREF()` y `Py_DECREF()` ahora se implementan como llamadas de función opacas para ocultar los detalles de implementación. (Aportado por Victor Stinner en gh-105387.)

14.2 Portar a Python 3.12

- Se han eliminado las API Unicode heredadas basadas en la representación `Py_UNICODE*`. Migre a API basadas en UTF-8 o `wchar_t*`.
- Las funciones de análisis de argumentos como `PyArg_ParseTuple()` ya no admiten el formato basado en `Py_UNICODE*` (por ejemplo, `u`, `Z`). Migre a otros formatos para Unicode como `s`, `z`, `es` y `U`.
- `tp_weaklist` para todos los tipos integrados estáticos siempre es `NULL`. Este es un campo solo interno en `PyTypeObject`, pero señalamos el cambio en caso de que alguien acceda al campo directamente de todos modos. Para evitar roturas, considere utilizar la C-API pública existente o, si es necesario, la macro `_PyObject_GET_WEAKREFS_LISTPTR()` (solo interna).
- Es posible que este `PyTypeObject.tp_subclasses` solo interno ya no sea un puntero de objeto válido. Su tipo se cambió a `void*` para reflejar esto. Mencionamos esto en caso de que alguien acceda directamente al campo interno.

  Para obtener una lista de subclases, llame al método Python `__subclasses__()` (usando `PyObject_CallMethod()`, por ejemplo).
- Agregado soporte para más opciones de formato (alineación a la izquierda, octales, hexadecimales en mayúsculas, cadenas `intmax_t`, `ptrdiff_t`, `wchar_t` C, ancho variable y precisión) en `PyUnicode_FromFormat()` y `PyUnicode_FromFormatV()`. (Aportado por Serhiy Storchaka en gh-98836.)
- Un carácter de formato no reconocido en `PyUnicode_FromFormat()` y `PyUnicode_FromFormatV()` ahora establece un `SystemError`. En versiones anteriores, provocaba que el resto de la cadena de formato se copiara tal cual en la cadena de resultado y se descartaran los argumentos adicionales. (Aportado por Serhiy Storchaka en gh-95781.)
- Se corrigió la ubicación incorrecta de los letreros en `PyUnicode_FromFormat()` y `PyUnicode_FromFormatV()`. (Aportado por Philip Georgi en gh-95504.)
- Las clases de extensión que deseen agregar un `__dict__` o una ranura de referencia débil deben usar `Py_TPFLAGS_MANAGED_DICT` y `Py_TPFLAGS_MANAGED_WEAKREF` en lugar de `tp_dictoffset` y `tp_weaklistoffset`, respectivamente. El uso de `tp_dictoffset` y `tp_weaklistoffset` aún se admite, pero no es totalmente compatible con la herencia múltiple (gh-95589) y el rendimiento puede ser peor. Las clases que declaran `Py_TPFLAGS_MANAGED_DICT` deben llamar a `_PyObject_VisitManagedDict()` y `_PyObject_ClearManagedDict()` para recorrer y borrar los diccionarios de su instancia. Para borrar referencias débiles, llame a `PyObject_ClearWeakRefs()`, como antes.
- La función `PyUnicode_FSDecoder()` ya no acepta rutas de tipo bytes, como los tipos `bytearray` y `memoryview`: solo se acepta el tipo `bytes` exacto para cadenas de bytes. (Aportado por Victor Stinner en gh-98393.)
- Las macros `Py_CLEAR`, `Py_SETREF` y `Py_XSETREF` ahora solo evalúan sus argumentos una vez. Si un argumento tiene efectos secundarios, estos efectos secundarios ya no se duplican. (Aportado por Victor Stinner en gh-98724.)
- El indicador de error del intérprete ahora siempre está normalizado. Esto significa que `PyErr_SetObject()`, `PyErr_SetString()` y las otras funciones que configuran el indicador de error ahora normalizan la excepción antes de almacenarla. (Aportado por Mark Shannon en gh-101578.)
- `_Py_RefTotal` ya no tiene autoridad y solo se conserva por compatibilidad con ABI. Tenga en cuenta que es un global interno y solo está disponible en compilaciones de depuración. Si lo está utilizando, deberá comenzar a utilizar `_Py_GetGlobalRefTotal()`.
- Las siguientes funciones ahora seleccionan una metaclase apropiada para el tipo recién creado:
  - `PyType_FromSpec()`
  - `PyType_FromSpecWithBases()`

  - `PyType_FromModuleAndSpec()`

  La creación de clases cuya metaclase anule `tp_new` está en desuso y en Python 3.14+ no estará permitida. Tenga en cuenta que estas funciones ignoran `tp_new` de la metaclase, lo que posiblemente permita una inicialización incompleta.

  Tenga en cuenta que `PyType_FromMetaclass()` (agregado en Python 3.12) ya no permite la creación de clases cuya metaclase anule `tp_new` (`__new__()` en Python).

  Dado que `tp_new` anula casi todo lo que hacen las funciones `PyType_From*`, las dos son incompatibles entre sí. El comportamiento existente (ignorar la metaclase durante varios pasos de la creación de tipos) no es seguro en general, ya que las (meta)clases suponen que se llamó a `tp_new`. No existe una solución general sencilla. Uno de los siguientes puede funcionar para usted:

  - Si controlas la metaclase, evita usar `tp_new` en ella:
    - Si se puede omitir la inicialización, se puede realizar en `tp_init`.
    - Si no es necesario crear una instancia de la metaclase desde Python, configure su `tp_new` en `NULL` usando el indicador `Py_TPFLAGS_DISALLOW_INSTANTIATION`. Esto lo hace aceptable para funciones `PyType_From*`.
  - Evite las funciones `PyType_From*`: si no necesita funciones específicas de C (ranuras o configuración del tamaño de la instancia), cree tipos mediante la metaclase calling.
  - Si *know* puede omitir `tp_new` de forma segura, filtre la advertencia de obsolescencia usando `warnings.catch_warnings()` de Python.

- `PyOS_InputHook` y `PyOS_ReadlineFunctionPointer` ya no se llaman en subinterpreters. Esto se debe a que los clientes generalmente dependen del estado global de todo el proceso (ya que estas devoluciones de llamada no tienen forma de recuperar el estado del módulo de extensión).

  Esto también evita situaciones en las que las extensiones pueden encontrarse ejecutándose en un subintérprete que no admiten (o que aún no se han cargado). Consulte gh-104668 para obtener más información.

- `PyLongObject` ha tenido cambios internos para un mejor rendimiento. Aunque las partes internas de `PyLongObject` son privadas, algunos módulos de extensión las utilizan. Ya no se debe acceder directamente a los campos internos, sino que se deben utilizar las funciones API que comienzan con `PyLong_...`. Se proporcionan dos nuevas funciones API *unstable* para un acceso eficiente al valor de `PyLongObject`s que cabe en una sola palabra de máquina:
  - `PyUnstable_Long_IsCompact()`
  - `PyUnstable_Long_CompactValue()`
- Ahora se requiere que los asignadores personalizados, configurados a través de `PyMem_SetAllocator()`, sean seguros para subprocesos, independientemente del dominio de memoria. Los asignadores que no tienen su propio estado, incluidos los «ganchos», no se ven afectados. Si su asignador personalizado aún no es seguro para subprocesos y necesita orientación, cree una nueva incidencia en GitHub y CC `@ericsnowcurrently`.

14.3 Obsoleto

- De acuerdo con **PEP 699**, el campo `ma_version_tag` en `PyDictObject` está obsoleto para los módulos de extensión. Acceder a este campo generará una advertencia del compilador en el momento de la compilación. Este campo se eliminará en Python 3.14. (Contribución de Ramvikrams y Kumar Aditya en gh-101193. PEP de Ken Jin.)
- Variable de configuración global obsoleta:
  - `Py_DebugFlag`: use `PyConfig.parser_debug`

  - `Py_VerboseFlag`: use `PyConfig.verbose`
  - `Py_QuietFlag`: use `PyConfig.quiet`
  - `Py_InteractiveFlag`: use `PyConfig.interactive`
  - `Py_InspectFlag`: use `PyConfig.inspect`
  - `Py_OptimizeFlag`: use `PyConfig.optimization_level`
  - `Py_NoSiteFlag`: use `PyConfig.site_import`
  - `Py_BytesWarningFlag`: use `PyConfig.bytes_warning`
  - `Py_FrozenFlag`: use `PyConfig.pathconfig_warnings`
  - `Py_IgnoreEnvironmentFlag`: use `PyConfig.use_environment`
  - `Py_DontWriteBytecodeFlag`: use `PyConfig.write_bytecode`
  - `Py_NoUserSiteDirectory`: use `PyConfig.user_site_directory`
  - `Py_UnbufferedStdioFlag`: use `PyConfig.buffered_stdio`
  - `Py_HashRandomizationFlag`: use `PyConfig.use_hash_seed` y `PyConfig.hash_seed`
  - `Py_IsolatedFlag`: use `PyConfig.isolated`
  - `Py_LegacyWindowsFSEncodingFlag`: use `PyPreConfig.legacy_windows_fs_encoding`
  - `Py_LegacyWindowsStdioFlag`: use `PyConfig.legacy_windows_stdio`
  - `Py_FileSystemDefaultEncoding`: use `PyConfig.filesystem_encoding`
  - `Py_HasFileSystemDefaultEncoding`: use `PyConfig.filesystem_encoding`
  - `Py_FileSystemDefaultEncodeErrors`: use `PyConfig.filesystem_errors`
  - `Py_UTF8Mode`: use `PyPreConfig.utf8_mode` (ver `Py_PreInitialize()`)

  La API `Py_InitializeFromConfig()` debe usarse con `PyConfig` en su lugar. (Aportado por Victor Stinner en gh-77782.)

- La creación de `immutable types` con bases mutables está obsoleta y se deshabilitará en Python 3.14. (gh-95388)
- El encabezado `structmember.h` está en desuso, aunque sigue estando disponible y no hay planes para eliminarlo.

  Su contenido ahora está disponible simplemente incluyendo `Python.h`, con un prefijo `Py` agregado si faltara:

  - `PyMemberDef`, `PyMember_GetOne()` y `PyMember_SetOne()`
  - Escriba macros como `Py_T_INT`, `Py_T_DOUBLE`, etc. (anteriormente `T_INT`, `T_DOUBLE`, etc.)
  - Las banderas `Py_READONLY` (anteriormente `READONLY`) y `Py_AUDIT_READ` (anteriormente todas en mayúsculas)

  Varios elementos no están expuestos desde `Python.h`:

  - `T_OBJECT` (use `Py_T_OBJECT_EX`)
  - `T_NONE` (anteriormente indocumentado y bastante peculiar)
  - La macro `WRITE_RESTRICTED` que no hace nada.
  - Las macros `RESTRICTED` y `READ_RESTRICTED`, equivalentes a `Py_AUDIT_READ`.
  - En algunas configuraciones, `<stddef.h>` no se incluye en `Python.h`. Debe incluirse manualmente cuando se utiliza `offsetof()`.

El encabezado obsoleto continúa proporcionando su contenido original con los nombres originales. Su código antiguo puede permanecer sin cambios, a menos que las macros de inclusión y sin espacio de nombres adicionales le molesten mucho.

(Contribuido en gh-47146 por Petr Viktorin, basado en trabajos anteriores de Alexander Belopolsky y Matthias Braun.)

- `PyErr_Fetch()` y `PyErr_Restore()` están en desuso. Utilice `PyErr_GetRaisedException()` y `PyErr_SetRaisedException()` en su lugar. (Aportado por Mark Shannon en gh-101578.)
- `PyErr_Display()` está en desuso. Utilice `PyErr_DisplayException()` en su lugar. (Aportado por Irit Katriel en gh-102755).
- `_PyErr_ChainExceptions` está en desuso. Utilice `_PyErr_ChainExceptions1` en su lugar. (Aportado por Irit Katriel en gh-102192.)
- El uso de `PyType_FromSpec()`, `PyType_FromSpecWithBases()` o `PyType_FromModuleAndSpec()` para crear una clase cuya metaclase anula `tp_new` está en desuso. En su lugar, llame a la metaclase.

Eliminación pendiente en Python 3.14

- El campo `ma_version_tag` en `PyDictObject` para módulos de extensión (**PEP 699**; gh-101193).
- Variables de configuración globales:
  - `Py_DebugFlag`: use `PyConfig.parser_debug`
  - `Py_VerboseFlag`: use `PyConfig.verbose`
  - `Py_QuietFlag`: use `PyConfig.quiet`
  - `Py_InteractiveFlag`: use `PyConfig.interactive`
  - `Py_InspectFlag`: use `PyConfig.inspect`
  - `Py_OptimizeFlag`: use `PyConfig.optimization_level`
  - `Py_NoSiteFlag`: use `PyConfig.site_import`
  - `Py_BytesWarningFlag`: use `PyConfig.bytes_warning`
  - `Py_FrozenFlag`: use `PyConfig.pathconfig_warnings`
  - `Py_IgnoreEnvironmentFlag`: use `PyConfig.use_environment`
  - `Py_DontWriteBytecodeFlag`: use `PyConfig.write_bytecode`
  - `Py_NoUserSiteDirectory`: use `PyConfig.user_site_directory`
  - `Py_UnbufferedStdioFlag`: use `PyConfig.buffered_stdio`
  - `Py_HashRandomizationFlag`: use `PyConfig.use_hash_seed` y `PyConfig.hash_seed`
  - `Py_IsolatedFlag`: use `PyConfig.isolated`
  - `Py_LegacyWindowsFSEncodingFlag`: use `PyPreConfig.legacy_windows_fs_encoding`
  - `Py_LegacyWindowsStdioFlag`: use `PyConfig.legacy_windows_stdio`
  - `Py_FileSystemDefaultEncoding`: use `PyConfig.filesystem_encoding`
  - `Py_HasFileSystemDefaultEncoding`: use `PyConfig.filesystem_encoding`
  - `Py_FileSystemDefaultEncodeErrors`: use `PyConfig.filesystem_errors`
  - `Py_UTF8Mode`: use `PyPreConfig.utf8_mode` (ver `Py_PreInitialize()`)

La API `Py_InitializeFromConfig()` debe usarse con `PyConfig` en su lugar.

- Creando `immutable types` con bases mutables (gh-95388).

Eliminación pendiente en Python 3.15

- `PyImport_ImportModuleNoBlock()`: utilizar `PyImport_ImportModule()`
- Tipo `Py_UNICODE_WIDE`: use `wchar_t`
- Tipo `Py_UNICODE`: use `wchar_t`
- Funciones de inicialización de Python:
  - `PySys_ResetWarnOptions()`: borrar `sys.warnoptions` y `warnings.filters`
  - `Py_GetExecPrefix()`: obtener `sys.exec_prefix`
  - `Py_GetPath()`: obtener `sys.path`
  - `Py_GetPrefix()`: obtener `sys.prefix`
  - `Py_GetProgramFullPath()`: obtener `sys.executable`
  - `Py_GetProgramName()`: obtener `sys.executable`
  - `Py_GetPythonHome()`: obtenga `PyConfig.home` o la variable de entorno `PYTHONHOME`

Eliminación pendiente en versiones futuras

Las siguientes API están obsoletas y se eliminarán, aunque actualmente no hay una fecha programada para su eliminación.

- `Py_TPFLAGS_HAVE_FINALIZE`: innecesario desde Python 3.8
- `PyErr_Fetch()`: utilizar `PyErr_GetRaisedException()`
- `PyErr_NormalizeException()`: utilizar `PyErr_GetRaisedException()`
- `PyErr_Restore()`: utilizar `PyErr_SetRaisedException()`
- `PyModule_GetFilename()`: utilizar `PyModule_GetFilenameObject()`
- `PyOS_AfterFork()`: utilizar `PyOS_AfterFork_Child()`
- `PySlice_GetIndicesEx()`: utilice `PySlice_Unpack()` y `PySlice_AdjustIndices()`
- `PyUnicode_AsDecodedObject()`: utilizar `PyCodec_Decode()`
- `PyUnicode_AsDecodedUnicode()`: utilizar `PyCodec_Decode()`
- `PyUnicode_AsEncodedObject()`: utilizar `PyCodec_Encode()`
- `PyUnicode_AsEncodedUnicode()`: utilizar `PyCodec_Encode()`
- `PyUnicode_READY()`: innecesario desde Python 3.12
- `PyErr_Display()`: utilizar `PyErr_DisplayException()`
- `_PyErr_ChainExceptions()`: utilizar `_PyErr_ChainExceptions1`
- Miembro `PyBytesObject.ob_shash`: llame a `PyObject_Hash()` en su lugar
- Miembro `PyDictObject.ma_version_tag`
- API de almacenamiento local de subprocesos (TLS):
  - `PyThread_create_key()`: utilizar `PyThread_tss_alloc()`

- `PyThread_delete_key()`: utilizar `PyThread_tss_free()`
- `PyThread_set_key_value()`: utilizar `PyThread_tss_set()`
- `PyThread_get_key_value()`: utilizar `PyThread_tss_get()`
- `PyThread_delete_key_value()`: utilizar `PyThread_tss_delete()`
- `PyThread_ReInitTLS()`: innecesario desde Python 3.7

14.4 Eliminado

- Elimine el archivo de encabezado `token.h`. Nunca hubo ninguna API C de tokenizador público. El archivo de encabezado `token.h` solo fue diseñado para ser utilizado por componentes internos de Python. (Aportado por Victor Stinner en gh-92651.)
- Se han eliminado las API Unicode heredadas. Consulte **PEP 623** para obtener más detalles.
  - `PyUnicode_WCHAR_KIND`
  - `PyUnicode_AS_UNICODE()`
  - `PyUnicode_AsUnicode()`
  - `PyUnicode_AsUnicodeAndSize()`
  - `PyUnicode_AS_DATA()`
  - `PyUnicode_FromUnicode()`
  - `PyUnicode_GET_SIZE()`
  - `PyUnicode_GetSize()`
  - `PyUnicode_GET_DATA_SIZE()`
- Elimine la macro de función `PyUnicode_InternImmortal()`. (Aportado por Victor Stinner en gh-85858.)

15 Notable changes in 3.12.4

15.1 ipaddress

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

Índice

P

V