What's New in Python

Versión 3.11.2

A. M. Kuchling

marzo 23, 2023

Python Software Foundation
Email: docs@python.org

Índice general

1 Resumen – Aspectos destacados de la versión **3**

2 Nuevas características **4**
2.1 PEP 657: Ubicaciones de errores detallados en rastreos . . . 4
2.2 PEP 654: Grupos de excepción y `except*` . . . 5
2.3 PEP 678: Las excepciones se pueden enriquecer con notas . . . 5
2.4 Mejoras en el iniciador de Windows `py.exe` . . . 5

3 Nuevas funciones relacionadas con las sugerencias de tipo **5**
3.1 PEP 646: Genéricos Variádicos . . . 6
3.2 PEP 655: Marcado de elementos `TypedDict` individuales como requeridos o no requeridos . . . 6
3.3 PEP 673: tipo `Self` . . . 6
3.4 PEP 675: tipo de cadena literal arbitraria . . . 7
3.5 PEP 681: Transformaciones de clases de datos . . . 8
3.6 PEP 563 puede no ser el futuro . . . 8

4 Otros cambios de idioma **8**

5 Otros cambios en la implementación de CPython **9**

6 Nuevos Módulos **10**

7 Módulos mejorados **10**
7.1 asíncio . . . 10
7.2 contextlib . . . 10
7.3 clases de datos . . . 10
7.4 fecha y hora . . . 11
7.5 enumeración . . . 11
7.6 fcntl . . . 12
7.7 fracciones . . . 12
7.8 herramientas funcionales . . . 12
7.9 hashlib . . . 12
7.10 IDLE y libre de inactividad . . . 13
7.11 inspeccionar . . . 13

7.12 lugar . . . 13
7.13 Inicio sesión . . . 13
7.14 Matemáticas . . . 14
7.15 operador . . . 14
7.16 sistema operativo . . . 14
7.17 rutalib . . . 14
7.18 re . . . 14
7.19 cerrar . . . 14
7.20 enchufe . . . 14
7.21 sqlite3 . . . 15
7.22 cuerda . . . 15
7.23 sistema . . . 15
7.24 configuración del sistema . . . 16
7.25 archivo temporal . . . 16
7.26 enhebrar . . . 16
7.27 tiempo . . . 16
7.28 tkinter . . . 16
7.29 rastrear . . . 16
7.30 mecanografía . . . 17
7.31 unicodedata . . . 18
7.32 prueba de unidad . . . 18
7.33 venv . . . 18
7.34 advertencias . . . 18
7.35 archivo zip . . . 18

8 Optimizaciones **18**

9 CPython más rápido **19**
9.1 Inicio más rápido . . . 19
9.2 Tiempo de ejecución más rápido . . . 20
9.3 Varios . . . 22
9.4 Preguntas más frecuentes . . . 22
9.5 Sobre . . . 22

10 Cambios en el código de bytes de CPython **23**
10.1 Nuevos códigos de operación . . . 23
10.2 Códigos de operación reemplazados . . . 24
10.3 Códigos de operación cambiados/eliminados . . . 25

11 Obsoleto **25**
11.1 Idioma/Construidos . . . 25
11.2 Módulos . . . 25
11.3 Biblioteca estándar . . . 26

12 Eliminación pendiente en Python 3.12 **27**

13 Remoto **28**

14 Migración a Python 3.11 **30**

15 Construir cambios **30**

16 Cambios en la API de C **31**
16.1 Nuevas características . . . 31
16.2 Migración a Python 3.11 . . . 33

16.3 Obsoleto 37
16.4 Eliminación pendiente en Python 3.12 38
16.5 Remoto 38

Índice **40**

Versión 3.11.2

Fecha marzo 23, 2023

Editor Pablo Galindo Salgado

Este artículo explica las nuevas características de Python 3.11, en comparación con 3.10.

Para obtener detalles completos, consulte changelog.

1 Resumen – Aspectos destacados de la versión

- Python 3.11 es entre un 10 y un 60 % más rápido que Python 3.10. En promedio, medimos un aumento de velocidad de 1.25x en el conjunto de pruebas de referencia estándar. Ver *CPython más rápido* para más detalles.

Nuevas funciones de sintaxis:

- *PEP 654: Grupos de excepción y except**

Nuevas funciones integradas:

- *PEP 678: Las excepciones se pueden enriquecer con notas*

Nuevos módulos de biblioteca estándar:

- **PEP 680**: `tomllib`: soporte para analizar TOML en la biblioteca estándar

Mejoras en el intérprete:

- *PEP 657: Ubicaciones de errores detallados en rastreos*
- Nueva opción de línea de comando `-P` y variable de entorno `PYTHONSAFEPATH` a *disable automatically prepending potentially unsafe paths* a `sys.path`

Nuevas funciones de escritura:

- *PEP 646: Genéricos Variádicos*
- *PEP 655: Marcado de elementos TypedDict individuales como requeridos o no requeridos*
- *PEP 673: tipo Self*
- *PEP 675: tipo de cadena literal arbitraria*
- *PEP 681: Transformaciones de clases de datos*

Importantes depreciaciones, eliminaciones y restricciones:

- **PEP 594**: *Many legacy standard library modules have been deprecated* y se eliminará en Python 3.13
- **PEP 624**: *Py_UNICODE encoder APIs have been removed*
- **PEP 670**: *Macros converted to static inline functions*

2 Nuevas características

2.1 PEP 657: Ubicaciones de errores detallados en rastreos

Al imprimir rastreos, el intérprete ahora señalará la expresión exacta que causó el error, en lugar de solo la línea. Por ejemplo:

```
Traceback (most recent call last):
  File "distance.py", line 11, in <module>
    print(manhattan_distance(p1, p2))
          ^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "distance.py", line 6, in manhattan_distance
    return abs(point_1.x - point_2.x) + abs(point_1.y - point_2.y)
                           ^^^^^^^^^
AttributeError: 'NoneType' object has no attribute 'x'
```

Las versiones anteriores del intérprete apuntaban solo a la línea, por lo que resultaba ambiguo qué objeto era `None`. Estos errores mejorados también pueden ser útiles cuando se trata de objetos `dict` profundamente anidados y múltiples llamadas a funciones:

```
Traceback (most recent call last):
  File "query.py", line 37, in <module>
    magic_arithmetic('foo')
  File "query.py", line 18, in magic_arithmetic
    return add_counts(x) / 25
           ^^^^^^^^^^^^^
  File "query.py", line 24, in add_counts
    return 25 + query_user(user1) + query_user(user2)
                ^^^^^^^^^^^^^^^^
  File "query.py", line 32, in query_user
    return 1 + query_count(db, response['a']['b']['c']['user'], retry=True)
                           ~~~~~~~~~~~~~~~~~~~~^^^^^
TypeError: 'NoneType' object is not subscriptable
```

Además de expresiones aritméticas complejas:

```
Traceback (most recent call last):
  File "calculation.py", line 54, in <module>
    result = (x / y / z) * (a / b / c)
             ~~~~~~^~~
ZeroDivisionError: division by zero
```

Además, la información utilizada por la función de rastreo mejorada está disponible a través de una API general, que se puede usar para correlacionar bytecode instructions con la ubicación del código fuente. Esta información se puede recuperar usando:

- El método `codeobject.co_positions()` en Python.
- La función `PyCode_Addr2Location()` en la API de C.

Ver **PEP 657** para más detalles. (Aportado por Pablo Galindo, Batuhan Taskaya y Ammar Askar en bpo-43950.)

Nota: Esta característica requiere almacenar las posiciones de las columnas en codeobjects, lo que puede resultar en un pequeño aumento en el uso de la memoria del intérprete y el uso del disco para los archivos de Python compilados. Para evitar almacenar la información adicional y desactivar la impresión de la información de seguimiento adicional, utilice la opción de línea de comando `-X no_debug_ranges` o la variable de entorno `PYTHONNODEBUGRANGES`.

2.2 PEP 654: Grupos de excepción y `except*`

PEP 654 presenta funciones de lenguaje que permiten que un programa genere y maneje múltiples excepciones no relacionadas simultáneamente. Los tipos integrados `ExceptionGroup` y `BaseExceptionGroup` permiten agrupar excepciones y generarlas juntas, y la nueva sintaxis `except*` generaliza `except` para hacer coincidir subgrupos de grupos de excepciones.

Ver **PEP 654** para más detalles.

(Aportado por Irit Katriel en bpo-45292. PEP escrito por Irit Katriel, Yury Selivanov y Guido van Rossum.)

2.3 PEP 678: Las excepciones se pueden enriquecer con notas

El método `add_note()` se agrega a `BaseException`. Se puede utilizar para enriquecer las excepciones con información de contexto que no está disponible en el momento en que se genera la excepción. Las notas añadidas aparecen en el rastreo predeterminado.

Ver **PEP 678** para más detalles.

(Aportado por Irit Katriel en bpo-45607. PEP escrito por Zac Hatfield-Dodds).

2.4 Mejoras en el iniciador de Windows `py.exe`

La copia de launcher incluida con Python 3.11 se ha actualizado significativamente. Ahora es compatible con la sintaxis de empresa/etiqueta tal como se define en **PEP 514** utilizando el argumento `-V:<company>/<tag>` en lugar del `-<major>.<minor>` limitado. Esto permite lanzar distribuciones distintas a `PythonCore`, la que está alojada en python.org.

Al usar los selectores `-V:`, se puede omitir la empresa o la etiqueta, pero se buscarán todas las instalaciones. Por ejemplo, `-V:OtherPython/` seleccionará la «mejor» etiqueta registrada para `OtherPython`, mientras que `-V:3.11` o `-V:/3.11` seleccionarán la «mejor» distribución con la etiqueta `3.11`.

Al usar los argumentos heredados `-<major>`, `-<major>.<minor>`, `-<major>-<bitness>` o `-<major>.<minor>-<bitness>`, se debe conservar todo el comportamiento existente de las versiones anteriores y solo se seleccionarán las versiones de `PythonCore`. Sin embargo, el sufijo `-64` ahora implica «no de 32 bits» (no necesariamente x86-64), ya que existen múltiples plataformas de 64 bits compatibles. Los tiempos de ejecución de 32 bits se detectan comprobando la etiqueta del tiempo de ejecución en busca de un sufijo `-32`. Todas las versiones de Python desde la 3.5 han incluido esto en sus compilaciones de 32 bits.

3 Nuevas funciones relacionadas con las sugerencias de tipo

Esta sección cubre los cambios principales que afectan las sugerencias de tipo **PEP 484** y el módulo `typing`.

3.1 PEP 646: Genéricos Variádicos

PEP 484 introdujo anteriormente `TypeVar`, lo que permite la creación de genéricos parametrizados con un solo tipo. **PEP 646** añade `TypeVarTuple`, permitiendo la parametrización con un número de tipos *arbitrary*. En otras palabras, un `TypeVarTuple` es una variable de tipo *variadic* que permite los genéricos *variadic*.

Esto permite una amplia variedad de casos de uso. En particular, permite parametrizar con el arreglo *shape* el tipo de estructuras similares a arreglos en bibliotecas de computación numérica como NumPy y TensorFlow. Los verificadores de tipo estático ahora podrán detectar errores relacionados con la forma en el código que usa estas bibliotecas.

Ver **PEP 646** para más detalles.

(Contribuido por Matthew Rahtz en bpo-43224, con contribuciones de Serhiy Storchaka y Jelle Zijlstra. PEP escrito por Mark Mendoza, Matthew Rahtz, Pradeep Kumar Srinivasan y Vincent Siles).

3.2 PEP 655: Marcado de elementos `TypedDict` individuales como requeridos o no requeridos

`Required` y `NotRequired` proporcionan una forma sencilla de marcar si deben estar presentes elementos individuales en un `TypedDict`. Anteriormente, esto solo era posible mediante la herencia.

Todos los campos siguen siendo obligatorios de forma predeterminada, a menos que el parámetro *total* se establezca en `False`, en cuyo caso todos los campos siguen sin ser obligatorios de forma predeterminada. Por ejemplo, lo siguiente especifica un `TypedDict` con una clave requerida y una no requerida:

```
class Movie(TypedDict):
   title: str
   year: NotRequired[int]

m1: Movie = {"title": "Black Panther", "year": 2018}  # OK
m2: Movie = {"title": "Star Wars"}  # OK (year is not required)
m3: Movie = {"year": 2022}  # ERROR (missing required field title)
```

La siguiente definición es equivalente:

```
class Movie(TypedDict, total=False):
   title: Required[str]
   year: int
```

Ver **PEP 655** para más detalles.

(Aportado por David Foster y Jelle Zijlstra en bpo-47087. PEP escrito por David Foster).

3.3 PEP 673: tipo `Self`

La nueva anotación `Self` proporciona una forma sencilla e intuitiva de anotar métodos que devuelven una instancia de su clase. Se comporta igual que el enfoque **specified in PEP 484** basado en `TypeVar`, pero es más conciso y más fácil de seguir.

Los casos de uso comunes incluyen constructores alternativos proporcionados como `classmethods` y métodos `__enter__()` que devuelven `self`:

```
class MyLock:
    def __enter__(self) -> Self:
        self.lock()
        return self
```

(continué en la próxima página)

(proviene de la página anterior)

```
    ...

class MyInt:
    @classmethod
    def fromhex(cls, s: str) -> Self:
        return cls(int(s, 16))

    ...
```

`Self` también se puede usar para anotar parámetros de método o atributos del mismo tipo que su clase envolvente.

Ver **PEP 673** para más detalles.

(Aportado por James Hilton-Balfe en bpo-46534. PEP escrito por Pradeep Kumar Srinivasan y James Hilton-Balfe).

3.4 PEP 675: tipo de cadena literal arbitraria

La nueva anotación `LiteralString` se puede usar para indicar que un parámetro de función puede ser de cualquier tipo de cadena literal. Esto permite que una función acepte tipos de cadenas literales arbitrarias, así como cadenas creadas a partir de otras cadenas literales. Los verificadores de tipos pueden hacer cumplir que las funciones confidenciales, como las que ejecutan declaraciones SQL o comandos de shell, se llamen solo con argumentos estáticos, lo que brinda protección contra ataques de inyección.

Por ejemplo, una función de consulta SQL podría anotarse de la siguiente manera:

```
def run_query(sql: LiteralString) -> ...
    ...

def caller(
    arbitrary_string: str,
    query_string: LiteralString,
    table_name: LiteralString,
) -> None:
    run_query("SELECT * FROM students")       # ok
    run_query(query_string)                   # ok
    run_query("SELECT * FROM " + table_name)  # ok
    run_query(arbitrary_string)               # type checker error
    run_query(                                # type checker error
        f"SELECT * FROM students WHERE name = {arbitrary_string}"
    )
```

Ver **PEP 675** para más detalles.

(Aportado por Jelle Zijlstra en bpo-47088. PEP escrito por Pradeep Kumar Srinivasan y Graham Bleaney).

3.5 PEP 681: Transformaciones de clases de datos

`dataclass_transform` se puede usar para decorar una clase, una metaclase o una función que en sí misma es un decorador. La presencia de `@dataclass_transform()` le dice a un verificador de tipo estático que el objeto decorado realiza una «magia» en tiempo de ejecución que transforma una clase, dándole comportamientos similares a `dataclass`.

Por ejemplo:

```
# The create_model decorator is defined by a library.
@typing.dataclass_transform()
def create_model(cls: Type[T]) -> Type[T]:
    cls.__init__ = ...
    cls.__eq__ = ...
    cls.__ne__ = ...
    return cls

# The create_model decorator can now be used to create new model classes:
@create_model
class CustomerModel:
    id: int
    name: str

c = CustomerModel(id=327, name="Eric Idle")
```

Ver **PEP 681** para más detalles.

(Aportado por Jelle Zijlstra en gh-91860. PEP escrito por Erik De Bonte y Eric Traut.)

3.6 PEP 563 puede no ser el futuro

PEP 563 Evaluación pospuesta de anotaciones (`from __future__ import annotations` future statement) que originalmente se planeó para su lanzamiento en Python 3.10 se suspendió indefinidamente. Consulte this message from the Steering Council para obtener más información.

4 Otros cambios de idioma

- Las expresiones de desempaquetado destacadas ahora se pueden usar en declaraciones `for`. (Consulte bpo-46725 para obtener más detalles).
- Ahora se permiten comprehensions asincrónicos dentro de las comprensiones en asynchronous functions. Las comprensiones externas implícitamente se vuelven asincrónicas en este caso. (Aportado por Serhiy Storchaka en bpo-33346.)
- Ahora se genera un `TypeError` en lugar de un `AttributeError` en declaraciones `with` y `contextlib.ExitStack.enter_context()` para objetos que no admiten el protocolo context manager, y en declaraciones `async with` y `contextlib.AsyncExitStack.enter_async_context()` para objetos que no admiten el protocolo asynchronous context manager. (Aportado por Serhiy Storchaka en bpo-12022 y bpo-44471).
- Se agregó `object.__getstate__()`, que proporciona la implementación predeterminada del método `__getstate__()`. Las instancias `copy`ing y `pickle`ing de subclases de tipos integrados `bytearray`, `set`, `frozenset`, `collections.OrderedDict`, `collections.deque`, `weakref.WeakSet` y `datetime.tzinfo` ahora copian y conservan atributos de instancia implementados como slots. (Aportado por Serhiy Storchaka en bpo-26579.)

- Se agregó una opción de línea de comando `-P` y una variable de entorno `PYTHONSAFEPATH`, que deshabilitan la anteposición automática a `sys.path` del directorio del script cuando se ejecuta un script, o el directorio actual cuando se usa `-c` y `-m`. Esto garantiza que `import` solo recopile la biblioteca estándar y los módulos instalados, y evita el remedo involuntario o malicioso de los módulos con los de un directorio local (y normalmente el usuario puede escribir). (Aportado por Victor Stinner en gh-57684.)
- Se agregó una opción `"z"` a formatspec que cambia negativo a cero positivo después de redondear a la precisión del formato. Ver **PEP 682** para más detalles. (Aportado por John Belmonte en gh-90153.)
- Ya no se aceptan bytes en `sys.path`. El soporte se interrumpió en algún momento entre Python 3.2 y 3.6, y nadie se dio cuenta hasta después del lanzamiento de Python 3.10.0. Además, recuperar la compatibilidad sería problemático debido a las interacciones entre `-b` y `sys.path_importer_cache` cuando hay una combinación de claves `str` y `bytes`. (Aportado por Thomas Grainger en gh-91181.)

5 Otros cambios en la implementación de CPython

- Los métodos especiales `__complex__()` para `complex` y `__bytes__()` para `bytes` se implementan para admitir los protocolos `typing.SupportsComplex` y `typing.SupportsBytes`. (Aportado por Mark Dickinson y Dong-hee Na en bpo-24234).
- `siphash13` se agrega como un nuevo algoritmo hash interno. Tiene propiedades de seguridad similares a `siphash24`, pero es un poco más rápido para entradas largas. `str`, `bytes` y algunos otros tipos ahora lo usan como algoritmo predeterminado para `hash()`. **PEP 552** hash-based .pyc files ahora también usa `siphash13`. (Aportado por Inada Naoki en bpo-29410.)
- Cuando una declaración `raise` sin parámetros vuelve a generar una excepción activa, el rastreo adjunto a esta excepción ahora siempre es `sys.exc_info()[1].__traceback__`. Esto significa que los cambios realizados en el rastreo en la cláusula `except` actual se reflejan en la excepción que se ha vuelto a generar. (Aportado por Irit Katriel en bpo-45711.)
- La representación del estado del intérprete de las excepciones manejadas (también conocido como `exc_info` o `_PyErr_StackItem`) ahora solo tiene el campo `exc_value`; Se han eliminado `exc_type` y `exc_traceback`, ya que se pueden derivar de `exc_value`. (Aportado por Irit Katriel en bpo-45711.)
- Se ha agregado un nuevo command line option, `AppendPath`, para el instalador de Windows. Se comporta de manera similar a `PrependPath`, pero agrega los directorios de instalación y scripts en lugar de anteponerlos. (Aportado por Bastian Neuburger en bpo-44934.)
- El campo `PyConfig.module_search_paths_set` ahora debe establecerse en `1` para que la inicialización use `PyConfig.module_search_paths` para inicializar `sys.path`. De lo contrario, la inicialización volverá a calcular la ruta y reemplazará los valores agregados a `module_search_paths`.
- La salida de la opción `--help` ahora cabe en 50 líneas/80 columnas. La información sobre las opciones Python environment variables y `-X` ahora está disponible utilizando los indicadores respectivos `--help-env` y `--help-xoptions`, y con el nuevo `--help-all`. (Contribución de Éric Araujo en bpo-46142.)
- La conversión entre `int` y `str` en bases que no sean 2 (binario), 4, 8 (octal), 16 (hexadecimal) o 32 como base 10 (decimal) ahora genera un `ValueError` si el número de dígitos en forma de cadena es superior a un límite para evitar posibles ataques de denegación de servicio debido a la complejidad algorítmica. Esta es una mitigación para CVE-2020-10735. Este límite se puede configurar o deshabilitar mediante la variable de entorno, el indicador de línea de comando o las API `sys`. Consulte la documentación de integer string conversion length limitation. El límite predeterminado es de 4300 dígitos en forma de cadena.

6 Nuevos Módulos

- `tomllib`: para analizar TOML. Ver **PEP 680** para más detalles. (Aportado por Taneli Hukkinen en bpo-40059.)
- `wsgiref.types`: tipos específicos de **WSGI** para la comprobación de tipos estáticos. (Aportado por Sebastian Rittau en bpo-42012.)

7 Módulos mejorados

7.1 asíncio

- Se agregó la clase `TaskGroup`, un asynchronous context manager que contiene un grupo de tareas que las esperará a todas al salir. Para código nuevo, se recomienda usar `create_task()` y `gather()` directamente. (Aportado por Yury Selivanov y otros en gh-90908.)
- Se agregó `timeout()`, un administrador de contexto asíncrono para establecer un tiempo de espera en operaciones asíncronas. Para código nuevo, se recomienda usar `wait_for()` directamente. (Aportado por Andrew Svetlov en gh-90927.)
- Se agregó la clase `Runner`, que expone la maquinaria utilizada por `run()`. (Aportado por Andrew Svetlov en gh-91218.)
- Se agregó la clase `Barrier` a las primitivas de sincronización en la biblioteca asyncio y la excepción `BrokenBarrierError` relacionada. (Aportado por Yves Duprat y Andrew Svetlov en gh-87518.)
- Se agregó el argumento de palabra clave *all_errors* a `asyncio.loop.create_connection()` para que se puedan generar varios errores de conexión como `ExceptionGroup`.
- Se agregó el método `asyncio.StreamWriter.start_tls()` para actualizar las conexiones basadas en secuencias existentes a TLS. (Aportado por Ian Good en bpo-34975.)
- Se agregaron funciones de socket de datagrama sin formato al bucle de eventos: `sock_sendto()`, `sock_recvfrom()` y `sock_recvfrom_into()`. Estos tienen implementaciones en `SelectorEventLoop` y `ProactorEventLoop`. (Aportado por Alex Grönholm en bpo-46805.)
- Se agregaron los métodos `cancelling()` y `uncancel()` a `Task`. Estos están destinados principalmente para uso interno, en particular por `TaskGroup`.

7.2 contextlib

- Se agregó un administrador de contexto `chdir()` no seguro en paralelo para cambiar el directorio de trabajo actual y luego restaurarlo al salir. Envoltorio simple alrededor de `chdir()`. (Aportación de Filipe Laíns en bpo-25625)

7.3 clases de datos

- Cambie la verificación de mutabilidad predeterminada del campo, permitiendo solo los valores predeterminados que son hashable en lugar de cualquier objeto que no sea una instancia de `dict`, `list` o `set`. (Aportado por Eric V. Smith en bpo-44674.)

7.4 fecha y hora

- Agregue `datetime.UTC`, un alias conveniente para `datetime.timezone.utc`. (Aportado por Kabir Kwatra en gh-91973.)
- `datetime.date.fromisoformat()`, `datetime.time.fromisoformat()` y `datetime.datetime.fromisoformat()` ahora se pueden usar para analizar la mayoría de los formatos ISO 8601 (excepto aquellos que admiten fracciones de horas y minutos). (Aportado por Paul Ganssle en gh-80010.)

7.5 enumeración

- Cambió el nombre de `EnumMeta` a `EnumType` (`EnumMeta` se mantuvo como un alias).
- Se agregó `StrEnum`, con miembros que se pueden usar como (y deben ser) cadenas.
- Se agregó `ReprEnum`, que solo modifica el `__repr__()` de los miembros mientras devuelve sus valores literales (en lugar de nombres) para `__str__()` y `__format__()` (utilizados por `str()`, `format()` y f-strings).
- Changed `IntEnum`, `IntFlag` and `StrEnum` to now inherit from `ReprEnum`, so their `str()` output now matches `format()` (both `str(AnIntEnum.ONE)` and `format(AnIntEnum.ONE)` return `'1'`, whereas before `str(AnIntEnum.ONE)` returned `'AnIntEnum.ONE'`.
- Se modificó `Enum.__format__()` (el valor predeterminado para `format()`, `str.format()` y f-strings) de enumeraciones con tipos combinados (por ejemplo, `int`, `str`) para incluir también el nombre de la clase en la salida, no solo la clave del miembro. Esto coincide con el comportamiento existente de `enum.Enum.__str__()`, devolviendo, p. `'AnEnum.MEMBER'` para una enumeración `AnEnum(str, Enum)` en lugar de solo `'MEMBER'`.
- Se agregó un nuevo parámetro de clase *boundary* a las enumeraciones `Flag` y la enumeración `FlagBoundary` con sus opciones, para controlar cómo manejar los valores de marca fuera de rango.
- Se agregó el decorador de enumeración `verify()` y la enumeración `EnumCheck` con sus opciones, para verificar las clases de enumeración contra varias restricciones específicas.
- Se agregaron los decoradores `member()` y `nonmember()` para garantizar que el objeto decorado no se convierta en un miembro de enumeración.
- Se agregó el decorador `property()`, que funciona como `property()` excepto para las enumeraciones. Use esto en lugar de `types.DynamicClassAttribute()`.
- Se agregó el decorador de enumeración `global_enum()`, que ajusta `__repr__()` y `__str__()` para mostrar valores como miembros de su módulo en lugar de la clase de enumeración. Por ejemplo, `'re.ASCII'` para el miembro `ASCII` de `re.RegexFlag` en lugar de `'RegexFlag.ASCII'`.
- `Flag` mejorado para admitir `len()`, iteración y `in`/`not in` en sus miembros. Por ejemplo, ahora funciona lo siguiente: `len(AFlag(3)) == 2 and list(AFlag(3)) == (AFlag.ONE, AFlag.TWO)`
- Se cambiaron `Enum` y `Flag` para que los miembros ahora se definan antes de llamar a `__init_subclass__()`; `dir()` ahora incluye métodos, etc., de tipos de datos combinados.
- Se modificó `Flag` para considerar solo los valores primarios (potencia de dos) canónicos, mientras que los valores compuestos (`3`, `6`, `10`, etc.) se consideran alias; las banderas invertidas son forzadas a su equivalente positivo.

7.6 fcntl

- On FreeBSD, the `F_DUP2FD` and `F_DUP2FD_CLOEXEC` flags respectively are supported, the former equals to `dup2` usage while the latter set the `FD_CLOEXEC` flag in addition.

7.7 fracciones

- Compatibilidad con la inicialización al estilo **PEP 515** de `Fraction` desde una cadena. (Aportado por Sergey B Kirpichev en bpo-44258.)
- `Fraction` ahora implementa un método `__int__`, de modo que pasa una verificación `isinstance(some_fraction, typing.SupportsInt)`. (Contribuido por Mark Dickinson en bpo-44547.)

7.8 herramientas funcionales

- `functools.singledispatch()` ahora admite `types.UnionType` y `typing.Union` como anotaciones en el argumento de envío.:

```
>>> from functools import singledispatch
>>> @singledispatch
... def fun(arg, verbose=False):
...     if verbose:
...         print("Let me just say,", end=" ")
...     print(arg)
...
>>> @fun.register
... def _(arg: int | float, verbose=False):
...     if verbose:
...         print("Strength in numbers, eh?", end=" ")
...     print(arg)
...
>>> from typing import Union
>>> @fun.register
... def _(arg: Union[list, set], verbose=False):
...     if verbose:
...         print("Enumerate this:")
...     for i, elem in enumerate(arg):
...         print(i, elem)
...
```

(Aportado por Yurii Karabas en bpo-46014.)

7.9 hashlib

- `hashlib.blake2b()` y `hashlib.blake2s()` ahora prefieren libb2 a la copia proporcionada por Python. (Aportado por Christian Heimes en bpo-47095.)
- El módulo interno `_sha3` con algoritmos SHA3 y SHAKE ahora usa *tiny_sha3* en lugar de *Keccak Code Package* para reducir el código y el tamaño binario. El módulo `hashlib` prefiere implementaciones SHA3 y SHAKE optimizadas de OpenSSL. El cambio afecta solo a las instalaciones sin compatibilidad con OpenSSL. (Aportado por Christian Heimes en bpo-47098.)
- Agregue `hashlib.file_digest()`, una función de ayuda para el hash eficiente de archivos u objetos similares a archivos. (Aportado por Christian Heimes en gh-89313.)

7.10 IDLE y libre de inactividad

- Aplicar resaltado de sintaxis a archivos `.pyi`. (Aportado por Alex Waygood y Terry Jan Reedy en bpo-45447).
- Incluya avisos al guardar Shell con entradas y salidas. (Aportado por Terry Jan Reedy en gh-95191.)

7.11 inspeccionar

- Agregue `getmembers_static()` para devolver todos los miembros sin activar la búsqueda dinámica a través del protocolo descriptor. (Contribuido por Weipeng Hong en bpo-30533.)
- Agregue `ismethodwrapper()` para verificar si el tipo de un objeto es `MethodWrapperType`. (Aportado por Hakan Çelik en bpo-29418.)
- Cambie las funciones relacionadas con el marco en el módulo `inspect` para devolver nuevas instancias de clase `FrameInfo` y `Traceback` (compatibles con las interfaces similares a named tuple anteriores) que incluyen la información de posición **PEP 657** extendida (número de línea final, columna y columna final). Las funciones afectadas son:
  - `inspect.getframeinfo()`
  - `inspect.getouterframes()`
  - `inspect.getinnerframes()`,
  - `inspect.stack()`
  - `inspect.trace()`

  (Aportado por Pablo Galindo en gh-88116.)

7.12 lugar

- Agregue `locale.getencoding()` para obtener la codificación de configuración regional actual. Es similar a `locale.getpreferredencoding(False)` pero ignora Python UTF-8 Mode.

7.13 Inicio sesión

- Se agregó `getLevelNamesMapping()` para devolver una asignación de nombres de nivel de registro (por ejemplo, `'CRITICAL'`) a los valores de su levels correspondiente (por ejemplo, `50`, de forma predeterminada). (Aportado por Andrei Kulakovin en gh-88024.)
- Se agregó un método `createSocket()` a `SysLogHandler` para que coincida con `SocketHandler.createSocket()`. Se llama automáticamente durante la inicialización del controlador y cuando se emite un evento, si no hay un socket activo. (Aportado por Kirill Pinchuk en gh-88457.)

7.14 Matemáticas

- Suma `math.exp2()`: devuelve 2 elevado a la potencia de x. (Aportado por Gideon Mitchell en bpo-45917.)
- Agregue `math.cbrt()`: devuelva la raíz cúbica de x. (Aportado por Ajith Ramachandran en bpo-44357.)
- Se cambió el comportamiento de dos casos de esquina `math.pow()`, para mantener la coherencia con la especificación IEEE 754. Las operaciones `math.pow(0.0, -math.inf)` y `math.pow(-0.0, -math.inf)` ahora devuelven `inf`. Anteriormente plantearon `ValueError`. (Contribuido por Mark Dickinson en bpo-44339.)
- El valor `math.nan` ahora está siempre disponible. (Aportado por Victor Stinner en bpo-46917.)

7.15 operador

- Se ha añadido una nueva función `operator.call`, de forma que `operator.call(obj, *args, **kwargs) == obj(*args, **kwargs)`. (Aportado por Antony Lee en bpo-44019.)

7.16 sistema operativo

- En Windows, `os.urandom()` ahora usa `BCryptGenRandom()`, en lugar de `CryptGenRandom()`, que está en desuso. (Aportado por Dong-hee Na en bpo-44611.)

7.17 rutalib

- `glob()` y `rglob()` solo devuelven directorios si *pattern* termina con un separador de componentes de nombre de ruta: `sep` o `altsep`. (Aportado por Eisuke Kawasima en bpo-22276 y bpo-33392).

7.18 re

- La agrupación atómica (`(?>...)`) y los cuantificadores posesivos (`*+`, `++`, `?+`, `{m,n}+`) ahora son compatibles con las expresiones regulares. (Aportado por Jeffrey C. Jacobs y Serhiy Storchaka en bpo-433030).

7.19 cerrar

- Agregue el parámetro opcional *dir_fd* en `shutil.rmtree()`. (Aportado por Serhiy Storchaka en bpo-46245.)

7.20 enchufe

- Agregue compatibilidad con CAN Socket para NetBSD. (Aportado por Thomas Klausner en bpo-30512.)
- `create_connection()` tiene una opción para generar, en caso de falla de conexión, un `ExceptionGroup` que contiene todos los errores en lugar de generar solo el último error. (Aportado por Irit Katriel en bpo-29980.)

7.21 sqlite3

- Ahora puede deshabilitar el autorizador pasando `None` a `set_authorizer()`. (Aportado por Erlend E. Aasland en bpo-44491.)
- El nombre de intercalación `create_collation()` ahora puede contener cualquier carácter Unicode. Los nombres de intercalación con caracteres no válidos ahora generan `UnicodeEncodeError` en lugar de `sqlite3.ProgrammingError`. (Aportado por Erlend E. Aasland en bpo-44688.)
- Las excepciones `sqlite3` ahora incluyen el código de error extendido de SQLite como `sqlite_errorcode` y el nombre de error de SQLite como `sqlite_errorname`. (Aportado por Aviv Palivoda, Daniel Shahaf y Erlend E. Aasland en bpo-16379 y bpo-24139).
- Agregue `setlimit()` y `getlimit()` a `sqlite3.Connection` para configurar y obtener límites de SQLite por conexión. (Aportado por Erlend E. Aasland en bpo-45243.)
- `sqlite3` ahora establece `sqlite3.threadsafety` en función del modo de subprocesamiento predeterminado con el que se ha compilado la biblioteca SQLite subyacente. (Aportado por Erlend E. Aasland en bpo-45613.)
- Las devoluciones de llamada de `sqlite3` C ahora usan excepciones que no se pueden generar si las devoluciones de llamada están habilitadas. Los usuarios ahora pueden registrar un `unraisable hook handler` para mejorar su experiencia de depuración. (Aportado por Erlend E. Aasland en bpo-45828.)
- Recuperar a través de reversión ya no genera `InterfaceError`. En cambio, dejamos que la biblioteca SQLite maneje estos casos. (Aportado por Erlend E. Aasland en bpo-44092.)
- Agregue `serialize()` y `deserialize()` a `sqlite3.Connection` para serializar y deserializar bases de datos. (Aportado por Erlend E. Aasland en bpo-41930.)
- Agregue `create_window_function()` a `sqlite3.Connection` para crear funciones de ventana agregadas. (Aportado por Erlend E. Aasland en bpo-34916.)
- Agregue `blobopen()` a `sqlite3.Connection`. `sqlite3.Blob` permite operaciones de E/S incrementales en blobs. (Contribuido por Aviv Palivoda y Erlend E. Aasland en bpo-24905).

7.22 cuerda

- Agregue `get_identifiers()` y `is_valid()` a `string.Template`, que devuelven respectivamente todos los marcadores de posición válidos y si hay algún marcador de posición no válido presente. (Aportado por Ben Kehoe en gh-90465.)

7.23 sistema

- `sys.exc_info()` now derives the `type` and `traceback` fields from the `value` (the exception instance), so when an exception is modified while it is being handled, the changes are reflected in the results of subsequent calls to `exc_info()`. (Contributed by Irit Katriel in bpo-45711.)
- Agregue `sys.exception()` que devuelve la instancia de excepción activa (equivalente a `sys.exc_info()[1]`). (Aportado por Irit Katriel en bpo-46328.)
- Agregue el indicador `sys.flags.safe_path`. (Aportado por Victor Stinner en gh-57684.)

7.24 configuración del sistema

- Se agregaron tres nuevos installation schemes (*posix_venv*, *nt_venv* y *venv*) y se usan cuando Python crea nuevos entornos virtuales o cuando se ejecuta desde un entorno virtual. Los primeros dos esquemas (*posix_venv* y *nt_venv*) son específicos del sistema operativo para Windows y no Windows, el *venv* es esencialmente un alias para uno de ellos según el sistema operativo en el que se ejecuta Python. Esto es útil para los distribuidores posteriores que modifican `sysconfig.get_preferred_scheme()`. El código de terceros que crea nuevos entornos virtuales debe usar el nuevo esquema de instalación *venv* para determinar las rutas, al igual que `venv`. (Aportado por Miro Hrončok en bpo-45413.)

7.25 archivo temporal

- Los objetos `SpooledTemporaryFile` ahora implementan completamente los métodos de `io.BufferedIOBase` o `io.TextIOBase` (según el modo de archivo). Esto les permite trabajar correctamente con API que esperan objetos similares a archivos, como módulos de compresión. (Aportado por Carey Metcalfe en gh-70363.)

7.26 enhebrar

- En Unix, si la función `sem_clockwait()` está disponible en la biblioteca C (glibc 2.30 y posterior), el método `threading.Lock.acquire()` ahora usa el reloj monotónico (`time.CLOCK_MONOTONIC`) para el tiempo de espera, en lugar de usar el reloj del sistema (`time.CLOCK_REALTIME`), para no verse afectado por cambios en el reloj del sistema. (Aportado por Victor Stinner en bpo-41710.)

7.27 tiempo

- En Unix, `time.sleep()` ahora usa la función `clock_nanosleep()` o `nanosleep()`, si está disponible, que tiene una resolución de 1 nanosegundo (10^{-9} segundos), en lugar de usar `select()` que tiene una resolución de 1 microsegundo (10^{-6} segundos). (Aportado por Benjamin Szőke y Victor Stinner en bpo-21302.)
- En Windows 8.1 y posteriores, `time.sleep()` ahora usa un temporizador de espera basado en high-resolution timers que tiene una resolución de 100 nanosegundos (10^{-7} segundos). Anteriormente tenía una resolución de 1 milisegundo (10^{-3} segundos). (Aportado por Benjamin Szőke, Dong-hee Na, Eryk Sun y Victor Stinner en bpo-21302 y bpo-45429).

7.28 tkinter

- Se agregó el método `info_patchlevel()` que devuelve la versión exacta de la biblioteca Tcl como una tupla con nombre similar a `sys.version_info`. (Aportado por Serhiy Storchaka en gh-91827.)

7.29 rastrear

- Agregue `traceback.StackSummary.format_frame_summary()` para permitir que los usuarios anulen qué marcos aparecen en el rastreo y cómo están formateados. (Aportado por Ammar Askar en bpo-44569.)
- Agregue `traceback.TracebackException.print()`, que imprime la instancia `TracebackException` formateada en un archivo. (Aportado por Irit Katriel en bpo-33809.)

7.30 mecanografía

Para conocer los cambios importantes, consulte *Nuevas funciones relacionadas con las sugerencias de tipo*.

- Agregue `typing.assert_never()` y `typing.Never`. `typing.assert_never()` es útil para pedirle a un verificador de tipos que confirme que no se puede acceder a una línea de código. En tiempo de ejecución, genera un `AssertionError`. (Aportado por Jelle Zijlstra en gh-90633.)
- Agregue `typing.reveal_type()`. Esto es útil para preguntarle a un verificador de tipos qué tipo ha inferido para una expresión dada. En tiempo de ejecución imprime el tipo del valor recibido. (Aportado por Jelle Zijlstra en gh-90572.)
- Agregue `typing.assert_type()`. Esto es útil para pedirle a un verificador de tipos que confirme que el tipo que ha inferido para una expresión dada coincide con el tipo dado. En tiempo de ejecución, simplemente devuelve el valor recibido. (Aportado por Jelle Zijlstra en gh-90638.)
- Los tipos `typing.TypedDict` ahora pueden ser genéricos. (Aportado por Samodya Abeysiriwardane en gh-89026.)
- Los tipos `NamedTuple` ahora pueden ser genéricos. (Aportado por Serhiy Storchaka en bpo-43923.)
- Permitir subclases de `typing.Any`. Esto es útil para evitar errores de verificación de tipos relacionados con clases altamente dinámicas, como simulacros. (Aportado por Shantanu Jain en gh-91154.)
- El decorador `typing.final()` ahora establece el atributo `__final__` en el objeto decorado. (Aportado por Jelle Zijlstra en gh-90500.)
- La función `typing.get_overloads()` se puede utilizar para la introspección de las sobrecargas de una función. `typing.clear_overloads()` se puede utilizar para borrar todas las sobrecargas registradas de una función. (Aportado por Jelle Zijlstra en gh-89263.)
- The `__init__()` method of `Protocol` subclasses is now preserved. (Contributed by Adrian Garcia Badarasco in gh-88970.)
- La representación de tipos de tuplas vacías (`Tuple[()]`) se simplifica. Esto afecta la introspección, p. `get_args(Tuple[()])` ahora se evalúa como `()` en lugar de `((),)`. (Aportado por Serhiy Storchaka en gh-91137.)
- Afloje los requisitos de tiempo de ejecución para las anotaciones de tipo eliminando la verificación invocable en la función privada `typing._type_check`. (Aportado por Gregory Beauregard en gh-90802.)
- `typing.get_type_hints()` ahora admite la evaluación de cadenas como referencias directas en PEP 585 generic aliases. (Aportado por Niklas Rosenstein en gh-85542.)
- `typing.get_type_hints()` ya no agrega `Optional` a los parámetros con `None` como predeterminado. (Aportado por Nikita Sobolev en gh-90353.)
- `typing.get_type_hints()` ahora admite la evaluación de anotaciones `ClassVar` con cadenas desnudas. (Aportado por Gregory Beauregard en gh-90711.)
- `typing.no_type_check()` ya no modifica clases y funciones externas. Ahora también marca correctamente los métodos de clase para que no se verifique el tipo. (Aportado por Nikita Sobolev en gh-90729.)

7.31 unicodedata

- The Unicode database has been updated to version 14.0.0. (Contributed by Benjamin Peterson in bpo-45190).

7.32 prueba de unidad

- Se agregaron los métodos `enterContext()` y `enterClassContext()` de la clase `TestCase`, el método `enterAsyncContext()` de la clase `IsolatedAsyncioTestCase` y la función `unittest.enterModuleContext()`. (Aportado por Serhiy Storchaka en bpo-45046.)

7.33 venv

- Cuando se crean nuevos entornos virtuales de Python, el *venv* sysconfig installation scheme se utiliza para determinar las rutas dentro del entorno. Cuando Python se ejecuta en un entorno virtual, el mismo esquema de instalación es el predeterminado. Eso significa que los distribuidores intermedios pueden cambiar el esquema de instalación de sysconfig predeterminado sin cambiar el comportamiento de los entornos virtuales. El código de terceros que también crea nuevos entornos virtuales debería hacer lo mismo. (Aportado por Miro Hrončok en bpo-45413.)

7.34 advertencias

- `warnings.catch_warnings()` ahora acepta argumentos para `warnings.simplefilter()`, lo que proporciona una forma más concisa de ignorar localmente las advertencias o convertirlas en errores. (Aportado por Zac Hatfield-Dodds en bpo-47074).

7.35 archivo zip

- Se agregó compatibilidad para especificar la codificación de nombres de miembros para leer metadatos en los encabezados de archivos y directorios de `ZipFile`. (Aportado por Stephen J. Turnbull y Serhiy Storchaka en bpo-28080).
- Se agregó `ZipFile.mkdir()` para crear nuevos directorios dentro de archivos ZIP. (Aportado por Sam Ezeh en gh-49083.)
- Se agregaron `stem`, `suffix` y `suffixes` a `zipfile.Path`. (Aportado por Miguel Brito en gh-88261.)

8 Optimizaciones

Esta sección cubre optimizaciones específicas independientes del proyecto *CPython más rápido*, que se trata en su propia sección.

- El compilador ahora optimiza printf-style % formatting simple en cadenas literales que contienen solo los códigos de formato `%s`, `%r` y `%a` y lo hace tan rápido como una expresión f-string correspondiente. (Aportado por Serhiy Storchaka en bpo-28307.)
- La división de enteros (`//`) está mejor ajustada para la optimización por parte de los compiladores. Ahora es un 20 % más rápido en x86-64 cuando se divide un `int` por un valor menor que `2**30`. (Aportado por Gregory P. Smith y Tim Peters en gh-90564.)
- `sum()` ahora es casi un 30 % más rápido para números enteros más pequeños que `2**30`. (Aportado por Stefan Behnel en gh-68264.)

- El cambio de tamaño de las listas está simplificado para el caso común, acelerando `list.append()` en ≈15 % y list comprehensions simples hasta en un 20-30 % (Contribuido por Dennis Sweeney en gh-91165).
- Los diccionarios no almacenan valores hash cuando todas las claves son objetos Unicode, lo que reduce el tamaño de `dict`. Por ejemplo, `sys.getsizeof(dict.fromkeys("abcdefg"))` se reduce de 352 bytes a 272 bytes (un 23 % más pequeño) en plataformas de 64 bits. (Aportado por Inada Naoki en bpo-46845.)
- El uso de `asyncio.DatagramProtocol` ahora es mucho más rápido cuando se transfieren archivos grandes a través de UDP, con velocidades 100 veces más altas para un archivo de ≈60 MiB. (Aportado por msoxzw en gh-91487.)
- Las funciones `math comb()` y `perm()` ahora son ≈10 veces más rápidas para argumentos grandes (con una aceleración mayor para *k* más grandes). (Aportado por Serhiy Storchaka en bpo-37295.)
- Las funciones `statistics mean(),variance()` y `stdev()` ahora consumen iteradores en una sola pasada en lugar de convertirlos primero en `list`. Esto es el doble de rápido y puede ahorrar una cantidad considerable de memoria. (Aportado por Raymond Hettinger en gh-90415.)
- `unicodedata.normalize()` ahora normaliza cadenas ASCII puras en tiempo constante. (Aportado por Dong-hee Na en bpo-44987.)

9 CPython más rápido

CPython 3.11 is an average of 25% faster than CPython 3.10 as measured with the pyperformance benchmark suite, when compiled with GCC on Ubuntu Linux. Depending on your workload, the overall speedup could be 10-60%.

This project focuses on two major areas in Python: *Inicio más rápido* and *Tiempo de ejecución más rápido*. Optimizations not covered by this project are listed separately under *Optimizaciones*.

9.1 Inicio más rápido

Importaciones congeladas / Objetos de código estático

Python caches bytecode in the __pycache__ directory to speed up module loading.

Previamente en 3.10, la ejecución del módulo de Python se veía así:

```
Read __pycache__ -> Unmarshal -> Heap allocated code object -> Evaluate
```

In Python 3.11, the core modules essential for Python startup are «frozen». This means that their codeobjects (and bytecode) are statically allocated by the interpreter. This reduces the steps in module execution process to:

```
Statically allocated code object -> Evaluate
```

El inicio del intérprete ahora es un 10-15 % más rápido en Python 3.11. Esto tiene un gran impacto para los programas de ejecución corta que usan Python.

(Contributed by Eric Snow, Guido van Rossum and Kumar Aditya in many issues.)

9.2 Tiempo de ejecución más rápido

Marcos de Python más baratos y perezosos

Python frames, holding execution information, are created whenever Python calls a Python function. The following are new frame optimizations:

- Simplificó el proceso de creación de marcos.
- Se evitó la asignación de memoria al reutilizar generosamente el espacio de marcos en la pila C.
- Simplificó la estructura del marco interno para que contenga solo información esencial. Los marcos contenían previamente información adicional de gestión de memoria y depuración.

Old-style frame objects are now created only when requested by debuggers or by Python introspection functions such as `sys._getframe()` and `inspect.currentframe()`. For most user code, no frame objects are created at all. As a result, nearly all Python functions calls have sped up significantly. We measured a 3-7% speedup in pyperformance.

(Aportado por Mark Shannon en bpo-44590.)

Llamadas a funciones de Python en línea

Durante una llamada de función de Python, Python llamará a una función C de evaluación para interpretar el código de esa función. Esto limita efectivamente la recursión pura de Python a lo que es seguro para la pila de C.

En 3.11, cuando CPython detecta código de Python que llama a otra función de Python, configura un nuevo marco y «salta» al nuevo código dentro del nuevo marco. Esto evita llamar a la función de interpretación de C por completo.

Most Python function calls now consume no C stack space, speeding them up. In simple recursive functions like fibonacci or factorial, we observed a 1.7x speedup. This also means recursive functions can recurse significantly deeper (if the user increases the recursion limit with `sys.setrecursionlimit()`). We measured a 1-3% improvement in pyperformance.

(Aportado por Pablo Galindo y Mark Shannon en bpo-45256.)

PEP 659: Intérprete Adaptativo Especializado

PEP 659 is one of the key parts of the Faster CPython project. The general idea is that while Python is a dynamic language, most code has regions where objects and types rarely change. This concept is known as *type stability*.

At runtime, Python will try to look for common patterns and type stability in the executing code. Python will then replace the current operation with a more specialized one. This specialized operation uses fast paths available only to those use cases/types, which generally outperform their generic counterparts. This also brings in another concept called *inline caching*, where Python caches the results of expensive operations directly in the bytecode.

The specializer will also combine certain common instruction pairs into one superinstruction, reducing the overhead during execution.

Python will only specialize when it sees code that is «hot» (executed multiple times). This prevents Python from wasting time on run-once code. Python can also de-specialize when code is too dynamic or when the use changes. Specialization is attempted periodically, and specialization attempts are not too expensive, allowing specialization to adapt to new circumstances.

(PEP escrito por Mark Shannon, con ideas inspiradas por Stefan Brunthaler. Consulte **PEP 659** para obtener más información. Implementación por Mark Shannon y Brandt Bucher, con ayuda adicional de Irit Katriel y Dennis Sweeney).

Operación	Forma	Especialización	Aceleración de la operación (hasta)	Colaborador(es)

Operaciones binarias	`x + x`
	`x - x`
	`x * x`
		Binary add, multiply and subtract for common types such as `int`, `float` and `str` take custom fast paths for their underlying types.	10%	Mark Shannon, Dong-hee Na, Brandt Bucher, Dennis Sweeney

Subíndice	`a[i]`	Subscripting container types such as `list`, `tuple` and `dict` directly index the underlying data structures. Subscripting custom `__getitem__()` is also inlined similar to *Llamadas a funciones de Python en línea*.	10-25%	Irit KatrielMark Shannon
Almacenar subíndice	`a[i] = z`	Similar a la especialización de subíndices anterior.	10-25%	dennis sweeney
Llamadas	`f(arg)` `C(arg)`	Calls to common builtin (C) functions and types such as `len()` and `str` directly call their underlying C version. This avoids going through the internal calling convention.	20%	Mark Shannon-Ken Jin
Cargar variable global	`print` `len`	El índice del objeto en el espacio de nombres globales/integrados se almacena en caché. La carga de globales e integrados requiere cero búsquedas de espacios de nombres.	[1]	marca shannon
Cargar atributo	`o.attr`	Similar a cargar variables globales. El índice del atributo dentro del espacio de nombres de la clase/objeto se almacena en caché. En la mayoría de los casos, la carga de atributos requerirá cero búsquedas de espacios de nombres.	[2]	marca shannon
Cargar métodos para llamar	`o.meth()`	La dirección real del método se almacena en caché. La carga de métodos ahora no tiene búsquedas de espacio de nombres, incluso para clases con largas cadenas de herencia.	10-20%	Ken JinMark Shannon
Atributo de la tienda	`o.attr = z`	Similar a la optimización de atributos de carga.	2% en rendimiento	marca shannon
Secuencia de desempaquetado	`*seq`	Specialized for common containers such as `list` and `tuple`. Avoids internal calling convention.	8%	brandt bucher

[1] A similar optimization already existed since Python 3.8. 3.11 specializes for more forms and reduces some overhead.

[2] Ya existía una optimización similar desde Python 3.10. 3.11 se especializa en más formularios. Además, bpo-45947 debería acelerar todas las cargas de atributos.

9.3 Varios

- Los objetos ahora requieren menos memoria debido a los espacios de nombres de objetos creados con pereza. Sus diccionarios de espacio de nombres ahora también comparten claves más libremente. (Contribuido por Mark Shannon en bpo-45340 y bpo-40116).
- «Zero-cost» exceptions are implemented, eliminating the cost of `try` statements when no exception is raised. (Contributed by Mark Shannon in bpo-40222.)
- Una representación más concisa de las excepciones en el intérprete redujo el tiempo necesario para detectar una excepción en aproximadamente un 10 %. (Aportado por Irit Katriel en bpo-45711.)
- `re`'s regular expression matching engine has been partially refactored, and now uses computed gotos (or «threaded code») on supported platforms. As a result, Python 3.11 executes the pyperformance regular expression benchmarks up to 10% faster than Python 3.10. (Contributed by Brandt Bucher in gh-91404.)

9.4 Preguntas más frecuentes

How should I write my code to utilize these speedups?

Write Pythonic code that follows common best practices; you don't have to change your code. The Faster CPython project optimizes for common code patterns we observe.

Will CPython 3.11 use more memory?

Maybe not; we don't expect memory use to exceed 20% higher than 3.10. This is offset by memory optimizations for frame objects and object dictionaries as mentioned above.

I don't see any speedups in my workload. Why?

Certain code won't have noticeable benefits. If your code spends most of its time on I/O operations, or already does most of its computation in a C extension library like NumPy, there won't be significant speedups. This project currently benefits pure-Python workloads the most.

Además, las cifras de pyperformance son una media geométrica. Incluso dentro de los puntos de referencia de pyrendimiento, ciertos puntos de referencia se han ralentizado ligeramente, mientras que otros se han acelerado casi 2 veces.

Is there a JIT compiler?

No. We're still exploring other optimizations.

9.5 Sobre

Faster CPython explora optimizaciones para CPython. El equipo principal está financiado por Microsoft para trabajar en esto a tiempo completo. Pablo Galindo Salgado también está financiado por Bloomberg LP para trabajar en el proyecto a tiempo parcial. Finalmente, muchos contribuyentes son voluntarios de la comunidad.

10 Cambios en el código de bytes de CPython

El código de bytes ahora contiene entradas de caché en línea, que toman la forma de las instrucciones `CACHE` recién agregadas. Muchos códigos de operación esperan ser seguidos por una cantidad exacta de cachés e indican al intérprete que los omita en tiempo de ejecución. Los cachés poblados pueden parecer instrucciones arbitrarias, por lo que se debe tener mucho cuidado al leer o modificar el código de bytes adaptativo sin procesar que contiene datos acelerados.

10.1 Nuevos códigos de operación

- `ASYNC_GEN_WRAP`, `RETURN_GENERATOR` y `SEND`, utilizados en generadores y co-rutinas.
- `COPY_FREE_VARS`, que evita la necesidad de un código especial del lado de la persona que llama para los cierres.
- `JUMP_BACKWARD_NO_INTERRUPT`, para usar en ciertos bucles donde no es deseable manejar interrupciones.
- `MAKE_CELL`, para crear cell-objects.
- `CHECK_EG_MATCH` y `PREP_RERAISE_STAR`, para manejar el *new exception groups and except** agregado en **PEP 654**.
- `PUSH_EXC_INFO`, para uso en controladores de excepciones.
- `RESUME`, no operativo, para el seguimiento interno, la depuración y las comprobaciones de optimización.

10.2 Códigos de operación reemplazados

Códigos de operación reemplazados	Nuevos códigos de operación	notas
`BINARY_*` `INPLACE_*`	`BINARY_OP`	Reemplazó todos los códigos de operación numéricos binarios/en el lugar con un solo código de operación

`CALL_FUNCTION`
`CALL_FUNCTION_KW`
`CALL_METHOD`
	`CALL`
	`KW_NAMES`
	`PRECALL`
	`PUSH_NULL`
		Separa el cambio de argumentos para métodos del manejo de argumentos de palabras clave; permite una mejor especialización de las llamadas

`DUP_TOP`
`DUP_TOP_TWO`
`ROT_TWO`
`ROT_THREE`
`ROT_FOUR`
`ROT_N`
	`COPY`
	`SWAP`
		Instrucciones de manipulación de pilas

`JUMP_IF_NOT_EXC_MATCH`	`CHECK_EXC_MATCH`	Ahora realiza la comprobación pero no salta.

`JUMP_ABSOLUTE`
`POP_JUMP_IF_FALSE`
`POP_JUMP_IF_TRUE`
	`JUMP_BACKWARD`
	`POP_JUMP_BACKWARD_IF_*`
	`POP_JUMP_FORWARD_IF_*`
		Ver[3]; Variantes `TRUE`, `FALSE`, `NONE` y `NOT_NONE` para cada dirección

`SETUP_WITH` `SETUP_ASYNC_WITH`	`BEFORE_WITH`	Configuración del bloque `with`

[3] Todos los códigos de operación de salto ahora son relativos, incluidos los `JUMP_IF_TRUE_OR_POP` y `JUMP_IF_FALSE_OR_POP` existentes. El argumento ahora es un desplazamiento de la instrucción actual en lugar de una ubicación absoluta.

10.3 Códigos de operación cambiados/eliminados

- Se cambiaron `MATCH_CLASS` y `MATCH_KEYS` para que ya no envíen un valor booleano adicional para indicar éxito/fracaso. En su lugar, se inserta `None` en caso de error en lugar de la tupla de valores extraídos.
- Se cambiaron los códigos de operación que funcionan con excepciones para reflejarlas y ahora se representan como un elemento en la pila en lugar de tres (ver gh-89874).
- Se eliminaron `COPY_DICT_WITHOUT_KEYS`, `GEN_START`, `POP_BLOCK`, `SETUP_FINALLY` y `YIELD_FROM`.

11 Obsoleto

Esta sección enumera las API de Python que han quedado obsoletas en Python 3.11.

Las API de C en desuso son *listed separately*.

11.1 Idioma/Construidos

- El encadenamiento de descriptores `classmethod` (introducido en bpo-19072) ahora está en desuso. Ya no se puede usar para envolver otros descriptores como `property`. El diseño central de esta función tenía fallas y causó una serie de problemas posteriores. Para «transmitir» un `classmethod`, considere usar el atributo `__wrapped__` que se agregó en Python 3.10. (Aportado por Raymond Hettinger en gh-89519.)
- Los escapes octales en cadenas y bytes literales con valores mayores que `0o377` (255 en decimal) ahora producen un `DeprecationWarning`. En una futura versión de Python, generarán un `SyntaxWarning` y eventualmente un `SyntaxError`. (Aportado por Serhiy Storchaka en gh-81548.)
- La delegación de `int()` a `__trunc__()` ahora está obsoleta. Llamar a `int(a)` cuando `type(a)` implementa `__trunc__()` pero no `__int__()` o `__index__()` ahora genera un `DeprecationWarning`. (Aportado por Zackery Spytz en bpo-44977).

11.2 Módulos

- **PEP 594** condujo a la desaprobación de los siguientes módulos programados para su eliminación en Python 3.13:

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

  (Aportado por Brett Cannon en bpo-47061 y Victor Stinner en gh-68966).
- The `asynchat`, `asyncore` and `smtpd` modules have been deprecated since at least Python 3.6. Their documentation and deprecation warnings have now been updated to note they will be removed in Python 3.12. (Contributed by Hugo van Kemenade in bpo-47022.)
- El paquete `lib2to3` y la herramienta 2to3 ahora están obsoletos y es posible que no puedan analizar Python 3.10 o posterior. Consulte **PEP 617**, que presenta el nuevo analizador PEG, para obtener más información. (Aportado por Victor Stinner en bpo-40360.)
- Los módulos no documentados `sre_compile`, `sre_constants` y `sre_parse` ahora están obsoletos. (Aportado por Serhiy Storchaka en bpo-47152.)

11.3 Biblioteca estándar

- The following have been deprecated in `configparser` since Python 3.2. Their deprecation warnings have now been updated to note they will be removed in Python 3.12:
  - la clase `configparser.SafeConfigParser`
  - la propiedad `configparser.ParsingError.filename`
  - el método `configparser.RawConfigParser.readfp()`

  (Aportado por Hugo van Kemenade en bpo-45173.)
- `configparser.LegacyInterpolation` ha quedado obsoleto en la cadena de documentación desde Python 3.2 y no aparece en la documentación de `configparser`. Ahora emite un `DeprecationWarning` y se eliminará en Python 3.13. Utilice `configparser.BasicInterpolation` o `configparser.ExtendedInterpolation` en su lugar. (Aportado por Hugo van Kemenade en bpo-46607.)
- El conjunto anterior de funciones `importlib.resources` quedó en desuso en favor de los reemplazos agregados en Python 3.9 y se eliminará en una versión futura de Python, debido a que no admite recursos ubicados dentro de los subdirectorios del paquete:
  - `importlib.resources.contents()`
  - `importlib.resources.is_resource()`
  - `importlib.resources.open_binary()`
  - `importlib.resources.open_text()`
  - `importlib.resources.read_binary()`
  - `importlib.resources.read_text()`
  - `importlib.resources.path()`
- La función `locale.getdefaultlocale()` está en desuso y se eliminará en Python 3.13. Utilice las funciones `locale.setlocale()`, `locale.getpreferredencoding(False)` y `locale.getlocale()` en su lugar. (Aportado por Victor Stinner en gh-90817.)
- La función `locale.resetlocale()` está en desuso y se eliminará en Python 3.13. Utilice `locale.setlocale(locale.LC_ALL, "")` en su lugar. (Aportado por Victor Stinner en gh-90817.)
- Ahora se aplicarán reglas más estrictas para las referencias de grupos numéricos y los nombres de grupos en regular expressions. Ahora solo se aceptarán secuencias de dígitos ASCII como referencia numérica, y el nombre del grupo en los patrones `bytes` y las cadenas de reemplazo solo pueden contener letras ASCII, dígitos y guiones bajos. Por ahora, se genera una advertencia de desaprobación para la sintaxis que viola estas reglas. (Aportado por Serhiy Storchaka en gh-91760.)
- En el módulo `re`, la función `re.template()` y los indicadores `re.TEMPLATE` y `re.T` correspondientes están obsoletos, ya que no estaban documentados y carecían de un propósito obvio. Se eliminarán en Python 3.13. (Aportado por Serhiy Storchaka y Miro Hrončok en gh-92728.)
- `turtle.settiltangle()` está en desuso desde Python 3.1; ahora emite una advertencia de desaprobación y se eliminará en Python 3.13. Utilice `turtle.tiltangle()` en su lugar (anteriormente se marcó incorrectamente como obsoleto y su cadena de documentación ahora está corregida). (Aportado por Hugo van Kemenade en bpo-45837.)
- `typing.Text`, que existe únicamente para proporcionar soporte de compatibilidad entre el código de Python 2 y Python 3, ahora está obsoleto. Su eliminación no está planificada actualmente, pero se recomienda a los usuarios que usen `str` en su lugar siempre que sea posible. (Aportado por Alex Waygood en gh-92332.)
- La sintaxis de argumento de palabra clave para construir tipos `typing.TypedDict` ahora está obsoleta. Se eliminará el soporte en Python 3.13. (Aportado por Jingchen Ye en gh-90224.)

- `webbrowser.MacOSX` está en desuso y se eliminará en Python 3.13. No está probado, no está documentado y no es utilizado por el propio `webbrowser`. (Aportado por Dong-hee Na en bpo-42255.)
- El comportamiento de devolver un valor de los métodos de prueba `TestCase` y `IsolatedAsyncioTestCase` (aparte del valor `None` predeterminado) ahora está obsoleto.
- Se descartaron las siguientes funciones `unittest` no documentadas formalmente, programadas para su eliminación en Python 3.13:
  - `unittest.findTestCases()`
  - `unittest.makeSuite()`
  - `unittest.getTestCaseNames()`

  Utilice métodos `TestLoader` en su lugar:
  - `unittest.TestLoader.loadTestsFromModule()`
  - `unittest.TestLoader.loadTestsFromTestCase()`
  - `unittest.TestLoader.getTestCaseNames()`

  (Aportado por Erlend E. Aasland en bpo-5846.)

12 Eliminación pendiente en Python 3.12

Las siguientes API de Python han quedado obsoletas en versiones anteriores de Python y se eliminarán en Python 3.12.

Las API de C pendientes de eliminación son *listed separately*.

- The `asynchat` module
- The `asyncore` module
- The entire distutils package
- The `imp` module
- The `typing.io` namespace
- The `typing.re` namespace
- `cgi.log()`
- `importlib.find_loader()`
- `importlib.abc.Loader.module_repr()`
- `importlib.abc.MetaPathFinder.find_module()`
- `importlib.abc.PathEntryFinder.find_loader()`
- `importlib.abc.PathEntryFinder.find_module()`
- `importlib.machinery.BuiltinImporter.find_module()`
- `importlib.machinery.BuiltinLoader.module_repr()`
- `importlib.machinery.FileFinder.find_loader()`
- `importlib.machinery.FileFinder.find_module()`
- `importlib.machinery.FrozenImporter.find_module()`
- `importlib.machinery.FrozenLoader.module_repr()`

- `importlib.machinery.PathFinder.find_module()`
- `importlib.machinery.WindowsRegistryFinder.find_module()`
- `importlib.util.module_for_loader()`
- `importlib.util.set_loader_wrapper()`
- `importlib.util.set_package_wrapper()`
- `pkgutil.ImpImporter`
- `pkgutil.ImpLoader`
- `pathlib.Path.link_to()`
- `sqlite3.enable_shared_cache()`
- `sqlite3.OptimizedUnicode()`
- `PYTHONTHREADDEBUG` environment variable
- The following deprecated aliases in `unittest`:

Deprecated alias	Method Name	Deprecated in
`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

13 Remoto

This section lists Python APIs that have been removed in Python 3.11.

Las API C eliminadas son *listed separately*.

- Se eliminó `@asyncio.coroutine()` decorator, lo que permite que las corrutinas basadas en generadores heredados sean compatibles con el código `async`/`await`. La función ha quedado obsoleta desde Python 3.8 y la eliminación se programó inicialmente para Python 3.10. Utilice `async def` en su lugar. (Aportado por Illia Volochii en bpo-43216.)
- Se eliminó `asyncio.coroutines.CoroWrapper` utilizado para envolver objetos de corrutina basados en generadores heredados en el modo de depuración. (Aportado por Illia Volochii en bpo-43216.)
- Debido a importantes problemas de seguridad, el parámetro *reuse_address* de `asyncio.loop.create_datagram_endpoint()`, deshabilitado en Python 3.9, ahora se eliminó por completo. Esto se debe al comportamiento de la opción de socket `SO_REUSEADDR` en UDP. (Aportado por Hugo van Kemenade en bpo-45129.)

- Se eliminó el módulo `binhex`, obsoleto en Python 3.9. También se eliminaron las funciones `binascii` relacionadas y obsoletas de manera similar:
  - `binascii.a2b_hqx()`
  - `binascii.b2a_hqx()`
  - `binascii.rlecode_hqx()`
  - `binascii.rldecode_hqx()`

  La función `binascii.crc_hqx()` permanece disponible.

  (Aportado por Victor Stinner en bpo-45085.)
- Se eliminó el comando `distutils bdist_msi` en desuso en Python 3.9. Utilice `bdist_wheel` (paquetes de ruedas) en su lugar. (Aportado por Hugo van Kemenade en bpo-45124.)
- Se eliminaron los métodos `__getitem__()` de `xml.dom.pulldom.DOMEventStream`, `wsgiref.util.FileWrapper` y `fileinput.FileInput`, obsoletos desde Python 3.9. (Aportado por Hugo van Kemenade en bpo-45132.)
- Se eliminaron las funciones obsoletas `gettext lgettext()`, `ldgettext()`, `lngettext()` y `ldngettext()`. También se eliminó la función `bind_textdomain_codeset()`, los métodos `NullTranslations.output_charset()` y `NullTranslations.set_output_charset()`, y el parámetro *codeset* de `translation()` y `install()`, ya que solo se usan para las funciones `l*gettext()`. (Aportado por Dong-hee Na y Serhiy Storchaka en bpo-44235).
- Eliminado del módulo `inspect`:
  - La función `getargspec()`, en desuso desde Python 3.0; utilice `inspect.signature()` o `inspect.getfullargspec()` en su lugar.
  - La función `formatargspec()`, obsoleta desde Python 3.5; use la función `inspect.signature()` o el objeto `inspect.Signature` directamente.
  - Los métodos `Signature.from_builtin()` y `Signature.from_function()` no documentados, obsoletos desde Python 3.5; utilice el método `Signature.from_callable()` en su lugar.

  (Aportado por Hugo van Kemenade en bpo-45320.)
- Se eliminó el método `__class_getitem__()` de `pathlib.PurePath`, porque no se usó y se agregó por error en versiones anteriores. (Aportado por Nikita Sobolev en bpo-46483.)
- Se eliminó la clase `MailmanProxy` en el módulo `smtpd`, ya que no se puede usar sin el paquete `mailman` externo. (Aportado por Dong-hee Na en bpo-35800.)
- Se eliminó el método obsoleto `split()` de `_tkinter.TkappType`. (Aportado por Erlend E. Aasland en bpo-38371.)
- Se eliminó la compatibilidad con el paquete de espacio de nombres del descubrimiento `unittest`. Se introdujo en Python 3.4 pero se rompió desde Python 3.7. (Aportado por Inada Naoki en bpo-23882.)
- Se eliminó el método `float.__set_format__()` privado no documentado, anteriormente conocido como `float.__setformat__()` en Python 3.7. Su cadena de documentación decía: «Probablemente no desee utilizar esta función. Existe principalmente para ser utilizada en el conjunto de pruebas de Python». (Aportado por Victor Stinner en bpo-46852.)
- El indicador de configuración `--experimental-isolated-subinterpreters` (y la macro `EXPERIMENTAL_ISOLATED_SUBINTERPRETERS` correspondiente) se han eliminado.
- Pynche — El editor de tonos y colores naturales de Python — se ha sacado de `Tools/scripts` y es being developed independently del árbol de fuentes de Python.

14 Migración a Python 3.11

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

Las notas de portabilidad para la API de C son *listed separately*.

- `open()`, `io.open()`, `codecs.open()` y `fileinput.FileInput` ya no aceptan `'U'` («nueva línea universal») en el modo de archivo. En Python 3, el modo «nueva línea universal» se usa de forma predeterminada cada vez que se abre un archivo en modo de texto, y el indicador `'U'` ha quedado obsoleto desde Python 3.3. El newline parameter para estas funciones controla cómo funcionan las nuevas líneas universales. (Aportado por Victor Stinner en bpo-37330.)
- Las posiciones de los nodos `ast.AST` ahora se validan cuando se proporcionan a `compile()` y otras funciones relacionadas. Si se detectan posiciones no válidas, se generará un `ValueError`. (Aportado por Pablo Galindo en gh-93351)
- Prohibido pasar ejecutores que no sean `concurrent.futures.ThreadPoolExecutor` a `asyncio.loop.set_default_executor()` luego de una obsolescencia en Python 3.8. (Aportado por Illia Volochii en bpo-43234.)
- `calendar`: las clases `calendar.LocaleTextCalendar` y `calendar.LocaleHTMLCalendar` ahora usan `locale.getlocale()`, en lugar de usar `locale.getdefaultlocale()`, si no se especifica una configuración regional. (Aportado por Victor Stinner en bpo-46659.)
- El módulo `pdb` ahora lee el archivo de configuración `.pdbrc` con la codificación `'UTF-8'`. (Contribuido por Srinivas Reddy Thatiparthy (□□□□□□□□□□□□ □□□□□□□ □□□□□□□□□□□) en bpo-41137.)
- El parámetro *population* de `random.sample()` debe ser una secuencia y ya no se admite la conversión automática de `sets` a `lists`. Además, si el tamaño de la muestra es mayor que el tamaño de la población, se genera un `ValueError`. (Aportado por Raymond Hettinger en bpo-40465.)
- Se eliminó el parámetro opcional *random* de `random.shuffle()`. Anteriormente, era una función aleatoria arbitraria para usar en la reproducción aleatoria; ahora, siempre se utilizará `random.random()` (su valor predeterminado anterior).
- En `re` re-syntax, las banderas en línea globales (por ejemplo, `(?i)`) ahora solo se pueden usar al comienzo de las expresiones regulares. Su uso en otros lugares ha quedado obsoleto desde Python 3.6. (Aportado por Serhiy Storchaka en bpo-47066.)
- En el módulo `re`, se corrigieron varios errores antiguos que, en casos excepcionales, podían hacer que los grupos de captura obtuvieran un resultado incorrecto. Por lo tanto, esto podría cambiar la salida capturada en estos casos. (Contribuido por Ma Lin en bpo-35859.)

15 Construir cambios

- CPython ahora tiene **PEP 11 Tier 3 support** para la compilación cruzada en las plataformas WebAssembly Emscripten (`wasm32-unknown-emscripten`, es decir, Python en el navegador) y WebAssembly System Interface (WASI) (`wasm32-unknown-wasi`). El esfuerzo está inspirado en trabajos anteriores como Pyodide. Estas plataformas proporcionan un subconjunto limitado de API POSIX; Las funciones y módulos de las bibliotecas estándar de Python relacionadas con redes, procesos, subprocesos, señales, mmap y usuarios/grupos no están disponibles o no funcionan. (Escritos aportados por Christian Heimes y Ethan Smith en gh-84461 y WASI aportados por Christian Heimes en gh-90473; plataformas promocionadas en gh-95085)
- Building CPython now requires:

  - A C11 compiler and standard library. Optional C11 features are not required. (Contributed by Victor Stinner in bpo-46656, bpo-45440 and bpo-46640.)
  - Compatibilidad con números de punto flotante IEEE 754. (Aportado por Victor Stinner en bpo-46917.)

- The `Py_NO_NAN` macro has been removed. Since CPython now requires IEEE 754 floats, NaN values are always available. (Contributed by Victor Stinner in bpo-46656.)
- El paquete `tkinter` ahora requiere Tcl/Tk versión 8.5.12 o posterior. (Aportado por Serhiy Storchaka en bpo-46996.)
- Las dependencias de compilación, los indicadores del compilador y los indicadores del vinculador para la mayoría de los módulos de extensión stdlib ahora son detectados por **`configure`**. pkg-config detecta los indicadores libffi, libnsl, libsqlite3, zlib, bzip2, liblzma, libcrypt, Tcl/Tk y uuid (si están disponibles). `tkinter` ahora requiere un comando pkg-config para detectar configuraciones de desarrollo para encabezados y bibliotecas Tcl/Tk. (Aportado por Christian Heimes y Erlend Egeberg Aasland en bpo-45847, bpo-45747 y bpo-45763).
- libpython ya no está vinculado con libcrypt. (Aportado por Mike Gilbert en bpo-45433.)
- CPython ahora se puede compilar con la opción ThinLTO pasando `thin` a `--with-lto`, es decir, `--with-lto=thin`. (Aportado por Dong-hee Na y Brett Holman en bpo-44340).
- Las listas libres para estructuras de objetos ahora se pueden deshabilitar. Se puede usar una nueva opción **`configure`** `--without-freelists` para deshabilitar todas las listas libres excepto el singleton de tupla vacío. (Aportado por Christian Heimes en bpo-45522.)
- `Modules/Setup` y `Modules/makesetup` se han mejorado y atado. Los módulos de extensión ahora se pueden construir a través de `makesetup`. Todos, excepto algunos módulos de prueba, se pueden vincular estáticamente a una biblioteca o binario principal. (Aportado por Brett Cannon y Christian Heimes en bpo-45548, bpo-45570, bpo-45571 y bpo-43974).

Nota: Utilice las variables de entorno `TCLTK_CFLAGS` y `TCLTK_LIBS` para especificar manualmente la ubicación de los encabezados y bibliotecas Tcl/Tk. Se han eliminado las opciones **`configure`** `--with-tcltk-includes` y `--with-tcltk-libs`.

En RHEL 7 y CentOS 7, los paquetes de desarrollo no proporcionan `tcl.pc` y `tk.pc`; usa `TCLTK_LIBS="-ltk8.5 -ltkstub8.5 -ltcl8.5"`. El directorio `Misc/rhel7` contiene archivos `.pc` e instrucciones sobre cómo compilar Python con Tcl/Tk y OpenSSL de RHEL 7 y CentOS 7.

- CPython ahora usará dígitos de 30 bits de manera predeterminada para la implementación de Python `int`. Anteriormente, el valor predeterminado era usar dígitos de 30 bits en plataformas con `SIZEOF_VOID_P >= 8` y dígitos de 15 bits en caso contrario. Todavía es posible solicitar explícitamente el uso de dígitos de 15 bits a través de la opción `--enable-big-digits` en el script de configuración o (para Windows) la variable `PYLONG_BITS_IN_DIGIT` en `PC/pyconfig.h`, pero es posible que esta opción se elimine en algún momento en el futuro. (Contribuido por Mark Dickinson en bpo-45569.)

16 Cambios en la API de C

16.1 Nuevas características

- Agregue una nueva función `PyType_GetName()` para obtener el nombre corto del tipo. (Aportado por Hai Shi en bpo-42035.)
- Agregue una nueva función `PyType_GetQualName()` para obtener el nombre completo del tipo. (Aportado por Hai Shi en bpo-42035.)

- Agregue nuevas funciones `PyThreadState_EnterTracing()` y `PyThreadState_LeaveTracing()` a la API de C limitada para suspender y reanudar el seguimiento y la creación de perfiles. (Aportado por Victor Stinner en bpo-43760.)
- Se agregó la constante `Py_Version` que tiene el mismo valor que `PY_VERSION_HEX`. (Aportado por Gabriele N. Tornetta en bpo-43931.)
- `Py_buffer` y las API ahora forman parte de la API limitada y la ABI estable:
  - `PyObject_CheckBuffer()`
  - `PyObject_GetBuffer()`
  - `PyBuffer_GetPointer()`
  - `PyBuffer_SizeFromFormat()`
  - `PyBuffer_ToContiguous()`
  - `PyBuffer_FromContiguous()`
  - `PyBuffer_CopyData()`
  - `PyBuffer_IsContiguous()`
  - `PyBuffer_FillContiguousStrides()`
  - `PyBuffer_FillInfo()`
  - `PyBuffer_Release()`
  - `PyMemoryView_FromBuffer()`
  - Ranuras de tipo `bf_getbuffer` y `bf_releasebuffer`

  (Aportado por Christian Heimes en bpo-45459.)
- Se agregó la función `PyType_GetModuleByDef`, utilizada para obtener el módulo en el que se definió un método, en los casos en que esta información no esté disponible directamente (a través de `PyCMethod`). (Aportado por Petr Viktorin en bpo-46613.)
- Agrega nuevas funciones para empaquetar y desempaquetar C doble (serializar y deserializar): `PyFloat_Pack2()`, `PyFloat_Pack4()`, `PyFloat_Pack8()`, `PyFloat_Unpack2()`, `PyFloat_Unpack4()` y `PyFloat_Unpack8()`. (Aportado por Victor Stinner en bpo-46906.)
- Agregue nuevas funciones para obtener atributos de objetos de marco: `PyFrame_GetBuiltins()`, `PyFrame_GetGenerator()`, `PyFrame_GetGlobals()`, `PyFrame_GetLasti()`.
- Se agregaron dos funciones nuevas para obtener y configurar la instancia de excepción activa: `PyErr_GetHandledException()` y `PyErr_SetHandledException()`. Estas son alternativas a `PyErr_SetExcInfo()` y `PyErr_GetExcInfo()` que funcionan con la representación de excepciones heredada de 3 tuplas. (Aportado por Irit Katriel en bpo-46343.)
- Se agregó el miembro `PyConfig.safe_path`. (Aportado por Victor Stinner en gh-57684.)

16.2 Migración a Python 3.11

- Algunas macros se han convertido en funciones estáticas en línea para evitar macro pitfalls. El cambio debería ser en su mayoría transparente para los usuarios, ya que las funciones de reemplazo emitirán sus argumentos a los tipos esperados para evitar advertencias del compilador debido a comprobaciones de tipos estáticos. Sin embargo, cuando la API de C limitada se establece en >=3.11, estas conversiones no se realizan y las personas que llaman deberán convertir argumentos a sus tipos esperados. Ver **PEP 670** para más detalles. (Aportado por Victor Stinner y Erlend E. Aasland en gh-89653.)

- `PyErr_SetExcInfo()` ya no usa los argumentos `type` y `traceback`, el intérprete ahora deriva esos valores de la instancia de excepción (el argumento `value`). La función aún roba referencias de los tres argumentos. (Aportado por Irit Katriel en bpo-45711.)

- `PyErr_GetExcInfo()` ahora deriva los campos `type` y `traceback` del resultado de la instancia de excepción (el campo `value`). (Aportado por Irit Katriel en bpo-45711.)

- `_frozen` tiene un nuevo campo `is_package` para indicar si el módulo congelado es o no un paquete. Anteriormente, un valor negativo en el campo `size` era el indicador. Ahora solo se pueden usar valores no negativos para `size`. (Aportado por Kumar Aditya en bpo-46608.)

- `_PyFrameEvalFunction()` ahora toma `_PyInterpreterFrame*` como su segundo parámetro, en lugar de `PyFrameObject*`. Consulte **PEP 523** para obtener más detalles sobre cómo usar este tipo de puntero de función.

- `PyCode_New()` y `PyCode_NewWithPosOnlyArgs()` ahora toman un argumento `exception_table` adicional. Debe evitarse el uso de estas funciones, en la medida de lo posible. Para obtener un objeto de código personalizado: cree un objeto de código usando el compilador, luego obtenga una versión modificada con el método `replace`.

- `PyCodeObject` ya no tiene los campos `co_code`, `co_varnames`, `co_cellvars` y `co_freevars`. En su lugar, utilice `PyCode_GetCode()`, `PyCode_GetVarnames()`, `PyCode_GetCellvars()` y `PyCode_GetFreevars()` respectivamente para acceder a ellos a través de la API de C. (Aportado por Brandt Bucher en bpo-46841 y Ken Jin en gh-92154 y gh-94936).

- Las antiguas macros de papelera (`Py_TRASHCAN_SAFE_BEGIN`/`Py_TRASHCAN_SAFE_END`) ahora están obsoletas. Deberían ser reemplazadas por las nuevas macros `Py_TRASHCAN_BEGIN` y `Py_TRASHCAN_END`.

  Una función tp_dealloc que tiene las macros antiguas, como:

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_SAFE_BEGIN(p);
    ...
    Py_TRASHCAN_SAFE_END
}
```

  debe migrar a las nuevas macros de la siguiente manera:

```
static void
mytype_dealloc(mytype *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, mytype_dealloc)
    ...
    Py_TRASHCAN_END
}
```

Tenga en cuenta que `Py_TRASHCAN_BEGIN` tiene un segundo argumento que debería ser la función de desasignación en la que se encuentra.

Para admitir versiones anteriores de Python en la misma base de código, puede definir las siguientes macros y usarlas en todo el código (crédito: se copiaron de la base de código `mypy`):

```
#if PY_VERSION_HEX >= 0x03080000
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_BEGIN(op, dealloc)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_END
#else
#  define CPy_TRASHCAN_BEGIN(op, dealloc) Py_TRASHCAN_SAFE_BEGIN(op)
#  define CPy_TRASHCAN_END(op) Py_TRASHCAN_SAFE_END(op)
#endif
```

- La función `PyType_Ready()` ahora genera un error si un tipo se define con el indicador `Py_TPFLAGS_HAVE_GC` establecido pero no tiene función transversal (`PyTypeObject.tp_traverse`). (Aportado por Victor Stinner en bpo-44263.)
- Los tipos de almacenamiento dinámico con el indicador `Py_TPFLAGS_IMMUTABLETYPE` ahora pueden heredar el protocolo vectorcall **PEP 590**. Anteriormente, esto solo era posible para static types. (Aportado por Erlend E. Aasland en bpo-43908)
- Dado que `Py_TYPE()` se cambia a una función estática en línea, `Py_TYPE(obj) = new_type` debe reemplazarse por `Py_SET_TYPE(obj, new_type)`: consulte la función `Py_SET_TYPE()` (disponible desde Python 3.9). Para compatibilidad con versiones anteriores, se puede usar esta macro:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_TYPE)
static inline void _Py_SET_TYPE(PyObject *ob, PyTypeObject *type)
{ ob->ob_type = type; }
#define Py_SET_TYPE(ob, type) _Py_SET_TYPE((PyObject*)(ob), type)
#endif
```

(Aportado por Victor Stinner en bpo-39573.)

- Dado que `Py_SIZE()` se cambia a una función estática en línea, `Py_SIZE(obj) = new_size` debe reemplazarse por `Py_SET_SIZE(obj, new_size)`: consulte la función `Py_SET_SIZE()` (disponible desde Python 3.9). Para compatibilidad con versiones anteriores, se puede usar esta macro:

```
#if PY_VERSION_HEX < 0x030900A4 && !defined(Py_SET_SIZE)
static inline void _Py_SET_SIZE(PyVarObject *ob, Py_ssize_t size)
{ ob->ob_size = size; }
#define Py_SET_SIZE(ob, size) _Py_SET_SIZE((PyVarObject*)(ob), size)
#endif
```

(Aportado por Victor Stinner en bpo-39573.)

- `<Python.h>` ya no incluye los archivos de encabezado `<stdlib.h>`, `<stdio.h>`, `<errno.h>` y `<string.h>` cuando la macro `Py_LIMITED_API` se establece en `0x030b0000` (Python 3.11) o superior. Las extensiones de C deben incluir explícitamente los archivos de encabezado después de `#include <Python.h>`. (Aportado por Victor Stinner en bpo-45434.)
- Los archivos de API no limitados `cellobject.h`, `classobject.h`, `code.h`, `context.h`, `funcobject.h`, `genobject.h` y `longintrepr.h` se han movido al directorio `Include/cpython`. Además, se eliminó el archivo de encabezado `eval.h`. Estos archivos no deben incluirse directamente, ya que ya están incluidos en `Python.h`: Include Files. Si se han incluido directamente, considere incluir `Python.h` en su lugar. (Aportado por Victor Stinner en bpo-35134.)
- La macro `PyUnicode_CHECK_INTERNED()` se ha excluido de la API de C limitada. Nunca se pudo usar allí, porque usaba estructuras internas que no están disponibles en la API de C limitada. (Aportado por Victor Stinner en bpo-46007.)

- Las siguientes funciones y tipos de cuadros ahora están disponibles directamente con `#include <Python.h>`, ya no es necesario agregar `#include <frameobject.h>`:
  - `PyFrame_Check()`
  - `PyFrame_GetBack()`
  - `PyFrame_GetBuiltins()`
  - `PyFrame_GetGenerator()`
  - `PyFrame_GetGlobals()`
  - `PyFrame_GetLasti()`
  - `PyFrame_GetLocals()`
  - `PyFrame_Type`

  (Aportado por Victor Stinner en gh-93937.)

- Los miembros de la estructura `PyFrameObject` se han eliminado de la API de C pública.

  Si bien la documentación señala que los campos `PyFrameObject` están sujetos a cambios en cualquier momento, se han mantenido estables durante mucho tiempo y se usaron en varias extensiones populares.

  En Python 3.11, la estructura del marco se reorganizó para permitir optimizaciones de rendimiento. Algunos campos se eliminaron por completo, ya que eran detalles de la implementación anterior.

  Campos `PyFrameObject`:

  - `f_back`: usa `PyFrame_GetBack()`.
  - `f_blockstack`: eliminado.
  - `f_builtins`: usa `PyFrame_GetBuiltins()`.
  - `f_code`: usa `PyFrame_GetCode()`.
  - `f_gen`: usa `PyFrame_GetGenerator()`.
  - `f_globals`: usa `PyFrame_GetGlobals()`.
  - `f_iblock`: eliminado.
  - `f_lasti`: usa `PyFrame_GetLasti()`. El código que usa `f_lasti` con `PyCode_Addr2Line()` debería usar `PyFrame_GetLineNumber()` en su lugar; puede ser más rápido.
  - `f_lineno`: usar `PyFrame_GetLineNumber()`
  - `f_locals`: usa `PyFrame_GetLocals()`.
  - `f_stackdepth`: eliminado.
  - `f_state`: sin API pública (renombrado como `f_frame.f_state`).
  - `f_trace`: sin API pública.
  - `f_trace_lines`: utiliza `PyObject_GetAttrString((PyObject*)frame, "f_trace_lines")`.
  - `f_trace_opcodes`: utiliza `PyObject_GetAttrString((PyObject*)frame, "f_trace_opcodes")`.
  - `f_localsplus`: sin API pública (renombrado como `f_frame.localsplus`).
  - `f_valuestack`: eliminado.

El objeto marco de Python ahora se crea de forma perezosa. Un efecto secundario es que no se debe acceder directamente al miembro `f_back`, ya que su valor ahora también se calcula de forma diferida. En su lugar, se debe llamar a la función `PyFrame_GetBack()`.

Los depuradores que accedieron a `f_locals` directamente *must* llaman a `PyFrame_GetLocals()` en su lugar. Ya no necesitan llamar a `PyFrame_FastToLocalsWithError()` o `PyFrame_LocalsToFast()`, de hecho, no deberían llamar a esas funciones. La actualización necesaria del marco ahora es administrada por la máquina virtual.

Código que define `PyFrame_GetCode()` en Python 3.8 y anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyCodeObject* PyFrame_GetCode(PyFrameObject *frame)
{
    Py_INCREF(frame->f_code);
    return frame->f_code;
}
#endif
```

Código que define `PyFrame_GetBack()` en Python 3.8 y anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyFrame_GetBack(PyFrameObject *frame)
{
    Py_XINCREF(frame->f_back);
    return frame->f_back;
}
#endif
```

O use pythoncapi_compat project para obtener estas dos funciones en versiones anteriores de Python.

- Cambios de los miembros de la estructura `PyThreadState`:
  - `frame`: eliminado, use `PyThreadState_GetFrame()` (función agregada a Python 3.9 por bpo-40429). Advertencia: la función devuelve un strong reference, necesita llamar a `Py_XDECREF()`.
  - `tracing`: cambiado, use `PyThreadState_EnterTracing()` y `PyThreadState_LeaveTracing()` (funciones agregadas a Python 3.11 por bpo-43760).
  - `recursion_depth`: eliminado, use `(tstate->recursion_limit - tstate->recursion_remaining)` en su lugar.
  - `stackcheck_counter`: eliminado.

Código que define `PyThreadState_GetFrame()` en Python 3.8 y anteriores:

```
#if PY_VERSION_HEX < 0x030900B1
static inline PyFrameObject* PyThreadState_GetFrame(PyThreadState *tstate)
{
    Py_XINCREF(tstate->frame);
    return tstate->frame;
}
#endif
```

Código que define `PyThreadState_EnterTracing()` y `PyThreadState_LeaveTracing()` en Python 3.10 y versiones anteriores:

```
#if PY_VERSION_HEX < 0x030B00A2
static inline void PyThreadState_EnterTracing(PyThreadState *tstate)
{
```

(continué en la próxima página)

(proviene de la página anterior)

```
    tstate->tracing++;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = 0;
#else
    tstate->use_tracing = 0;
#endif
}

static inline void PyThreadState_LeaveTracing(PyThreadState *tstate)
{
    int use_tracing = (tstate->c_tracefunc != NULL || tstate->c_profilefunc !=␣
↪NULL);
    tstate->tracing--;
#if PY_VERSION_HEX >= 0x030A00A1
    tstate->cframe->use_tracing = use_tracing;
#else
    tstate->use_tracing = use_tracing;
#endif
}
#endif
```

O use the pythoncapi_compat project para obtener estas funciones en funciones antiguas de Python.

- Se alienta a los distribuidores a compilar Python con la biblioteca Blake2 optimizada libb2.
- El campo `PyConfig.module_search_paths_set` ahora debe establecerse en 1 para que la inicialización use `PyConfig.module_search_paths` para inicializar `sys.path`. De lo contrario, la inicialización volverá a calcular la ruta y reemplazará los valores agregados a `module_search_paths`.
- `PyConfig_Read()` ya no calcula la ruta de búsqueda inicial y no completará ningún valor en `PyConfig.module_search_paths`. Para calcular rutas predeterminadas y luego modificarlas, finalice la inicialización y use `PySys_GetObject()` para recuperar `sys.path` como un objeto de lista de Python y modificarlo directamente.

16.3 Obsoleto

- Deseche las siguientes funciones para configurar la inicialización de Python:
  - `PySys_AddWarnOptionUnicode()`
  - `PySys_AddWarnOption()`
  - `PySys_AddXOption()`
  - `PySys_HasWarnOptions()`
  - `PySys_SetArgvEx()`
  - `PySys_SetArgv()`
  - `PySys_SetPath()`
  - `Py_SetPath()`
  - `Py_SetProgramName()`
  - `Py_SetPythonHome()`
  - `Py_SetStandardStreamEncoding()`
  - `_Py_SetProgramFullPath()`

Utilice la nueva API `PyConfig` de Python Initialization Configuration en su lugar (**PEP 587**). (Aportado por Victor Stinner en gh-88279.)

- Deje obsoleto el miembro `ob_shash` de `PyBytesObject`. Utilice `PyObject_Hash()` en su lugar. (Aportado por Inada Naoki en bpo-46864.)

16.4 Eliminación pendiente en Python 3.12

Las siguientes API de C quedaron obsoletas en versiones anteriores de Python y se eliminarán en Python 3.12.

- `PyUnicode_AS_DATA()`
- `PyUnicode_AS_UNICODE()`
- `PyUnicode_AsUnicodeAndSize()`
- `PyUnicode_AsUnicode()`
- `PyUnicode_FromUnicode()`
- `PyUnicode_GET_DATA_SIZE()`
- `PyUnicode_GET_SIZE()`
- `PyUnicode_GetSize()`
- `PyUnicode_IS_COMPACT()`
- `PyUnicode_IS_READY()`
- `PyUnicode_READY()`
- `Py_UNICODE_WSTR_LENGTH()`
- `_PyUnicode_AsUnicode()`
- `PyUnicode_WCHAR_KIND`
- `PyUnicodeObject`
- `PyUnicode_InternImmortal()`

16.5 Remoto

- Se han eliminado `PyFrame_BlockSetup()` y `PyFrame_BlockPop()`. (Aportado por Mark Shannon en bpo-40222.)
- Quite las siguientes macros matemáticas usando la variable `errno`:
  - `Py_ADJUST_ERANGE1()`
  - `Py_ADJUST_ERANGE2()`
  - `Py_OVERFLOWED()`
  - `Py_SET_ERANGE_IF_OVERFLOW()`
  - `Py_SET_ERRNO_ON_MATH_ERROR()`

  (Aportado por Victor Stinner en bpo-45412.)
- Elimine las macros `Py_UNICODE_COPY()` y `Py_UNICODE_FILL()`, obsoletas desde Python 3.3. Utilice `PyUnicode_CopyCharacters()` o `memcpy()` (cadena `wchar_t*`) y las funciones `PyUnicode_Fill()` en su lugar. (Aportado por Victor Stinner en bpo-41123.)

- Elimine el archivo de encabezado `pystrhex.h`. Solo contiene funciones privadas. Las extensiones C solo deben incluir el archivo de encabezado principal `<Python.h>`. (Aportado por Victor Stinner en bpo-45434.)
- Quite la macro `Py_FORCE_DOUBLE()`. Fue utilizado por la macro `Py_IS_INFINITY()`. (Aportado por Victor Stinner en bpo-45440.)
- Los siguientes elementos ya no están disponibles cuando se define `Py_LIMITED_API`:
  - `PyMarshal_WriteLongToFile()`
  - `PyMarshal_WriteObjectToFile()`
  - `PyMarshal_ReadObjectFromString()`
  - `PyMarshal_WriteObjectToString()`
  - la macro `Py_MARSHAL_VERSION`

  Estos no son parte del limited API.

  (Aportado por Victor Stinner en bpo-45474.)
- Excluya `PyWeakref_GET_OBJECT()` de la API de C limitada. Nunca funcionó ya que la estructura `PyWeakReference` es opaca en la API de C limitada. (Aportado por Victor Stinner en bpo-35134.)
- Quite la macro `PyHeapType_GET_MEMBERS()`. Fue expuesto en la API pública de C por error, solo debe ser utilizado por Python internamente. Utilice el miembro `PyTypeObject.tp_members` en su lugar. (Aportado por Victor Stinner en bpo-40170.)
- Elimine la macro `HAVE_PY_SET_53BIT_PRECISION` (movida a la API de C interna). (Aportado por Victor Stinner en bpo-45412.)
- Elimine las API del codificador `Py_UNICODE`, ya que han quedado obsoletas desde Python 3.3, se usan poco y son ineficientes en relación con las alternativas recomendadas.

  Las funciones eliminadas son:
  - `PyUnicode_Encode()`
  - `PyUnicode_EncodeASCII()`
  - `PyUnicode_EncodeLatin1()`
  - `PyUnicode_EncodeUTF7()`
  - `PyUnicode_EncodeUTF8()`
  - `PyUnicode_EncodeUTF16()`
  - `PyUnicode_EncodeUTF32()`
  - `PyUnicode_EncodeUnicodeEscape()`
  - `PyUnicode_EncodeRawUnicodeEscape()`
  - `PyUnicode_EncodeCharmap()`
  - `PyUnicode_TranslateCharmap()`
  - `PyUnicode_EncodeDecimal()`
  - `PyUnicode_TransformDecimalToASCII()`

  Ver **PEP 624** para más detalles y **migration guidance**. (Aportado por Inada Naoki en bpo-44029.)

Índice

P

Python Enhancement Proposals
    PEP 11, 30
    PEP 11#tier-3, 30
    PEP 484, 5, 6
    PEP 484#annotating-instance-and-class-methods, 6
    PEP 514, 5
    PEP 515, 12
    PEP 523, 33
    PEP 552, 9
    PEP 563, 8
    PEP 587, 38
    PEP 590, 34
    PEP 594, 3, 25
    PEP 617, 25
    PEP 624, 3, 39
    PEP 624#alternative-apis, 39
    PEP 646, 6
    PEP 654, 5, 23
    PEP 655, 6
    PEP 657, 4, 13
    PEP 659, 20
    PEP 670, 3, 33
    PEP 673, 7
    PEP 675, 7
    PEP 678, 5
    PEP 680, 3, 10
    PEP 681, 8
    PEP 682, 9
    PEP 3333, 10
PYTHONNODEBUGRANGES, 4
PYTHONSAFEPATH, 3, 9
PYTHONTHREADDEBUG, 28

V

variables de entorno
    PYTHONNODEBUGRANGES, 4
    PYTHONSAFEPATH, 3, 9
    PYTHONTHREADDEBUG, 28