What's New in Python

Versión 3.10.5

A. M. Kuchling

agosto 01, 2022

Python Software Foundation
Email: docs@python.org

Índice general

Liberación 3.10.5

Fecha agosto 01, 2022

Editor Pablo Galindo Salgado

This article explains the new features in Python 3.10, compared to 3.9. Python 3.10 was released on October 4, 2021. For full details, see the changelog.

1 Resumen: aspectos destacados de la versión

Nuevas funciones de sintaxis:

- **PEP 634**, Coincidencia de patrones estructurales: Especificación
- **PEP 635**, Coincidencia de patrones estructurales: motivación y fundamento
- **PEP 636**, Coincidencia de patrones estructurales: Tutorial
- bpo-12782, los administradores de contexto entre paréntesis ahora están oficialmente permitidos.

Nuevas funciones en la biblioteca estándar:

- **PEP 618**, agregue verificación de longitud opcional al cierre.

Mejoras en el intérprete:

- **PEP 626**, números de línea precisos para depuración y otras herramientas.

Nuevas funciones de escritura:

- **PEP 604**, Permitir escribir tipos de unión como X | Y
- **PEP 613**, alias de tipo explícito
- **PEP 612**, variables de especificación de parámetros

Desactivaciones, eliminaciones o restricciones importantes:

- **PEP 644**, requiere OpenSSL 1.1.1 o más reciente
- **PEP 632**, módulo distutils obsoleto.
- **PEP 623**, desaprobar y prepararse para la eliminación del miembro wstr en PyUnicodeObject.
- **PEP 624**, eliminar las API del codificador Py_UNICODE
- **PEP 597**, agregar codificación opcional

2 Nuevas características

2.1 Administradores de contexto entre paréntesis

Ahora se admite el uso de entre paréntesis para continuar en varias líneas en los administradores de contexto. Esto permite formatear una colección larga de administradores de contexto en múltiples líneas de una manera similar a como era posible anteriormente con declaraciones de importación. Por ejemplo, todos estos ejemplos ahora son válidos:

```
with (CtxManager() as example):
    ...

with (
    CtxManager1(),
    CtxManager2()
):
    ...

with (CtxManager1() as example,
      CtxManager2()):
    ...

with (CtxManager1(),
      CtxManager2() as example):
    ...

with (
    CtxManager1() as example1,
    CtxManager2() as example2
):
    ...
```

también es posible usar una coma al final del grupo adjunto:

```
with (
    CtxManager1() as example1,
    CtxManager2() as example2,
    CtxManager3() as example3,
):
    ...
```

Esta nueva sintaxis utiliza las capacidades no LL (1) del nuevo analizador. Consulte **PEP 617** para obtener más detalles.

(Contribuido por Guido van Rossum, Pablo Galindo y Lysandros Nikolaou en bpo-12782 y bpo-40334.)

2.2 Mejores mensajes de error

SyntaxErrors

Al analizar el código que contiene paréntesis o corchetes sin cerrar, el intérprete ahora incluye la ubicación del corchete de paréntesis sin cerrar en lugar de mostrar *SyntaxError: unexpected EOF while parsing* o señalar una ubicación incorrecta. Por ejemplo, considere el siguiente código (observe el "{" sin cerrar):

```
expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
            38: 4, 39: 4, 45: 5, 46: 5, 47: 5, 48: 5, 49: 5, 54: 6,
some_other_code = foo()
```

Las versiones anteriores del intérprete informaron lugares confusos como la ubicación del error de sintaxis:

```
File "example.py", line 3
    some_other_code = foo()
                    ^
SyntaxError: invalid syntax
```

pero en Python 3.10 se emite un error más informativo:

```
File "example.py", line 1
    expected = {9: 1, 18: 2, 19: 2, 27: 3, 28: 3, 29: 3, 36: 4, 37: 4,
               ^
SyntaxError: '{' was never closed
```

De manera similar, los errores que involucran cadenas literales no cerradas (entre comillas simples y triples) ahora apuntan al inicio de la cadena en lugar de informar EOF / EOL.

Estas mejoras están inspiradas en trabajos anteriores en el intérprete de PyPy.

(Contribuido por Pablo Galindo en bpo-42864 y Batuhan Taskaya en bpo-40176.)

Las excepciones de `SyntaxError` planteadas por el intérprete ahora resaltarán el rango de error completo de la expresión que constituye el error de sintaxis en sí, en lugar de solo dónde se detecta el problema. De esta manera, en lugar de mostrar (antes de Python 3.10):

```
>>> foo(x, z for z in range(10), t, w)
  File "<stdin>", line 1
    foo(x, z for z in range(10), t, w)
           ^
SyntaxError: Generator expression must be parenthesized
```

ahora Python 3.10 mostrará la excepción como:

```
>>> foo(x, z for z in range(10), t, w)
  File "<stdin>", line 1
    foo(x, z for z in range(10), t, w)
           ^^^^^^^^^^^^^^^^^^^^
SyntaxError: Generator expression must be parenthesized
```

Esta mejora fue aportada por Pablo Galindo en bpo-43914.

Se ha incorporado una cantidad considerable de nuevos mensajes especializados para excepciones `SyntaxError`. Algunos de los más notables son los siguientes:

- Falta `:` antes de los bloques:

  ```
  >>> if rocket.position > event_horizon
    File "<stdin>", line 1
      if rocket.position > event_horizon
                                        ^
  SyntaxError: expected ':'
  ```

  (Contributed by Pablo Galindo in bpo-42997.)

- Tuplas sin paréntesis en objetivos de comprensión:

  ```
  >>> {x,y for x,y in zip('abcd', '1234')}
    File "<stdin>", line 1
      {x,y for x,y in zip('abcd', '1234')}
  ```

(continué en la próxima página)

(proviene de la página anterior)

```
    ^
SyntaxError: did you forget parentheses around the comprehension target?
```

(Contributed by Pablo Galindo in bpo-43017.)

- Faltan comas en literales de colección y entre expresiones:

```
>>> items = {
... x: 1,
... y: 2
... z: 3,
  File "<stdin>", line 3
    y: 2
       ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

(Contributed by Pablo Galindo in bpo-43822.)

- Varios tipos de excepciones sin paréntesis:

```
>>> try:
...     build_dyson_sphere()
... except NotEnoughScienceError, NotEnoughResourcesError:
  File "<stdin>", line 3
    except NotEnoughScienceError, NotEnoughResourcesError:
           ^
SyntaxError: multiple exception types must be parenthesized
```

(Contributed by Pablo Galindo in bpo-43149.)

- Falta `:` y valores en literales de diccionario:

```
>>> values = {
... x: 1,
... y: 2,
... z:
... }
  File "<stdin>", line 4
    z:
     ^
SyntaxError: expression expected after dictionary key and ':'

>>> values = {x:1, y:2, z w:3}
  File "<stdin>", line 1
    values = {x:1, y:2, z w:3}
                        ^
SyntaxError: ':' expected after dictionary key
```

(Contributed by Pablo Galindo in bpo-43823.)

- Bloques `try` sin bloques `except` o `finally`:

```
>>> try:
...     x = 2
... something = 3
  File "<stdin>", line 3
    something  = 3
```

(continué en la próxima página)

(proviene de la página anterior)

```
    ^^^^^^^^^
SyntaxError: expected 'except' or 'finally' block
```

(Contributed by Pablo Galindo in bpo-44305.)

- Uso de = en lugar de == en comparaciones:

```
>>> if rocket.position = event_horizon:
  File "<stdin>", line 1
    if rocket.position = event_horizon:
                       ^
SyntaxError: cannot assign to attribute here. Maybe you meant '==' instead of '='?
```

(Contributed by Pablo Galindo in bpo-43797.)

- Uso de * en f-strings:

```
>>> f"Black holes {*all_black_holes} and revelations"
  File "<stdin>", line 1
    (*all_black_holes)
     ^
SyntaxError: f-string: cannot use starred expression here
```

(Contributed by Pablo Galindo in bpo-41064.)

Errores de sangría

Muchas excepciones de `IndentationError` ahora tienen más contexto con respecto a qué tipo de bloque esperaba una sangría, incluida la ubicación de la declaración:

```
>>> def foo():
...     if lel:
...     x = 2
  File "<stdin>", line 3
    x = 2
    ^
IndentationError: expected an indented block after 'if' statement in line 2
```

AttributeErrors

Al imprimir `AttributeError`, `PyErr_Display()` ofrecerá sugerencias de nombres de atributos similares en el objeto desde el que se generó la excepción:

```
>>> collections.namedtoplo
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: module 'collections' has no attribute 'namedtoplo'. Did you mean: namedtuple?
```

(Contribuido por Pablo Galindo en bpo-38530.)

Advertencia: Tenga en cuenta que esto no funcionará si no se llama a `PyErr_Display()` para mostrar el error, lo que puede suceder si se utiliza alguna otra función de visualización de error personalizada. Este es un escenario común en algunos REPL como IPython.

NameErrors

Al imprimir `NameError` generado por el intérprete, `PyErr_Display()` ofrecerá sugerencias de nombres de variable similares en la función desde la que se generó la excepción:

```
>>> schwarzschild_black_hole = None
>>> schwarschild_black_hole
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'schwarschild_black_hole' is not defined. Did you mean: schwarzschild_
↪black_hole?
```

(Contribuido por Pablo Galindo en bpo-38530.)

Advertencia: Tenga en cuenta que esto no funcionará si no se llama a `PyErr_Display()` para mostrar el error, lo que puede suceder si se utiliza alguna otra función de visualización de error personalizada. Este es un escenario común en algunos REPL como IPython.

2.3 PEP 626: números de línea precisos para depuración y otras herramientas

PEP 626 ofrece números de línea más precisos y confiables para herramientas de depuración, creación de perfiles y cobertura. Los eventos de rastreo, con el número de línea correcto, se generan para todas las líneas de código ejecutadas y solo para las líneas de código que se ejecutan.

El atributo `f_lineno` de los objetos marco siempre contendrá el número de línea esperado.

El atributo `co_lnotab` de los objetos de código está obsoleto y se eliminará en 3.12. El código que necesita convertir de desplazamiento a número de línea debe usar el nuevo método `co_lines()` en su lugar.

2.4 PEP 634: Coincidencia de patrones estructurales

Se ha agregado la coincidencia de patrones estructurales en forma de un *match statement* and *case statements* de patrones con acciones asociadas. Los patrones constan de secuencias, asignaciones, tipos de datos primitivos e instancias de clases. La coincidencia de patrones permite a los programas extraer información de tipos de datos complejos, ramificar la estructura de los datos y aplicar acciones específicas basadas en diferentes formas de datos.

Sintaxis y operaciones

La sintaxis genérica de la coincidencia de patrones es:

```
match subject:
    case <pattern_1>:
        <action_1>
    case <pattern_2>:
        <action_2>
    case <pattern_3>:
        <action_3>
    case _:
        <action_wildcard>
```

Una declaración de coincidencia toma una expresión y compara su valor con los patrones sucesivos dados como uno o más bloques de casos. Específicamente, la coincidencia de patrones funciona mediante:

1. usando datos con tipo y forma (el `subject`)
2. evaluar el `subject` en la declaración `match`
3. comparar el sujeto con cada patrón en una declaración `case` de arriba a abajo hasta que se confirme una coincidencia.
4. ejecutar la acción asociada con el patrón de la coincidencia confirmada
5. Si no se confirma una coincidencia exacta, el último caso, un comodín _, si se proporciona, se utilizará como caso coincidente. Si no se confirma una coincidencia exacta y no existe un comodín, todo el bloque de coincidencias es inactivo.

Enfoque declarativo

Los lectores pueden ser conscientes de la coincidencia de patrones a través del simple ejemplo de hacer coincidir un sujeto (objeto de datos) con un literal (patrón) con la declaración de cambio que se encuentra en C, Java o JavaScript (y muchos otros lenguajes). A menudo, la declaración de cambio se utiliza para comparar un objeto / expresión con declaraciones de casos que contienen literales.

Se pueden encontrar ejemplos más poderosos de coincidencia de patrones en lenguajes como Scala y Elixir. Con la coincidencia de patrones estructurales, el enfoque es «declarativo» y establece explícitamente las condiciones (los patrones) para que los datos coincidan.

Si bien una serie «imperativa» de instrucciones que utilizan declaraciones «if» anidadas podría usarse para lograr algo similar a la coincidencia de patrones estructurales, es menos claro que el enfoque «declarativo». En cambio, el enfoque «declarativo» establece las condiciones que se deben cumplir para una coincidencia y es más legible a través de sus patrones explícitos. Si bien la coincidencia de patrones estructurales se puede usar en su forma más simple comparando una variable con un literal en una declaración de caso, su verdadero valor para Python radica en su manejo del tipo y la forma del sujeto.

Patrón simple: coincidir con un literal

Veamos este ejemplo como coincidencia de patrones en su forma más simple: un valor, el sujeto, se empareja con varios literales, los patrones. En el siguiente ejemplo, `status` es el tema de la declaración de coincidencia. Los patrones son cada una de las declaraciones de casos, donde los literales representan códigos de estado de solicitud. La acción asociada al caso se ejecuta después de una coincidencia:

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
        case _:
            return "Something's wrong with the internet"
```

Si a la función anterior se le pasa un `status` de 418, se devuelve «Soy una tetera». Si a la función anterior se le pasa un `status` de 500, la declaración de caso con _ coincidirá como un comodín y se devuelve «Algo anda mal con Internet». Tenga en cuenta el último bloque: el nombre de la variable, _, actúa como *wildcard* y asegura que el sujeto siempre coincidirá. El uso de _ es opcional.

Puede combinar varios literales en un solo patrón usando | («o»)

```
case 401 | 403 | 404:
    return "Not allowed"
```

Comportamiento sin el comodín

Si modificamos el ejemplo anterior eliminando el último bloque de caso, el ejemplo se convierte en:

```
def http_error(status):
    match status:
        case 400:
            return "Bad request"
        case 404:
            return "Not found"
        case 418:
            return "I'm a teapot"
```

Sin el uso de _ en una declaración de caso, es posible que no exista una coincidencia. Si no existe ninguna coincidencia, el comportamiento es inactivo. Por ejemplo, si se pasa `status` de 500, se produce una no operación.

Patrones con un literal y una variable

Los patrones pueden verse como asignaciones de desempaquetado y se puede usar un patrón para vincular variables. En este ejemplo, un punto de datos se puede descomprimir en su coordenada xy coordenada y:

```
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
```

(continué en la próxima página)

(proviene de la página anterior)

```
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

El primer patrón tiene dos literales, `(0, 0)`, y se puede considerar como una extensión del patrón literal que se muestra arriba. Los siguientes dos patrones combinan un literal y una variable, y la variable *binds* un valor del sujeto (`point`). El cuarto patrón captura dos valores, lo que lo hace conceptualmente similar a la asignación de desembalaje `(x, y) = point`.

Patrones y clases

Si está usando clases para estructurar sus datos, puede usar como patrón el nombre de la clase seguido de una lista de argumentos que se asemeja a un constructor. Este patrón tiene la capacidad de capturar atributos de clase en variables:

```
class Point:
    x: int
    y: int

def location(point):
    match point:
        case Point(x=0, y=0):
            print("Origin is the point's location.")
        case Point(x=0, y=y):
            print(f"Y={y} and the point is on the y-axis.")
        case Point(x=x, y=0):
            print(f"X={x} and the point is on the x-axis.")
        case Point():
            print("The point is located somewhere else on the plane.")
        case _:
            print("Not a point")
```

Patrones con parámetros posicionales

Puede usar parámetros posicionales con algunas clases integradas que proporcionan un orden para sus atributos (por ejemplo, clases de datos). También puede definir una posición específica para atributos en patrones configurando el atributo especial `__match_args__` en sus clases. Si se establece en («x», «y»), los siguientes patrones son todos equivalentes (y todos vinculan el atributo `y` a la variable `var`):

```
Point(1, var)
Point(1, y=var)
Point(x=1, y=var)
Point(y=var, x=1)
```

Patrones anidados

Los patrones se pueden anidar arbitrariamente. Por ejemplo, si nuestros datos son una lista corta de puntos, podrían coincidir así:

```
match points:
    case []:
        print("No points in the list.")
    case [Point(0, 0)]:
        print("The origin is the only point in the list.")
    case [Point(x, y)]:
        print(f"A single point {x}, {y} is in the list.")
    case [Point(0, y1), Point(0, y2)]:
        print(f"Two points on the Y axis at {y1}, {y2} are in the list.")
    case _:
        print("Something else is found in the list.")
```

Patrones complejos y el comodín

Hasta este punto, los ejemplos han utilizado _ solo en la última declaración de caso. Se puede utilizar un comodín en patrones más complejos, como `('error', code, _)`. Por ejemplo:

```
match test_variable:
    case ('warning', code, 40):
        print("A warning has been received.")
    case ('error', code, _):
        print(f"An error {code} occurred.")
```

En el caso anterior, `test_variable` coincidirá con (“error”, código, 100) y (“error”, código, 800).

Guardia

Podemos agregar una cláusula `if` a un patrón, conocido como «guardia». Si la guardia es falsa, `match` pasa a probar el siguiente bloque de caso. Tenga en cuenta que la captura de valor ocurre antes de que se evalúe la guardia:

```
match point:
    case Point(x, y) if x == y:
        print(f"The point is located on the diagonal Y=X at {x}.")
    case Point(x, y):
        print(f"Point is not on the diagonal.")
```

Otras características clave

Varias otras características clave:

- Al igual que las asignaciones de desempaquetado, los patrones de tupla y lista tienen exactamente el mismo significado y en realidad coinciden con secuencias arbitrarias. Técnicamente, el tema debe ser una secuencia. Por lo tanto, una excepción importante es que los patrones no coinciden con los iteradores. Además, para evitar un error común, los patrones de secuencia no coinciden con las cadenas.
- Los patrones de secuencia admiten comodines: `[x, y, *rest]` y `(x, y, *rest)` funcionan de manera similar a los comodines en las asignaciones de desempaquetado. El nombre después de `*` también puede ser _, por lo que `(x, y, *_)` coincide con una secuencia de al menos dos elementos sin vincular los elementos restantes.

- Patrones de mapeo: `{"bandwidth": b, "latency": l}` captura los valores `"bandwidth"` y `"latency"` de un dict. A diferencia de los patrones de secuencia, las claves adicionales se ignoran. También se admite un comodín `**rest`. (Pero `**_` sería redundante, por lo que no está permitido).
- Los subpatrones se pueden capturar utilizando la palabra clave `as`:

```
case (Point(x1, y1), Point(x2, y2) as p2): ...
```

  Esto vincula x1, y1, x2, y2 como cabría esperar sin la cláusula `as` y p2 a todo el segundo elemento del tema.

- La mayoría de los literales se comparan por igualdad. Sin embargo, los singleton `True`, `False` y `None` se comparan por identidad.
- Las constantes con nombre se pueden usar en patrones. Estas constantes nombradas deben ser nombres con puntos para evitar que la constante se interprete como una variable de captura:

```
from enum import Enum
class Color(Enum):
    RED = 0
    GREEN = 1
    BLUE = 2

match color:
    case Color.RED:
        print("I see red!")
    case Color.GREEN:
        print("Grass is green")
    case Color.BLUE:
        print("I'm feeling the blues :(")
```

Para obtener la especificación completa, consulte **PEP 634**. La motivación y el fundamento están en **PEP 635**, y un tutorial más largo está en **PEP 636**.

2.5 Opción opcional `EncodingWarning` y `encoding="locale"`

La codificación predeterminada de `TextIOWrapper` y `open()` depende de la plataforma y la configuración regional. Dado que UTF-8 se usa en la mayoría de las plataformas Unix, omitir la opción `encoding` al abrir archivos UTF-8 (por ejemplo, JSON, YAML, TOML, Markdown) es un error muy común. Por ejemplo:

```
# BUG: "rb" mode or encoding="utf-8" should be used.
with open("data.json") as f:
    data = json.load(f)
```

Para encontrar este tipo de error, se agrega un `EncodingWarning` opcional. Se emite cuando `sys.flags.warn_default_encoding` es verdadero y se utiliza la codificación predeterminada específica de la configuración regional.

Se agregan la opción `-X warn_default_encoding` y `PYTHONWARNDEFAULTENCODING` para habilitar la advertencia.

Consulte io-text-encoding para obtener más información.

3 Nuevas funciones relacionadas con las sugerencias de tipos

Esta sección cubre los cambios importantes que afectan a las sugerencias de tipo **PEP 484** y al módulo `typing`.

3.1 PEP 604: Nuevo tipo de operador unión

Se introdujo un nuevo operador de unión de tipos que habilita la sintaxis `X | Y`. Esto proporciona una forma más limpia de expresar "tipo X o tipo Y" en lugar de usar `typing.Union`, especialmente en sugerencias de tipo.

En versiones anteriores de Python, para aplicar una sugerencia de tipo para funciones que aceptan argumentos de múltiples tipos, se usó `typing.Union`:

```
def square(number: Union[int, float]) -> Union[int, float]:
    return number ** 2
```

Las sugerencias de tipo ahora se pueden escribir de una manera más sucinta:

```
def square(number: int | float) -> int | float:
    return number ** 2
```

Esta nueva sintaxis también se acepta como segundo argumento para `isinstance()` y `issubclass()`:

```
>>> isinstance(1, int | str)
True
```

Consulte types-union y **PEP 604** para obtener más detalles.

(Contribuido por Maggie Moss y Philippe Prados en bpo-41428, con adiciones de Yurii Karabas y Serhiy Storchaka en bpo-44490.)

3.2 PEP 612: Variables de especificación de parámetros

Se han agregado al módulo `typing` dos nuevas opciones para mejorar la información proporcionada a los verificadores de tipo estático para **PEP 484** "s `Callable`.

La primera es la variable de especificación de parámetros. Se utilizan para reenviar los tipos de parámetros de un invocable a otro invocable, un patrón que se encuentra comúnmente en funciones y decoradores de orden superior. Se pueden encontrar ejemplos de uso en `typing.ParamSpec`. Anteriormente, no había una manera fácil de escribir anotar la dependencia de los tipos de parámetros de una manera tan precisa.

La segunda opción es el nuevo operador `Concatenate`. Se usa junto con las variables de especificación de parámetros para escribir anotar un invocable de orden superior que agrega o elimina parámetros de otro invocable. Se pueden encontrar ejemplos de uso en `typing.Concatenate`.

Consulte `typing.Callable`, `typing.ParamSpec`, `typing.Concatenate`, `typing.ParamSpecArgs`, `typing.ParamSpecKwargs` y **PEP 612** para obtener más detalles.

(Contribuido por Ken Jin en bpo-41559, con pequeñas mejoras de Jelle Zijlstra en bpo-43783. PEP escrito por Mark Mendoza.)

3.3 PEP 613: TypeAlias

PEP 484 introdujo el concepto de alias de tipo, solo requiriendo que sean asignaciones no anotadas de nivel superior. Esta simplicidad a veces dificultaba que los verificadores de tipos distinguieran entre alias de tipos y asignaciones ordinarias, especialmente cuando se trataba de referencias directas o tipos no válidos. Comparar:

```
StrCache = 'Cache[str]'  # a type alias
LOG_PREFIX = 'LOG[DEBUG]'  # a module constant
```

Ahora el módulo `typing` tiene un valor especial `TypeAlias` que le permite declarar los alias de tipo de forma más explícita:

```
StrCache: TypeAlias = 'Cache[str]'  # a type alias
LOG_PREFIX = 'LOG[DEBUG]'  # a module constant
```

Consulte **PEP 613** para obtener más detalles.

(Contribuido por Mikhail Golubev en bpo-41923.)

3.4 PEP 647: protectores de tipo definidos por el usuario

Se ha agregado `TypeGuard` al módulo `typing` para anotar las funciones de protección de tipos y mejorar la información proporcionada a los verificadores de tipos estáticos durante el estrechamiento de tipos. Para obtener más información, consulte la documentación de `TypeGuard` y **PEP 647**.

(Contribuido por Ken Jin y Guido van Rossum en bpo-43766. PEP escrito por Eric Traut.)

4 Otros cambios de idioma

- El tipo `int` tiene un nuevo método `int.bit_count()`, que devuelve el número de unos en la expansión binaria de un entero dado, también conocido como recuento de población. (Contribuido por Niklas Fiekas en bpo-29882.)
- Las vistas devueltas por `dict.keys()`, `dict.values()` y `dict.items()` ahora tienen todas un atributo `mapping` que proporciona un objeto `types.MappingProxyType` que envuelve el diccionario original. (Contribuido por Dennis Sweeney en bpo-40890.)
- **PEP 618**: La función `zip()` ahora tiene un indicador `strict` opcional, que se utiliza para requerir que todos los iterables tengan la misma longitud.
- Las funciones integradas y de extensión que toman argumentos enteros ya no aceptan `Decimal` s, `Fraction` sy otros objetos que se pueden convertir a números enteros solo con una pérdida (por ejemplo, que tienen el método `__int__()` pero no tienen el método `__index__()`). (Contribuido por Serhiy Storchaka en bpo-37999.)
- Si `object.__ipow__()` devuelve `NotImplemented`, el operador retrocederá correctamente a `object.__pow__()` y `object.__rpow__()` como se esperaba. (Contribuido por Alex Shkop en bpo-38302.)
- Las expresiones de asignación ahora se pueden usar sin paréntesis dentro de literales de conjuntos y comprensiones de conjuntos, así como en índices de secuencia (pero no por sectores).
- Las funciones tienen un nuevo atributo `__builtins__` que se usa para buscar símbolos incorporados cuando se ejecuta una función, en lugar de buscar en `__globals__['__builtins__']`. El atributo se inicializa desde `__globals__["__builtins__"]` si existe, de lo contrario desde las incorporaciones actuales. (Contribuido por Mark Shannon en bpo-42990.)
- Se han agregado dos nuevas funciones integradas: `aiter()` y `anext()` para proporcionar contrapartes asíncronas a `iter()` y `next()`, respectivamente. (Contribuido por Joshua Bronson, Daniel Pope y Justin Wang en bpo-31861.)

- Los métodos estáticos (`@staticmethod`) y los métodos de clase (`@classmethod`) ahora heredan los atributos del método (`__module__`, `__name__`, `__qualname__`, `__doc__`, `__annotations__`) y tienen un nuevo atributo `__wrapped__`. Además, los métodos estáticos ahora se pueden llamar como funciones regulares. (Contribuido por Victor Stinner en bpo-43682.)
- Las anotaciones para objetivos complejos (todo junto a los objetivos `simple name` definidos por **PEP 526**) ya no causan ningún efecto de tiempo de ejecución con `from __future__ import annotations`. (Contribuido por Batuhan Taskaya en bpo-42737.)
- Los objetos de clase y módulo ahora crean de forma perezosa anotaciones vacías dictados a pedido. Los dictados de anotaciones se almacenan en el `__dict__` del objeto para compatibilidad con versiones anteriores. Esto mejora las mejores prácticas para trabajar con `__annotations__`; para obtener más información, consulte annotations-howto. (Contribuido por Larry Hastings en bpo-43901.)
- Las anotaciones consisten en `yield`, `yield from`, `await` o expresiones con nombre ahora están prohibidas bajo `from __future__ import annotations` debido a sus efectos secundarios. (Contribuido por Batuhan Taskaya en bpo-42725.)
- El uso de variables independientes, `super()` y otras expresiones que podrían alterar el procesamiento de la tabla de símbolos como anotaciones ahora no tienen efecto bajo `from __future__ import annotations`. (Contribuido por Batuhan Taskaya en bpo-42725.)
- Los valores hash de NaN tanto del tipo `float` como del tipo `decimal.Decimal` ahora dependen de la identidad del objeto. Anteriormente, siempre tenían hash en `0` aunque los valores de NaN no son iguales entre sí. Esto provocó un comportamiento de tiempo de ejecución potencialmente cuadrático debido a colisiones de hash excesivas al crear diccionarios y conjuntos que contienen varios NaN. (Contribuido por Raymond Hettinger en bpo-43475.)
- Un `SyntaxError` (en lugar de una constante `NameError`) se lanzara cuando se elimine la constante `__debug__`. (Contribuido por Dong-hee Na en bpo-45000).
- Las excepciones `SyntaxError` ahora tienen atributos `end_lineno` y `end_offset`. Serán `None` si no se determinan. (Contribuido por Pablo Galindo en bpo-43914.)

5 Nuevos módulos

- Ninguno todavía.

6 Módulos mejorados

6.1 asyncio

Agregue el método `connect_accepted_socket()` faltante. (Contribuido por Alex Grönholm en bpo-41332.)

6.2 argumentar

La frase engañosa «argumentos opcionales» fue reemplazada por «opciones» en la ayuda de argparse. Algunas pruebas pueden requerir una adaptación si se basan en la coincidencia exacta de la salida. (Contribuido por Raymond Hettinger en bpo-9694.)

6.3 formación

El método `index()` de `array.array` ahora tiene parámetros *start* and *stop* opcionales. (Contribuido por Anders Lorentsen y Zackery Spytz en bpo-31956.)

6.4 asynchat, asyncore, smtpd

Estos módulos se han marcado como obsoletos en la documentación del módulo desde Python 3.6. Ahora se ha agregado un `DeprecationWarning` en tiempo de importación a estos tres módulos.

6.5 base64

Agregue `base64.b32hexencode()` y `base64.b32hexdecode()` para admitir la codificación Base32 con alfabeto hexadecimal extendido.

6.6 bdb

Agregue `clearBreakpoints()` para restablecer todos los puntos de interrupción establecidos. (Contribuido por Irit Katriel en bpo-24160.)

6.7 bisecar

Se agregó la posibilidad de proporcionar una función *key* a las API en el módulo `bisect`. (Contribuido por Raymond Hettinger en bpo-4356.)

6.8 códecs

Agregue una función `codecs.unregister()` para anular el registro de una función de búsqueda de códec. (Contribuido por Hai Shi en bpo-41842.)

6.9 colecciones.abc

El `__args__` del parameterized generic para `collections.abc.Callable` ahora es consistente con `typing.Callable`. `collections.abc.Callable` genérico ahora aplana los parámetros de tipo, similar a lo que hace actualmente `typing.Callable`. Esto significa que `collections.abc.Callable[[int, str], str]` tendrá `__args__` de `(int, str, str)`; anteriormente esto era `([int, str], str)`. Para permitir este cambio, `types.GenericAlias` ahora puede ser subclasificado, y se devolverá una subclase al subíndice el tipo `collections.abc.Callable`. Tenga en cuenta que se puede generar un `TypeError` para formas no válidas de parametrizar `collections.abc.Callable` que pueden haber pasado silenciosamente en Python 3.9. (Contribuido por Ken Jin en bpo-42195.)

6.10 contextlib

Agregue un administrador de contexto `contextlib.aclosing()` para cerrar de forma segura los generadores asíncronos y los objetos que representan recursos liberados de manera asíncrona. (Contribuido por Joongi Kim y John Belmonte en bpo-41229.)

Agregue soporte de administrador de contexto asincrónico a `contextlib.nullcontext()`. (Contribuido por Tom Gringauz en bpo-41543.)

Agregue `AsyncContextDecorator`, para admitir el uso de administradores de contexto asíncronos como decoradores.

6.11 maldiciones

Las funciones de color extendidas agregadas en ncurses 6.1 serán utilizadas de forma transparente por `curses.color_content()`, `curses.init_color()`, `curses.init_pair()` y `curses.pair_content()`. Una nueva función, `curses.has_extended_color_support()`, indica si la biblioteca ncurses subyacente proporciona compatibilidad de color ampliada. (Contribuido por Jeffrey Kintscher y Hans Petter Jansson en bpo-36982.)

Las constantes `BUTTON5_*` ahora se exponen en el módulo `curses` si las proporciona la biblioteca de curses subyacente. (Contribuido por Zackery Spytz en bpo-39273.)

6.12 clases de datos

__slots__

Se agregó el parámetro `slots` en el decorador `dataclasses.dataclass()`. (Contribuido por Yurii Karabas en bpo-42269)

Campos solo de palabras clave

dataclasses ahora admite campos que son solo palabras clave en el método __init__ generado. Hay varias formas de especificar campos de solo palabras clave.

Puede decir que todos los campos son solo palabras clave:

```
from dataclasses import dataclass

@dataclass(kw_only=True)
class Birthday:
    name: str
    birthday: datetime.date
```

Tanto `name` como `birthday` son parámetros de solo palabras clave para el método __init__ generado.

Puede especificar solo palabras clave por campo:

```
from dataclasses import dataclass

@dataclass
class Birthday:
    name: str
    birthday: datetime.date = field(kw_only=True)
```

Aquí solo `birthday` es solo palabra clave. Si configura `kw_only` en campos individuales, tenga en cuenta que existen reglas sobre el reordenamiento de los campos debido a que los campos de solo palabras clave deben seguir campos que no son solo de palabras clave. Consulte la documentación completa de clases de datos para obtener más detalles.

También puede especificar que todos los campos que siguen a un marcador KW_ONLY sean solo de palabras clave. Este será probablemente el uso más común:

```
from dataclasses import dataclass, KW_ONLY

@dataclass
class Point:
    x: float
    y: float
    _: KW_ONLY
    z: float = 0.0
    t: float = 0.0
```

Here, `z` and `t` are keyword-only parameters, while `x` and `y` are not. (Contributed by Eric V. Smith in bpo-43532.)

6.13 distutils

Todo el paquete `distutils` está obsoleto y se eliminará en Python 3.12. Su funcionalidad para especificar compilaciones de paquetes ya ha sido completamente reemplazada por paquetes de terceros `setuptools` y `packaging`, y la mayoría de las otras API de uso común están disponibles en otras partes de la biblioteca estándar (como `platform`, `shutil`, `subprocess` o `sysconfig`). No hay planes para migrar ninguna otra funcionalidad de `distutils`, y las aplicaciones que utilizan otras funciones deben planificar la realización de copias privadas del código. Consulte **PEP 632** para obtener más información.

Se eliminó el comando `bdist_wininst` en desuso en Python 3.8. Ahora se recomienda el comando `bdist_wheel` para distribuir paquetes binarios en Windows. (Contribuido por Victor Stinner en bpo-42802.)

6.14 doctest

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133.)

6.15 codificaciones

`encodings.normalize_encoding()` ahora ignora los caracteres que no son ASCII. (Contribuido por Hai Shi en bpo-39337.)

6.16 entrada de archivo

Agregue los parámetros *encoding* and *errors* en `fileinput.input()` y `fileinput.FileInput`. (Contribuido por Inada Naoki en bpo-43712.)

`fileinput.hook_compressed()` ahora devuelve el objeto `TextIOWrapper` cuando *mode* es «r» y el archivo está comprimido, como archivos sin comprimir. (Contribuido por Inada Naoki en bpo-5758.)

6.17 manipulador de faltas

El módulo `faulthandler` ahora detecta si ocurre un error fatal durante la recolección de un recolector de basura. (Contribuido por Victor Stinner en bpo-44466.)

6.18 GC

Agregue ganchos de auditoría para `gc.get_objects()`, `gc.get_referrers()` y `gc.get_referents()`. (Contribuido por Pablo Galindo en bpo-43439.)

6.19 glob

Agregue los parámetros *root_dir* and *dir_fd* en `glob()` y `iglob()` que permiten especificar el directorio raíz para la búsqueda. (Contribuido por Serhiy Storchaka en bpo-38144.)

6.20 hashlib

El módulo hashlib requiere OpenSSL 1.1.1 o más reciente. (Contribuido por Christian Heimes en **PEP 644** y bpo-43669.)

El módulo hashlib tiene soporte preliminar para OpenSSL 3.0.0. (Contribuido por Christian Heimes en bpo-38820 y otros números).

El respaldo de Python puro de `pbkdf2_hmac()` está en desuso. En el futuro, PBKDF2-HMAC solo estará disponible cuando Python se haya construido con soporte OpenSSL. (Contribuido por Christian Heimes en bpo-43880.)

6.21 hmac

El módulo hmac ahora usa la implementación HMAC de OpenSSL internamente. (Contribuido por Christian Heimes en bpo-40645.)

6.22 IDLE e idlelib

Make IDLE invoke `sys.excepthook()` (when started without "-n"). User hooks were previously ignored. (Contributed by Ken Hilton in bpo-43008.)

Rearrange the settings dialog. Split the General tab into Windows and Shell/Ed tabs. Move help sources, which extend the Help menu, to the Extensions tab. Make space for new options and shorten the dialog. The latter makes the dialog better fit small screens. (Contributed by Terry Jan Reedy in bpo-40468.) Move the indent space setting from the Font tab to the new Windows tab. (Contributed by Mark Roseman and Terry Jan Reedy in bpo-33962.)

The changes above were backported to a 3.9 maintenance release.

Agrega una barra lateral de Shell. Mueva el indicador principal ("»>") a la barra lateral. Agregue mensajes secundarios ("…") a la barra lateral. El clic izquierdo y el arrastre opcional seleccionan una o más líneas de texto, como con la barra lateral del número de línea del editor. Al hacer clic derecho después de seleccionar líneas de texto, se muestra un menú contextual con "copiar con indicaciones". Esto comprime los mensajes de la barra lateral con líneas del texto seleccionado. Esta opción también aparece en el menú contextual del texto. (Contribuido por Tal Einat en bpo-37903.)

Use spaces instead of tabs to indent interactive code. This makes interactive code entries "look right". Making this feasible was a major motivation for adding the shell sidebar. (Contributed by Terry Jan Reedy in bpo-37892.)

Highlight the new soft keywords `match`, `case`, and _ in pattern-matching statements. However, this highlighting is not perfect and will be incorrect in some rare cases, including some _-s in `case` patterns. (Contributed by Tal Einat in bpo-44010.)

New in 3.10 maintenance releases.

Apply syntax highlighting to *.pyi* files. (Contributed by Alex Waygood and Terry Jan Reedy in bpo-45447.)

6.23 importlib.metadata

Paridad de características con `importlib_metadata` 4.6 (history).

importlib.metadata entry points now provide a nicer experience for selecting entry points by group and name through a new `importlib.metadata.EntryPoints` class. See the Compatibility Note in the docs for more info on the deprecation and usage.

Se agregó `importlib.metadata.packages_distributions()` para resolver módulos y paquetes de Python de nivel superior en su `importlib.metadata.Distribution`.

6.24 inspeccionar

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133.)

Agregue `inspect.get_annotations()`, que calcula de manera segura las anotaciones definidas en un objeto. Resuelve las peculiaridades de acceder a las anotaciones en varios tipos de objetos y hace muy pocas suposiciones sobre el objeto que examina. `inspect.get_annotations()` también puede eliminar correctamente las cadenas de anotaciones. `inspect.get_annotations()` ahora se considera la mejor práctica para acceder al dictado de anotaciones definido en cualquier objeto de Python; Para obtener más información sobre las mejores prácticas para trabajar con anotaciones, consulte annotations-howto. De manera relacionada, `inspect.signature()`, `inspect.Signature.from_callable()` y `inspect.Signature.from_function()` ahora llaman a `inspect.get_annotations()` para recuperar anotaciones. Esto significa que `inspect.signature()` y `inspect.Signature.from_callable()` ahora también pueden anular cadenas de anotaciones. (Contribuido por Larry Hastings en bpo-43817.)

6.25 itertools

Add `itertools.pairwise()`. (Contributed by Raymond Hettinger in bpo-38200.)

6.26 caché de línea

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133.)

6.27 os

Agregue soporte `os.cpu_count()` para VxWorks RTOS. (Contribuido por Peixing Xin en bpo-41440.)

Agregue una nueva función `os.eventfd()` y ayudantes relacionados para envolver el syscall `eventfd2` en Linux. (Contribuido por Christian Heimes en bpo-41001.)

Agregue `os.splice()` que permite mover datos entre dos descriptores de archivos sin copiar entre el espacio de direcciones del kernel y el espacio de direcciones del usuario, donde uno de los descriptores de archivos debe hacer referencia a una tubería. (Contribuido por Pablo Galindo en bpo-41625.)

Agregue `O_EVTONLY`, `O_FSYNC`, `O_SYMLINK` y `O_NOFOLLOW_ANY` para macOS. (Contribuido por Dong-hee Na en bpo-43106.)

6.28 os.path

`os.path.realpath()` ahora acepta un argumento de solo palabra clave *strict*. Cuando se establece en `True`, `OSError` se genera si no existe una ruta o se encuentra un bucle de enlace simbólico. (Contribuido por Barney Gale en bpo-43757.)

6.29 Pathlib

Add slice support to `PurePath.parents`. (Contributed by Joshua Cannon in bpo-35498.)

Add negative indexing support to `PurePath.parents`. (Contributed by Yaroslav Pankovych in bpo-21041.)

Agregue el método `Path.hardlink_to` que reemplaza a `link_to()`. El nuevo método tiene el mismo orden de argumentos que `symlink_to()`. (Contribuido por Barney Gale en bpo-39950.)

`pathlib.Path.stat()` y `chmod()` ahora aceptan un argumento de solo palabra clave *follow_symlinks* para mantener la coherencia con las funciones correspondientes en el módulo `os`. (Contribuido por Barney Gale en bpo-39906.)

6.30 plataforma

Add `platform.freedesktop_os_release()` to retrieve operation system identification from freedesktop.org os-release standard file. (Contributed by Christian Heimes in bpo-28468.)

6.31 pprint

`pprint.pprint()` ahora acepta un nuevo argumento de palabra clave `underscore_numbers`. (Contribuido por sblondon en bpo-42914.)

`pprint` ahora puede imprimir de forma bonita instancias de `dataclasses.dataclass`. (Contribuido por Lewis Gaul en bpo-43080.)

6.32 py_compile

Agregue la opción `--quiet` a la interfaz de línea de comandos de `py_compile`. (Contribuido por Gregory Schevchenko en bpo-38731.)

6.33 pyclbr

Agregue un atributo `end_lineno` a los objetos `Function` y `Class` en el árbol devuelto por `pyclbr.readline()` y `pyclbr.readline_ex()`. Coincide con el `lineno` existente (inicio). (Contribuido por Aviral Srivastava en bpo-38307.)

6.34 dejar de lado

El módulo `shelve` ahora usa `pickle.DEFAULT_PROTOCOL` por defecto en lugar del protocolo `pickle 3` al crear estantes. (Contribuido por Zackery Spytz en bpo-34204.)

6.35 Estadísticas

Agregue las funciones `covariance()`, `correlation()` de Pearson y `linear_regression()` simple. (Contribuido por Tymoteusz Wołodźko en bpo-38490.)

6.36 sitio

Cuando un módulo no define `__loader__`, recurre a `__spec__.loader`. (Contribuido por Brett Cannon en bpo-42133.)

6.37 enchufe

La excepción `socket.timeout` ahora es un alias de `TimeoutError`. (Contribuido por Christian Heimes en bpo-42413.)

Agregue la opción para crear sockets MPTCP con `IPPROTO_MPTCP` (Contribuido por Rui Cunha en bpo-43571.)

Agregue la opción `IP_RECVTOS` para recibir el tipo de servicio (ToS) o los campos DSCP / ECN (Contribuido por Georg Sauthoff en bpo-44077).

6.38 ssl

El módulo ssl requiere OpenSSL 1.1.1 o más reciente. (Contribuido por Christian Heimes en **PEP 644** y bpo-43669.)

El módulo ssl tiene soporte preliminar para OpenSSL 3.0.0 y la nueva opción `OP_IGNORE_UNEXPECTED_EOF`. (Contribuido por Christian Heimes en bpo-38820, bpo-43794, bpo-43788, bpo-43791, bpo-43799, bpo-43920, bpo-43789 y bpo-43811.)

La función obsoleta y el uso de constantes obsoletas ahora dan como resultado un `DeprecationWarning`. `ssl.SSLContext.options` tiene `OP_NO_SSLv2` y `OP_NO_SSLv3` configurados de forma predeterminada y, por lo tanto, no puede advertir sobre la configuración de la bandera nuevamente. El *deprecation section* tiene una lista de funciones obsoletas. (Contribuido por Christian Heimes en bpo-43880.)

El módulo ssl ahora tiene una configuración predeterminada más segura. Los cifrados sin confidencialidad directa o SHA-1 MAC están deshabilitados de forma predeterminada. El nivel de seguridad 2 prohíbe claves RSA, DH y ECC débiles

con menos de 112 bits de seguridad. `SSLContext` tiene por defecto la versión mínima del protocolo TLS 1.2. Los ajustes se basan en la investigación de Hynek Schlawack. (Contribuido por Christian Heimes en bpo-43998.)

Los protocolos obsoletos SSL 3.0, TLS 1.0 y TLS 1.1 ya no son compatibles oficialmente. Python no los bloquea activamente. Sin embargo, las opciones de compilación de OpenSSL, las configuraciones de distribución, los parches de proveedores y los conjuntos de cifrado pueden impedir un protocolo de enlace exitoso.

Agregue un parámetro *timeout* a la función `ssl.get_server_certificate()`. (Contribuido por Zackery Spytz en bpo-31870.)

El módulo ssl utiliza tipos de pila e inicialización multifase. (Contribuido por Christian Heimes en bpo-42333.)

Se ha agregado un nuevo indicador de verificación `VERIFY_X509_PARTIAL_CHAIN`. (Contribuido por l0x en bpo-40849.)

6.39 sqlite3

Agregue eventos de auditoría para `connect/handle()`, `enable_load_extension()` y `load_extension()`. (Contribuido por Erlend E. Aasland en bpo-43762.)

6.40 sys

Agregar atributo `sys.orig_argv`: la lista de los argumentos originales de la línea de comando pasados al ejecutable de Python. (Contribuido por Victor Stinner en bpo-23427.)

Agregue `sys.stdlib_module_names`, que contiene la lista de nombres de módulos de biblioteca estándar. (Contribuido por Victor Stinner en bpo-42955.)

6.41 _hilo

`_thread.interrupt_main()` ahora toma un número de señal opcional para simular (el predeterminado sigue siendo `signal.SIGINT`). (Contribuido por Antoine Pitrou en bpo-43356.)

6.42 enhebrar

Agregue `threading.gettrace()` y `threading.getprofile()` para recuperar las funciones establecidas por `threading.settrace()` y `threading.setprofile()` respectivamente. (Contribuido por Mario Corchero en bpo-42251.)

Agregue `threading.__excepthook__` para permitir recuperar el valor original de `threading.excepthook()` en caso de que esté configurado en un valor roto o diferente. (Contribuido por Mario Corchero en bpo-42308.)

6.43 rastrear

Las funciones `format_exception()`, `format_exception_only()` y `print_exception()` ahora pueden tomar un objeto de excepción como un argumento solo posicional. (Contribuido por Zackery Spytz y Matthias Bussonnier en bpo-26389.)

6.44 tipos

Reintroduzca las clases `types.EllipsisType`, `types.NoneType` y `types.NotImplementedType`, proporcionando un nuevo conjunto de tipos fácilmente interpretables por los verificadores de tipos. (Contribuido por Bas van Beek en bpo-41810.)

6.45 mecanografía

For major changes, see *Nuevas funciones relacionadas con las sugerencias de tipos*.

El comportamiento de `typing.Literal` se modificó para cumplir con **PEP 586** y para coincidir con el comportamiento de los verificadores de tipo estático especificados en el PEP.

1. `Literal` ahora elimina los parámetros duplicados.
2. Las comparaciones de igualdad entre objetos `Literal` ahora son independientes del orden.
3. `Literal` comparisons now respect types. For example, `Literal[0] == Literal[False]` previously evaluated to `True`. It is now `False`. To support this change, the internally used type cache now supports differentiating types.
4. Los objetos `Literal` ahora generarán una excepción `TypeError` durante las comparaciones de igualdad si alguno de sus parámetros no es hashable. Tenga en cuenta que declarar `Literal` con parámetros que no se pueden aplicar hash no arrojará un error:

```
>>> from typing import Literal
>>> Literal[{0}]
>>> Literal[{0}] == Literal[{False}]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

(Contribuido por Yurii Karabas en bpo-42345.)

Add new function `typing.is_typeddict()` to introspect if an annotation is a `typing.TypedDict`. (Contributed by Patrick Reader in bpo-41792.)

Subclasses of `typing.Protocol` which only have data variables declared will now raise a `TypeError` when checked with `isinstance` unless they are decorated with `runtime_checkable()`. Previously, these checks passed silently. Users should decorate their subclasses with the `runtime_checkable()` decorator if they want runtime protocols. (Contributed by Yurii Karabas in bpo-38908.)

Importing from the `typing.io` and `typing.re` submodules will now emit `DeprecationWarning`. These submodules have been deprecated since Python 3.8 and will be removed in a future version of Python. Anything belonging to those submodules should be imported directly from `typing` instead. (Contributed by Sebastian Rittau in bpo-38291.)

6.46 prueba de unidad

Agregue un nuevo método `assertNoLogs()` para complementar el `assertLogs()` existente. (Contribuido por Kit Yan Choi en bpo-39385.)

6.47 urllib.parse

Las versiones de Python anteriores a Python 3.10 permitían el uso de `;` y `&` como separadores de parámetros de consulta en `urllib.parse.parse_qs()` y `urllib.parse.parse_qsl()`. Debido a problemas de seguridad y para cumplir con las recomendaciones más recientes del W3C, esto se ha cambiado para permitir solo una clave separadora, con `&` como predeterminado. Este cambio también afecta a `cgi.parse()` y `cgi.parse_multipart()` ya que utilizan las funciones afectadas internamente. Para obtener más detalles, consulte su documentación respectiva. (Contribuido por Adam Goldschmidt, Senthil Kumaran y Ken Jin en bpo-42967.)

La presencia de caracteres de nueva línea o tabulación en partes de una URL permite algunas formas de ataques. Siguiendo la especificación WHATWG que actualiza: rfc: *3986*, la nueva línea ASCII `\n`, `\r` y los caracteres de tabulación `\t` son eliminados de la URL por el analizador en `urllib.parse` para prevenir tales ataques. Los caracteres de eliminación están controlados por una nueva variable de nivel de módulo `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (Ver bpo-43882)

6.48 xml

Agregue una clase `LexicalHandler` al módulo `xml.sax.handler`. (Contribuido por Jonathan Gossage y Zackery Spytz en bpo-35018.)

6.49 zipimport

Agregue métodos relacionados con **PEP 451**: `find_spec()`, `zipimport.zipimporter.create_module()` y `zipimport.zipimporter.exec_module()`. (Contribuido por Brett Cannon en bpo-42131.)

Agregue el método `invalidate_caches()`. (Contribuido por Desmond Cheong en bpo-14678.)

7 Optimizaciones

- Los constructores `str()`, `bytes()` y `bytearray()` ahora son más rápidos (alrededor del 30-40% para objetos pequeños). (Contribuido por Serhiy Storchaka en bpo-41334.)
- El módulo `runpy` ahora importa menos módulos. El tiempo de inicio del comando `python3 -m module-name` es 1,4 veces más rápido en promedio. En Linux, `python3 -I -m module-name` importa 69 módulos en Python 3.9, mientras que solo importa 51 módulos (-18) en Python 3.10. (Contribuido por Victor Stinner en bpo-41006 y bpo-41718.)
- La instrucción `LOAD_ATTR` ahora usa un nuevo mecanismo «por caché de código de operación». Es aproximadamente un 36% más rápido ahora para los atributos regulares y un 44% más rápido para las tragamonedas. (Contribuido por Pablo Galindo y Yury Selivanov en bpo-42093 y Guido van Rossum en bpo-42927, basado en ideas implementadas originalmente en PyPy y MicroPython).
- Al compilar Python con `--enable-optimizations`, ahora `-fno-semantic-interposition` se agrega tanto a la línea de compilación como a la de enlace. Esto acelera las compilaciones del intérprete de Python creado con `--enable-shared` con `gcc` hasta en un 30%. Consulte this article para obtener más detalles. (Contribuido por Victor Stinner y Pablo Galindo en bpo-38980.)

- Utilice un nuevo código de gestión del búfer de salida para los módulos `bz2` / `lzma` / `zlib` y agregue la función `.readall()` a la clase `_compression.DecompressReader`. La descompresión de bz2 ahora es 1.09x ~ 1.17x más rápida, la descompresión de lzma 1.20x ~ 1.32x más rápida, `GzipFile.read(-1)` 1.11x ~ 1.18x más rápida. (Contribuido por Ma Lin, revisado por Gregory P. Smith, en bpo-41486)
- When using stringized annotations, annotations dicts for functions are no longer created when the function is created. Instead, they are stored as a tuple of strings, and the function object lazily converts this into the annotations dict on demand. This optimization cuts the CPU time needed to define an annotated function by half. (Contributed by Yurii Karabas and Inada Naoki in bpo-42202.)
- Las funciones de búsqueda de subcadenas como `str1 in str2` y `str2.find(str1)` ahora utilizan a veces el algoritmo de búsqueda de cadenas «bidireccional» de Crochemore & Perrin para evitar el comportamiento cuadrático en cadenas largas. (Contribuido por Dennis Sweeney en bpo-41972)
- Add micro-optimizations to `_PyType_Lookup()` to improve type attribute cache lookup performance in the common case of cache hits. This makes the interpreter 1.04 times faster on average. (Contributed by Dino Viehland in bpo-43452.)
- The following built-in functions now support the faster **PEP 590** vectorcall calling convention: `map()`, `filter()`, `reversed()`, `bool()` and `float()`. (Contributed by Dong-hee Na and Jeroen Demeyer in bpo-43575, bpo-43287, bpo-41922, bpo-41873 and bpo-41870.)
- `BZ2File` performance is improved by removing internal `RLock`. This makes `BZ2File` thread unsafe in the face of multiple simultaneous readers or writers, just like its equivalent classes in `gzip` and `lzma` have always been. (Contributed by Inada Naoki in bpo-43785.)

8 Obsoleto

- Currently Python accepts numeric literals immediately followed by keywords, for example `0in x`, `1or x`, `0if 1else 2`. It allows confusing and ambiguous expressions like `[0x1for x in y]` (which can be interpreted as `[0x1 for x in y]` or `[0x1f or x in y]`). Starting in this release, a deprecation warning is raised if the numeric literal is immediately followed by one of keywords `and`, `else`, `for`, `if`, `in`, `is` and `or`. In future releases it will be changed to syntax warning, and finally to syntax error. (Contributed by Serhiy Storchaka in bpo-43833.)
- A partir de esta versión, habrá un esfuerzo concertado para comenzar a limpiar la semántica de importación antigua que se mantuvo para la compatibilidad con Python 2.7. Específicamente, `find_loader()` / `find_module()` (reemplazado por `find_spec()`), `load_module()` (reemplazado por `exec_module()`), `module_repr()` (que el sistema de importación se ocupa por usted), el atributo `__package__` (reemplazado por `__spec__.parent`), el atributo `__loader__` (reemplazado por `__spec__.loader`) y el atributo `__cached__` (reemplazado por `__spec__.cached`) se eliminará lentamente (así como otras clases y métodos en `importlib`). `ImportWarning` y / o `DeprecationWarning` se generarán según corresponda para ayudar a identificar el código que debe actualizarse durante esta transición.
- Todo el espacio de nombres `distutils` está obsoleto y se eliminará en Python 3.12. Consulte la sección *module changes* para obtener más información.
- Los argumentos que no son enteros para `random.randrange()` están en desuso. El `ValueError` está en desuso en favor de un `TypeError`. (Contribuido por Serhiy Storchaka y Raymond Hettinger en bpo-37319.)
- Los diversos métodos `load_module()` de `importlib` se han documentado como obsoletos desde Python 3.6, pero ahora también activarán un `DeprecationWarning`. Utilice `exec_module()` en su lugar. (Contribuido por Brett Cannon en bpo-26131.)
- `zimport.zipimporter.load_module()` ha quedado obsoleto en preferencia a `exec_module()`. (Contribuido por Brett Cannon en bpo-26131.)

- El uso de `load_module()` por parte del sistema de importación ahora activa un `ImportWarning` ya que se prefiere `exec_module()`. (Contribuido por Brett Cannon en bpo-26131.)
- El uso de `importlib.abc.MetaPathFinder.find_module()` y `importlib.abc.PathEntryFinder.find_module()` por parte del sistema de importación ahora activa un `ImportWarning`, ya que se prefieren `importlib.abc.MetaPathFinder.find_spec()` y `importlib.abc.PathEntryFinder.find_spec()`, respectivamente. Puede utilizar `importlib.util.spec_from_loader()` para ayudar en la migración. (Contribuido por Brett Cannon en bpo-42134.)
- El uso de `importlib.abc.PathEntryFinder.find_loader()` por parte del sistema de importación ahora activa un `ImportWarning`, ya que se prefiere `importlib.abc.PathEntryFinder.find_spec()`. Puede utilizar `importlib.util.spec_from_loader()` para ayudar en la migración. (Contribuido por Brett Cannon en bpo-43672.)
- The various implementations of `importlib.abc.MetaPathFinder.find_module()` (`importlib.machinery.BuiltinImporter.find_module()`, `importlib.machinery.FrozenImporter.find_module()`, `importlib.machinery.WindowsRegistryFinder.find_module()`, `importlib.machinery.PathFinder.find_module()`, `importlib.abc.MetaPathFinder.find_module()`), `importlib.abc.PathEntryFinder.find_module()` (`importlib.machinery.FileFinder.find_module()`), and `importlib.abc.PathEntryFinder.find_loader()` (`importlib.machinery.FileFinder.find_loader()`) now raise `DeprecationWarning` and are slated for removal in Python 3.12 (previously they were documented as deprecated in Python 3.4). (Contributed by Brett Cannon in bpo-42135.)
- `importlib.abc.Finder` está en desuso (incluido su único método, `find_module()`). Tanto `importlib.abc.MetaPathFinder` como `importlib.abc.PathEntryFinder` ya no heredan de la clase. Los usuarios deben heredar de una de estas dos clases según corresponda. (Contribuido por Brett Cannon en bpo-42135.)
- Las bajas de `imp`, `importlib.find_loader()`, `importlib.util.set_package_wrapper()`, `importlib.util.set_loader_wrapper()`, `importlib.util.module_for_loader()`, `pkgutil.ImpImporter` y `pkgutil.ImpLoader` se han actualizado para incluir Python 3.12 como la versión programada de eliminación (comenzaron a generar `DeprecationWarning` en versiones anteriores de Python). (Contribuido por Brett Cannon en bpo-43720.)
- El sistema de importación ahora usa el atributo `__spec__` en los módulos antes de recurrir a `module_repr()` para el método `__repr__()` de un módulo. La eliminación del uso de `module_repr()` está programada para Python 3.12. (Contribuido por Brett Cannon en bpo-42137.)
- `importlib.abc.Loader.module_repr()`, `importlib.machinery.FrozenLoader.module_repr()` y `importlib.machinery.BuiltinLoader.module_repr()` están en desuso y están programados para su eliminación en Python 3.12. (Contribuido por Brett Cannon en bpo-42136.)
- `sqlite3.OptimizedUnicode` ha sido indocumentado y obsoleto desde Python 3.3, cuando se convirtió en un alias para `str`. Ahora está en desuso, programado para su eliminación en Python 3.12. (Contribuido por Erlend E. Aasland en bpo-42264.)
- `asyncio.get_event_loop()` now emits a deprecation warning if there is no running event loop. In the future it will be an alias of `get_running_loop()`. `asyncio` functions which implicitly create `Future` or `Task` objects now emit a deprecation warning if there is no running event loop and no explicit *loop* argument is passed: `ensure_future()`, `wrap_future()`, `gather()`, `shield()`, `as_completed()` and constructors of `Future`, `Task`, `StreamReader`, `StreamReaderProtocol`. (Contributed by Serhiy Storchaka in bpo-39529.)
- La función incorporada no documentada `sqlite3.enable_shared_cache` ahora está en desuso, programada para su eliminación en Python 3.12. Su uso está fuertemente desaconsejado por la documentación de SQLite3. Consulte the SQLite3 docs para obtener más detalles. Si se debe usar una caché compartida, abra la base de datos en modo URI usando el parámetro de consulta `cache=shared`. (Contribuido por Erlend E. Aasland en bpo-24464.)

- Los siguientes métodos `threading` ahora están en desuso:
  - `threading.currentThread` => `threading.current_thread()`
  - `threading.activeCount` => `threading.active_count()`
  - `threading.Condition.notifyAll` => `threading.Condition.notify_all()`
  - `threading.Event.isSet` => `threading.Event.is_set()`
  - `threading.Thread.setName` => `threading.Thread.name`
  - `threading.thread.getName` => `threading.Thread.name`
  - `threading.Thread.isDaemon` => `threading.Thread.daemon`
  - `threading.Thread.setDaemon` => `threading.Thread.daemon`

  (Contributed by Jelle Zijlstra in gh-87889.)
- `pathlib.Path.link_to()` está en desuso y está programado para su eliminación en Python 3.12. Utilice `pathlib.Path.hardlink_to()` en su lugar. (Contribuido por Barney Gale en bpo-39950.)
- `cgi.log()` está obsoleto y está programado para su eliminación en Python 3.12. (Contribuido por Inada Naoki en bpo-41139.)
- Las siguientes funciones de `ssl` han quedado obsoletas desde Python 3.6, Python 3.7 u OpenSSL 1.1.0 y se eliminarán en 3.11:
  - `OP_NO_SSLv2`, `OP_NO_SSLv3`, `OP_NO_TLSv1`, `OP_NO_TLSv1_1`, `OP_NO_TLSv1_2` y `OP_NO_TLSv1_3` se reemplazan por `sslSSLContext.minimum_version` y `sslSSLContext.maximum_version`.
  - `PROTOCOL_SSLv2`, `PROTOCOL_SSLv3`, `PROTOCOL_SSLv23`, `PROTOCOL_TLSv1`, `PROTOCOL_TLSv1_1`, `PROTOCOL_TLSv1_2` y `PROTOCOL_TLS` están en desuso a favor de `PROTOCOL_TLS_CLIENT` y `PROTOCOL_TLS_SERVER`
  - `wrap_socket()` es reemplazado por `ssl.SSLContext.wrap_socket()`
  - `match_hostname()`
  - `RAND_pseudo_bytes()`, `RAND_egd()`
  - Las funciones NPN como `ssl.SSLSocket.selected_npn_protocol()` y `ssl.SSLContext.set_npn_protocols()` son reemplazadas por ALPN.
- La depuración de subprocesos (variable de entorno `PYTHONTHREADDEBUG`) está obsoleta en Python 3.10 y se eliminará en Python 3.12. Esta característica requiere un debug build of Python. (Contribuido por Victor Stinner en bpo-44584.)
- Importing from the `typing.io` and `typing.re` submodules will now emit `DeprecationWarning`. These submodules will be removed in a future version of Python. Anything belonging to these submodules should be imported directly from `typing` instead. (Contributed by Sebastian Rittau in bpo-38291.)

9 Eliminado

- Se eliminaron los métodos especiales `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` y `__rdivmod__` de la clase `complex`. Siempre levantaron un `TypeError`. (Contribuido por Serhiy Storchaka en bpo-41974.)
- Se ha eliminado el método `ParserBase.error()` del módulo `_markupbase` privado e indocumentado. `html.parser.HTMLParser` es la única subclase de `ParserBase` y su implementación `error()` ya se eliminó en Python 3.5. (Contribuido por Berker Peksag en bpo-31844.)
- Se eliminó el atributo `unicodedata.ucnhash_CAPI` que era un objeto interno de PyCapsule. La estructura `_PyUnicode_Name_CAPI` privada relacionada se movió a la API C interna. (Contribuido por Victor Stinner en bpo-42157.)
- Se eliminó el módulo `parser`, que quedó obsoleto en 3.9 debido al cambio al nuevo analizador PEG, así como todos los archivos fuente y de encabezado C que solo usaba el analizador anterior, incluidos `node.h`, `parser.h`, `graminit.h` y `grammar.h`.
- Se eliminaron las funciones de la API pública de C `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags` y `PyNode_Compile` que estaban en desuso en 3.9 debido al cambio al nuevo analizador PEG.
- Se eliminó el módulo `formatter`, que estaba en desuso en Python 3.4. Es algo obsoleto, poco usado y no probado. Originalmente estaba programado para ser eliminado en Python 3.6, pero tales eliminaciones se retrasaron hasta después de Python 2.7 EOL. Los usuarios existentes deben copiar cualquier clase que utilicen en su código. (Contribuido por Dong-hee Na y Terry J. Reedy en bpo-42299.)
- Se eliminó la función `PyModule_GetWarningsModule()` que ahora era inútil debido a que el módulo _warnings se convirtió en un módulo incorporado en 2.6. (Contribuido por Hai Shi en bpo-42599.)
- Elimine los alias obsoletos a collections-abstract-base-classes del módulo `collections`. (Contribuido por Victor Stinner en bpo-37324.)
- El parámetro `loop` se ha eliminado de la mayoría de `asyncio`"s API de alto nivel después de la desaprobación en Python 3.8. La motivación detrás de este cambio es múltiple:

  1. Esto simplifica la API de alto nivel.
  2. Las funciones en la API de alto nivel han obtenido implícitamente el bucle de eventos en ejecución del hilo actual desde Python 3.7. No es necesario pasar el bucle de eventos a la API en la mayoría de los casos de uso normales.
  3. El paso de bucles de eventos es propenso a errores, especialmente cuando se trata de bucles que se ejecutan en diferentes subprocesos.

  Note that the low-level API will still accept `loop`. See *Cambios en la API de Python* for examples of how to replace existing code.

  (Contribuido por Yurii Karabas, Andrew Svetlov, Yury Selivanov y Kyle Stanley en bpo-42392.)

10 Portar a Python 3.10

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

10.1 Cambios en la sintaxis de Python

- Deprecation warning is now emitted when compiling previously valid syntax if the numeric literal is immediately followed by a keyword (like in `0in x`). In future releases it will be changed to syntax warning, and finally to a syntax error. To get rid of the warning and make the code compatible with future releases just add a space between the numeric literal and the following keyword. (Contributed by Serhiy Storchaka in bpo-43833.)

10.2 Cambios en la API de Python

- Los parámetros *etype* de las funciones `format_exception()`, `format_exception_only()`, y `print_exception()` en el módulo `traceback` han sido renombradas a *exc*. (Contribuido por Zackery Spytz y Matthias Bussonnier en bpo-26389.)
- `atexit`: en la salida de Python, si falla una devolución de llamada registrada con `atexit.register()`, ahora se registra su excepción. Anteriormente, solo se registraban algunas excepciones y la última excepción siempre se ignoraba en silencio. (Contribuido por Victor Stinner en bpo-42639.)
- `collections.abc.Callable` genérico ahora aplana los parámetros de tipo, similar a lo que hace actualmente `typing.Callable`. Esto significa que `collections.abc.Callable[[int, str], str]` tendrá `__args__` de `(int, str, str)`; anteriormente esto era `([int, str], str)`. El código que accede a los argumentos a través de `typing.get_args()` o `__args__` debe tener en cuenta este cambio. Además, `TypeError` se puede generar para formas no válidas de parametrizar `collections.abc.Callable` que pueden haber pasado silenciosamente en Python 3.9. (Contribuido por Ken Jin en bpo-42195.)
- `socket.htons()` y `socket.ntohs()` ahora generan `OverflowError` en lugar de `DeprecationWarning` si el parámetro dado no cabe en un entero sin signo de 16 bits. (Contribuido por Erlend E. Aasland en bpo-42393.)
- El parámetro `loop` se ha eliminado de la mayoría de `asyncio`"s API de alto nivel después de la desaprobación en Python 3.8.

  Una corrutina que actualmente se ve así:

```
async def foo(loop):
    await asyncio.sleep(1, loop=loop)
```

  Debería ser reemplazado por esto:

```
async def foo():
    await asyncio.sleep(1)
```

  Si `foo()` fue diseñado específicamente *not* para ejecutarse en el bucle de eventos en ejecución del hilo actual (por ejemplo, ejecutándose en el bucle de eventos de otro hilo), considere usar `asyncio.run_coroutine_threadsafe()` en su lugar.

  (Contribuido por Yurii Karabas, Andrew Svetlov, Yury Selivanov y Kyle Stanley en bpo-42392.)
- El constructor `types.FunctionType` ahora hereda las incorporaciones actuales si el diccionario *globals* no tiene clave `"__builtins__"`, en lugar de usar `{"None": None}` como incorporaciones: el mismo comportamiento que las funciones `eval()` y `exec()`. La definición de una función con `def function(...):`

... en Python no se ve afectada, los globales no se pueden anular con esta sintaxis: también hereda las incorporaciones actuales. (Contribuido por Victor Stinner en bpo-42990.)

10.3 Cambios en la API de C

- Las funciones de API C `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags`, `PyNode_Compile` y el tipo utilizado por estas funciones, `struct _node`, se eliminaron debido al cambio al nuevo analizador PEG.

  La fuente debe compilarse ahora directamente en un objeto de código utilizando, por ejemplo, `Py_CompileString()`. A continuación, el objeto de código resultante se puede evaluar utilizando, por ejemplo, `PyEval_EvalCode()`.

  Específicamente:

  - Una llamada a `PyParser_SimpleParseStringFlags` seguida de `PyNode_Compile` se puede reemplazar llamando a `Py_CompileString()`.
  - No hay reemplazo directo para `PyParser_SimpleParseFileFlags`. Para compilar código a partir de un argumento `FILE *`, deberá leer el archivo en C y pasar el búfer resultante a `Py_CompileString()`.
  - Para compilar un archivo con un nombre de archivo `char *`, abra explícitamente el archivo, léalo y compile el resultado. Una forma de hacerlo es utilizando el módulo `io` con `PyImport_ImportModule()`, `PyObject_CallMethod()`, `PyBytes_AsString()` y `Py_CompileString()`, como se muestra a continuación. (Se omiten las declaraciones y el manejo de errores).

    ```
    io_module = Import_ImportModule("io");
    fileobject = PyObject_CallMethod(io_module, "open", "ss", filename, "rb");
    source_bytes_object = PyObject_CallMethod(fileobject, "read", "");
    result = PyObject_CallMethod(fileobject, "close", "");
    source_buf = PyBytes_AsString(source_bytes_object);
    code = Py_CompileString(source_buf, filename, Py_file_input);
    ```

  - Para los objetos `FrameObject`, el miembro `f_lasti` ahora representa un desplazamiento de código de palabra en lugar de un desplazamiento simple en la cadena de código de bytes. Esto significa que este número debe multiplicarse por 2 para usarse con API que esperan un desplazamiento de bytes en su lugar (como `PyCode_Addr2Line()`, por ejemplo). Tenga en cuenta también que el miembro `f_lasti` de los objetos `FrameObject` no se considera estable: utilice `PyFrame_GetLineNumber()` en su lugar.

11 Cambios en el código de bytes de CPython

- The `MAKE_FUNCTION` instruction now accepts either a dict or a tuple of strings as the function's annotations. (Contributed by Yurii Karabas and Inada Naoki in bpo-42202.)

12 Construir cambios

- **PEP 644**: Python ahora requiere OpenSSL 1.1.1 o más reciente. OpenSSL 1.0.2 ya no es compatible. (Contribuido por Christian Heimes en bpo-43669.)
- Las funciones C99 `snprintf()` y `vsnprintf()` ahora son necesarias para construir Python. (Contribuido por Victor Stinner en bpo-36020.)
- `sqlite3` requires SQLite 3.7.15 or higher. (Contributed by Sergey Fedoseev and Erlend E. Aasland in bpo-40744 and bpo-40810.)
- El módulo `atexit` ahora debe construirse siempre como un módulo integrado. (Contribuido por Victor Stinner en bpo-42639.)
- Agregue la opción `--disable-test-modules` al script `configure`: no cree ni instale módulos de prueba. (Contribuido por Xavier de Gaye, Thomas Petazzoni y Peixing Xin en bpo-27640.)
- Agregue `--with-wheel-pkg-dir=PATH option` al script `./configure`. Si se especifica, el módulo `ensurepip` busca paquetes de ruedas `setuptools` y `pip` en este directorio: si ambos están presentes, estos paquetes de ruedas se utilizan en lugar de los paquetes de ruedas asegurados.

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

  (Contribuido por Victor Stinner en bpo-42856.)
- Agregue un nuevo `configure --without-static-libpython option` para no construir la biblioteca estática `libpythonMAJOR.MINOR.a` y no instalar el archivo de objeto `python.o`.

  (Contribuido por Victor Stinner en bpo-43103.)
- El script `configure` ahora usa la utilidad `pkg-config`, si está disponible, para detectar la ubicación de los encabezados y bibliotecas Tcl / Tk. Como antes, esas ubicaciones se pueden especificar explícitamente con las opciones de configuración `--with-tcltk-includes` y `--with-tcltk-libs`. (Contribuido por Manolis Stamatogiannakis en bpo-42603.)
- Agregue la opción `--with-openssl-rpath` al script `configure`. La opción simplifica la construcción de Python con una instalación personalizada de OpenSSL, p. Ej. `./configure --with-openssl=/path/to/openssl --with-openssl-rpath=auto`. (Contribuido por Christian Heimes en bpo-43466.)

13 Cambios en la API de C

13.1 PEP 652: Mantenimiento del ABI estable

La ABI estable (interfaz binaria de aplicación) para módulos de extensión o incrustación de Python ahora está definida explícitamente. stable describe las garantías de estabilidad C API y ABI junto con las mejores prácticas para usar la ABI estable.

(Contribuido por Petr Viktorin en **PEP 652** y bpo-43795.)

13.2 Nuevas características

- El resultado de `PyNumber_Index()` ahora siempre tiene el tipo exacto `int`. Anteriormente, el resultado podría haber sido una instancia de una subclase de `int`. (Contribuido por Serhiy Storchaka en bpo-40792.)
- Agregue un nuevo miembro `orig_argv` a la estructura `PyConfig`: la lista de los argumentos originales de la línea de comandos pasados al ejecutable de Python. (Contribuido por Victor Stinner en bpo-23427.)
- Se han agregado las macros `PyDateTime_DATE_GET_TZINFO()` y `PyDateTime_TIME_GET_TZINFO()` para acceder a los atributos `tzinfo` de los objetos `datetime.datetime` y `datetime.time`. (Contribuido por Zackery Spytz en bpo-30155.)
- Agregue una función `PyCodec_Unregister()` para anular el registro de una función de búsqueda de códec. (Contribuido por Hai Shi en bpo-41842.)
- Se agregó la función `PyIter_Send()` para permitir el envío de valor al iterador sin generar la excepción `StopIteration`. (Contribuido por Vladimir Matveev en bpo-41756.)
- Agregue `PyUnicode_AsUTF8AndSize()` a la API C limitada. (Contribuido por Alex Gaynor en bpo-41784.)
- Agregue la función `PyModule_AddObjectRef()`: similar a `PyModule_AddObject()` pero no robe una referencia al valor en caso de éxito. (Contribuido por Victor Stinner en bpo-1635741.)
- Agregue las funciones `Py_NewRef()` y `Py_XNewRef()` para incrementar el recuento de referencia de un objeto y devolver el objeto. (Contribuido por Victor Stinner en bpo-42262.)
- Las funciones `PyType_FromSpecWithBases()` y `PyType_FromModuleAndSpec()` ahora aceptan una sola clase como argumento *bases*. (Contribuido por Serhiy Storchaka en bpo-42423.)
- La función `PyType_FromModuleAndSpec()` ahora acepta la ranura NULL `tp_doc`. (Contribuido por Hai Shi en bpo-41832.)
- La función `PyType_GetSlot()` puede aceptar static types. (Contribuido por Hai Shi y Petr Viktorin en bpo-41073.)
- Agregue una nueva función `PySet_CheckExact()` a la C-API para verificar si un objeto es una instancia de `set` pero no una instancia de un subtipo. (Contribuido por Pablo Galindo en bpo-43277.)
- Agregue `PyErr_SetInterruptEx()` que permite pasar un número de señal para simular. (Contribuido por Antoine Pitrou en bpo-43356.)
- La API C limitada ahora es compatible si Python is built in debug mode (si se define la macro `Py_DEBUG`). En la API C limitada, las funciones `Py_INCREF()` y `Py_DECREF()` ahora se implementan como llamadas de función opacas, en lugar de acceder directamente al miembro `PyObject.ob_refcnt`, si Python está construido en modo de depuración y la macro `Py_LIMITED_API` apunta a Python 3.10 o más reciente. Se hizo posible admitir la API C limitada en modo de depuración porque la estructura `PyObject` es la misma en el modo de liberación y depuración desde Python 3.8 (ver bpo-36465).

  La API C limitada todavía no es compatible con la compilación especial `--with-trace-refs` (macro `Py_TRACE_REFS`). (Contribuido por Victor Stinner en bpo-43688.)
- Agregue la función `Py_Is(x, y)` para probar si el objeto *x* object is the *y*, lo mismo que `x is y` en Python. Agregue también las funciones `Py_IsNone()`, `Py_IsTrue()`, `Py_IsFalse()` para probar si un objeto es, respectivamente, el singleton `None`, el singleton `True` o el singleton `False`. (Contribuido por Victor Stinner en bpo-43753.)
- Agregue nuevas funciones para controlar el recolector de basura desde el código C: `PyGC_Enable()`, `PyGC_Disable()`, `PyGC_IsEnabled()`. Estas funciones permiten activar, desactivar y consultar el estado del recolector de basura desde código C sin tener que importar el módulo `gc`.
- Agregue un nuevo indicador de tipo `Py_TPFLAGS_DISALLOW_INSTANTIATION` para no permitir la creación de instancias de tipo. (Contribuido por Victor Stinner en bpo-43916.)

- Agregue un nuevo indicador de tipo `Py_TPFLAGS_IMMUTABLETYPE` para crear objetos de tipo inmutables: los atributos de tipo no se pueden establecer ni eliminar. (Contribuido por Victor Stinner y Erlend E. Aasland en bpo-43908.)

13.3 Portar a Python 3.10

- La macro `PY_SSIZE_T_CLEAN` ahora debe definirse para usar los formatos `PyArg_ParseTuple()` y `Py_BuildValue()` que usan `#`: `es#`, `et#`, `s#`, `u#`, `y#`, `z#`, `U#` y `Z#`. Consulte Parsing arguments and building values y **PEP 353**. (Contribuido por Victor Stinner en bpo-40943.)
- Dado que `Py_REFCNT()` se cambia a la función estática en línea, `Py_REFCNT(obj) = new_refcnt` debe reemplazarse con `Py_SET_REFCNT(obj, new_refcnt)`: consulte `Py_SET_REFCNT()` (disponible desde Python 3.9). Para compatibilidad con versiones anteriores, esta macro se puede utilizar:

```
#if PY_VERSION_HEX < 0x030900A4
#  define Py_SET_REFCNT(obj, refcnt) ((Py_REFCNT(obj) = (refcnt)), (void)0)
#endif
```

  (Contribuido por Victor Stinner en bpo-39573.)
- Se había permitido llamar a `PyDict_GetItem()` sin GIL retenido por motivos históricos. Ya no está permitido. (Contribuido por Victor Stinner en bpo-40839.)
- `PyUnicode_FromUnicode(NULL, size)` y `PyUnicode_FromStringAndSize(NULL, size)` generan `DeprecationWarning` ahora. Utilice `PyUnicode_New()` para asignar un objeto Unicode sin datos iniciales. (Contribuido por Inada Naoki en bpo-36346.)
- La estructura `_PyUnicode_Name_CAPI` privada de la API `unicodedata.ucnhash_CAPI` de PyCapsule se ha movido a la API C interna. (Contribuido por Victor Stinner en bpo-42157.)
- Las funciones `Py_GetPath()`, `Py_GetPrefix()`, `Py_GetExecPrefix()`, `Py_GetProgramFullPath()`, `Py_GetPythonHome()` y `Py_GetProgramName()` ahora devuelven `NULL` si se llaman antes de `Py_Initialize()` (antes de que se inicialice Python). Utilice el nuevo Python Initialization Configuration API para obtener el Python Path Configuration.. (Contribuido por Victor Stinner en bpo-42260.)
- Las macros `PyList_SET_ITEM()`, `PyTuple_SET_ITEM()` y `PyCell_SET()` ya no se pueden utilizar como valor l o valor r. Por ejemplo, `x = PyList_SET_ITEM(a, b, c)` y `PyList_SET_ITEM(a, b, c) = x` ahora fallan con un error del compilador. Previene errores como la prueba `if (PyList_SET_ITEM (a, b, c) < 0) ...`. (Contribuido por Zackery Spytz y Victor Stinner en bpo-30459.)
- The non-limited API files `odictobject.h`, `parser_interface.h`, `picklebufobject.h`, `pyarena.h`, `pyctype.h`, `pydebug.h`, `pyfpe.h`, and `pytime.h` have been moved to the `Include/cpython` directory. These files must not be included directly, as they are already included in `Python.h`: Include Files. If they have been included directly, consider including `Python.h` instead. (Contributed by Nicholas Sim in bpo-35134.)
- Utilice el indicador de tipo `Py_TPFLAGS_IMMUTABLETYPE` para crear objetos de tipo inmutable. No confíe en `Py_TPFLAGS_HEAPTYPE` para decidir si un objeto de tipo es mutable o no; compruebe si `Py_TPFLAGS_IMMUTABLETYPE` está configurado en su lugar. (Contribuido por Victor Stinner y Erlend E. Aasland en bpo-43908.)
- The undocumented function `Py_FrozenMain` has been removed from the limited API. The function is mainly useful for custom builds of Python. (Contributed by Petr Viktorin in bpo-26241.)

13.4 Obsoleto

- La función `PyUnicode_InternImmortal()` ahora está en desuso y se eliminará en Python 3.12: use `PyUnicode_InternInPlace()` en su lugar. (Contribuido por Victor Stinner en bpo-41692.)

13.5 Eliminado

- Se eliminaron las funciones `Py_UNICODE_str*` que manipulaban cadenas `Py_UNICODE*`. (Contribuido por Inada Naoki en bpo-41123.)
  - `Py_UNICODE_strlen`: utilice `PyUnicode_GetLength()` o `PyUnicode_GET_LENGTH`
  - `Py_UNICODE_strcat`: utilice `PyUnicode_CopyCharacters()` o `PyUnicode_FromFormat()`
  - `Py_UNICODE_strcpy`, `Py_UNICODE_strncpy`: utilice `PyUnicode_CopyCharacters()` o `PyUnicode_Substring()`
  - `Py_UNICODE_strcmp`: utilice `PyUnicode_Compare()`
  - `Py_UNICODE_strncmp`: utilice `PyUnicode_Tailmatch()`
  - `Py_UNICODE_strchr`, `Py_UNICODE_strrchr`: utilice `PyUnicode_FindChar()`
- Eliminado `PyUnicode_GetMax()`. Migra a las API nuevas (**PEP 393**). (Contribuido por Inada Naoki en bpo-41103.)
- Eliminado `PyLong_FromUnicode()`. Migra a `PyLong_FromUnicodeObject()`. (Contribuido por Inada Naoki en bpo-41103.)
- Eliminado `PyUnicode_AsUnicodeCopy()`. Utilice `PyUnicode_AsUCS4Copy()` o `PyUnicode_AsWideCharString()` (contribución de Inada Naoki en bpo-41103).
- Variable `_Py_CheckRecursionLimit` eliminada: ha sido reemplazada por `ceval.recursion_limit` de la estructura `PyInterpreterState`. (Contribuido por Victor Stinner en bpo-41834.)
- Se eliminaron las macros `Py_ALLOW_RECURSION` y `Py_END_ALLOW_RECURSION` sin documentar y el campo `recursion_critical` de la estructura `PyInterpreterState`. (Contribuido por Serhiy Storchaka en bpo-41936.)
- Se eliminó la función `PyOS_InitInterrupts()` indocumentada. La inicialización de Python ya instala implícitamente controladores de señales: consulte `PyConfig.install_signal_handlers`. (Contribuido por Victor Stinner en bpo-41713.)
- Elimina la función `PyAST_Validate()`. Ya no es posible construir un objeto AST (tipo `mod_ty`) con la API C pública. La función ya estaba excluida de la API C limitada (**PEP 384**). (Contribuido por Victor Stinner en bpo-43244.)
- Elimine el archivo de encabezado `symtable.h` y las funciones no documentadas:
  - `PyST_GetScope()`
  - `PySymtable_Build()`
  - `PySymtable_BuildObject()`
  - `PySymtable_Free()`
  - `Py_SymtableString()`
  - `Py_SymtableStringObject()`

La función `Py_SymtableString()` fue parte de la ABI estable por error, pero no se pudo usar porque el archivo de encabezado `symtable.h` se excluyó de la API C limitada.

En su lugar, utilice el módulo Python `symtable`. (Contribuido por Victor Stinner en bpo-43244.)

- Elimine `PyOS_ReadlineFunctionPointer()` de los encabezados limitados de la API C y de `python3.dll`, la biblioteca que proporciona la ABI estable en Windows. Dado que la función toma un argumento `FILE*`, no se puede garantizar su estabilidad ABI. (Contribuido por Petr Viktorin en bpo-43868.)
- Elimine los archivos de encabezado `ast.h`, `asdl.h` y `Python-ast.h`. Estas funciones no estaban documentadas y se excluyeron de la API C limitada. La mayoría de los nombres definidos por estos archivos de encabezado no tenían el prefijo `Py` y, por lo tanto, podrían crear conflictos de nombres. Por ejemplo, `Python-ast.h` definió una macro `Yield` que estaba en conflicto con el nombre `Yield` utilizado por el encabezado `<winbase.h>` de Windows. En su lugar, utilice el módulo Python `ast`. (Contribuido por Victor Stinner en bpo-43244.)
- Elimine las funciones de compilador y analizador utilizando el tipo `struct _mod`, porque se eliminó la API de AST C pública:
  - `PyAST_Compile()`
  - `PyAST_CompileEx()`
  - `PyAST_CompileObject()`
  - `PyFuture_FromAST()`
  - `PyFuture_FromASTObject()`
  - `PyParser_ASTFromFile()`
  - `PyParser_ASTFromFileObject()`
  - `PyParser_ASTFromFilename()`
  - `PyParser_ASTFromString()`
  - `PyParser_ASTFromStringObject()`

  Estas funciones no estaban documentadas y se excluyeron de la API C limitada. (Contribuido por Victor Stinner en bpo-43244.)
- Elimine el archivo de encabezado `pyarena.h` con funciones:
  - `PyArena_New()`
  - `PyArena_Free()`
  - `PyArena_Malloc()`
  - `PyArena_AddPyObject()`

  Estas funciones no estaban documentadas, estaban excluidas de la API C limitada y el compilador solo las usaba internamente. (Contribuido por Victor Stinner en bpo-43244.)
- El miembro `PyThreadState.use_tracing` se ha eliminado para optimizar Python. (Contribuido por Mark Shannon en bpo-43760.)

Índice

P

V