
What's New in Python

Release 3.10.19

A. M. Kuchling

outubro 16, 2025

Python Software Foundation
Email: docs@python.org

Sumário

1	Resumo – Destaques da versão	3
2	Novas funcionalidades	4
2.1	Gerenciadores de contexto entre parênteses	4
2.2	Melhores mensagens de erro	4
2.3	PEP 626: Números de linha precisos para depuração e outras ferramentas	8
2.4	PEP 634: Correspondência de padrão estrutural	8
2.5	EncodingWarning opcional e opção encoding="locale"	12
3	Novos recursos relacionados a dicas de tipo	13
3.1	PEP 604: novo operador de união de tipo	13
3.2	PEP 612: variáveis de especificação de parâmetros	13
3.3	PEP 613: TypeAlias	14
3.4	PEP 647: guardas de tipo definidas pelo usuário	14
4	Outras mudanças na linguagem	14
5	Novos módulos	15
6	Módulos melhorados	15
6.1	asyncio	15
6.2	argparse	15
6.3	array	16
6.4	asynchat, asyncore, smtpd	16
6.5	base64	16
6.6	bdb	16
6.7	bisect	16
6.8	codecs	16
6.9	collections.abc	16
6.10	contextlib	16
6.11	curses	17
6.12	dataclasses	17
6.13	distutils	18
6.14	doctest	18
6.15	encodings	18
6.16	fileinput	18
6.17	faulthandler	18
6.18	gc	18

6.19	glob	19
6.20	hashlib	19
6.21	hmac	19
6.22	IDLE e idlelib	19
6.23	importlib.metadata	20
6.24	inspect	20
6.25	itertools	20
6.26	linecache	20
6.27	os	20
6.28	os.path	21
6.29	pathlib	21
6.30	platform	21
6.31	pprint	21
6.32	py_compile	21
6.33	pyclbr	21
6.34	shelve	21
6.35	statistics	22
6.36	site	22
6.37	socket	22
6.38	ssl	22
6.39	sqlite3	23
6.40	sys	23
6.41	tempfile	23
6.42	_thread	23
6.43	threading	23
6.44	traceback	23
6.45	types	23
6.46	typing	24
6.47	unittest	24
6.48	urllib.parse	24
6.49	xml	25
6.50	zipimport	25
7	Otimizações	25
8	Descontinuados	26
9	Removidos	28
10	Portando para Python 3.10	29
10.1	Alterações na sintaxe Python	29
10.2	Alterações na API Python	29
10.3	Alterações na API C	30
11	Alterações de bytecode do CPython	30
12	Mudanças na construção	30
13	Alterações na API C	31
13.1	PEP 652: Mantendo a ABI estável	31
13.2	Novas funcionalidades	31
13.3	Portando para Python 3.10	32
13.4	Descontinuados	33
13.5	Removidos	33
14	Recursos de segurança notáveis no 3.10.7	35
15	Recursos de segurança notáveis no 3.10.8	35
16	Notable Changes in 3.10.12	35

16.1 tarfile	35
17 Notable changes in 3.10.15	35
17.1 ipaddress	35
17.2 email	36
18 Notable changes in 3.10.18	36
18.1 os.path	36
18.2 tarfile	36
Índice	37

Versão 3.10.19

Data outubro 16, 2025

Editor Pablo Galindo Salgado

Este artigo explica os novos recursos no Python 3.10, em comparação com 3.9. Python 3.10 foi lançado em 4 de outubro de 2021. Veja changelog para uma lista completa de mudanças.

1 Resumo – Destaques da versão

Novos recursos de sintaxe:

- **PEP 634**, Correspondência de padrão estrutural: especificação
- **PEP 635**, Correspondência de padrão estrutural: motivação e justificativa
- **PEP 636**, Correspondência de padrão estrutural: Tutorial
- **bpo-12782**, Gerenciadores de contexto entre parênteses agora são permitidos oficialmente.

Novos recursos na biblioteca padrão:

- **PEP 618**, Adiciona verificação de comprimento opcional ao zip.

Melhorias no interpretador:

- **PEP 626**, Números de linha precisos para depuração e outras ferramentas.

Novos recursos de tipagem:

- **PEP 604**, Permite escrever tipos de união como X | Y
- **PEP 612**, Variáveis de especificação de parâmetro
- **PEP 613**, Apelidos de tipo explícitos
- **PEP 647**, Guardas de Tipo Definidas Pelo Usuário

Descontinuações, remoções ou restrições importantes:

- **PEP 644**, Exige OpenSSL 1.1.1 ou mais novo
- **PEP 632**, Descontinua o módulo distutils.
- **PEP 623**, Descontinua e prepara para a remoção do membro wstr em PyUnicodeObject.
- **PEP 624**, Remove APIs codificadoras de Py_UNICODE
- **PEP 597**, Adiciona EncodingWarning opcional

2 Novas funcionalidades

2.1 Gerenciadores de contexto entre parênteses

O uso de parênteses para continuação em várias linhas em gerenciadores de contexto agora é suportado. Isso permite a formatação de uma longa coleção de gerenciadores de contexto em várias linhas de maneira semelhante à que era possível anteriormente com instruções de importação. Por exemplo, todos esses exemplos agora são válidos:

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

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

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

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

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

também é possível usar uma vírgula no final do grupo fechado:

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

Esta nova sintaxe usa as capacidades não LL(1) do novo analisador sintático. Confira [PEP 617](#) para mais detalhes.

(Contribuição de Guido van Rossum, Pablo Galindo e Lysandros Nikolaou em [bpo-12782](#) e [bpo-40334](#).)

2.2 Melhores mensagens de erro

SyntaxErrors

Ao analisar o código que contém parênteses ou chaves não fechados, o interpretador agora inclui o local da chave não fechada de parênteses em vez de exibir *SyntaxError: unexpected EOF while parsing* ou apontando para algum local incorreto. Por exemplo, considere o seguinte código (observe o “{” não fechado):

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

Versões anteriores do interpretador relatavam lugares confusos como local do erro de sintaxe:

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

mas no Python 3.10, um erro mais informativo é emitido:

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

De maneira semelhante, erros envolvendo literais de string não fechadas (entre aspas simples e triplas) agora apontam para o início da string em vez de relatar EOF/EOL.

Essas melhorias são inspiradas em trabalhos anteriores no interpretador PyPy.

(Contribuição de Pablo Galindo em [bpo-42864](#) e Batuhan Taskaya em [bpo-40176](#).)

As exceções `SyntaxError` levantadas pelo interpretador agora destacam o intervalo total de erros da expressão que constitui o próprio erro de sintaxe, em vez de apenas onde o problema foi detectado. Desta forma, em vez de exibir (antes do Python 3.10):

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

agora o Python 3.10 vai exibir a exceção como:

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

Esta melhoria foi contribuída por Pablo Galindo em [bpo-43914](#).

Uma quantidade considerável de novas mensagens especializadas para exceções `SyntaxError` foram incorporadas. Alguns dos mais notáveis são os seguintes:

- Faltando : antes de blocos:

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

(Contribuição de Pablo Galindo em [bpo-42997](#).)

- Tuplas sem parênteses em alvos de compreensão:

```
>>> {x,y for x,y in zip('abcd', '1234')}
File "<stdin>", line 1
    {x,y for x,y in zip('abcd', '1234')}
      ^
SyntaxError: did you forget parentheses around the comprehension_
↳target?
```

(Contribuição de Pablo Galindo em [bpo-43017](#).)

- Faltando vírgulas em literais de coleção e entre expressões:

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

(Contribuição de Pablo Galindo em [bpo-43822](#).)

- Vários tipos de exceção sem parênteses:

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

(Contribuição de Pablo Galindo em [bpo-43149](#).)

- Faltando : em valores em literais de dicionário:

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

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

(Contribuição de Pablo Galindo em [bpo-43823](#).)

- Blocos try sem blocos except ou finally:

```
>>> try:
...     x = 2
...     something = 3
  File "<stdin>", line 3
    something = 3
    ^^^^^^^^^
SyntaxError: expected 'except' or 'finally' block
```

(Contribuição de Pablo Galindo em [bpo-44305](#).)

- Uso de = em vez de == nas comparações:

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

(Contribuição de Pablo Galindo em [bpo-43797](#).)

- Uso de `*` em f-strings:

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

(Contribuição de Pablo Galindo em [bpo-41064](#).)

IndentationErrors

Muitas exceções `IndentationError` agora têm mais contexto sobre que tipo de bloco estava esperando um indentação, incluindo a localização da instrução:

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

AttributeErrors

Ao exibir `AttributeError`, `PyErr_Display()` oferecerá sugestões de nomes de atributos semelhantes no objeto a partir do qual a exceção foi levantada:

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

(Contribuição de Pablo Galindo em [bpo-38530](#).)

Aviso: Observe que isso não funcionará se `PyErr_Display()` não for chamado para exibir o erro que pode ocorrer se alguma outra função personalizada de exibição de erro for usada. Este é um cenário comum em alguns REPLs, laços de leitura-avaliação-impressão, como o IPython.

NameErrors

Ao exibir `NameError` levantada pelo interpretador, `PyErr_Display()` irá oferecer sugestões de nomes de variáveis semelhantes na função de onde a exceção foi levantada:

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

(Contribuição de Pablo Galindo em [bpo-38530](#).)

Aviso: Observe que isso não funcionará se `PyErr_Display()` não for chamado para exibir o erro que pode ocorrer se alguma outra função personalizada de exibição de erro for usada. Este é um cenário comum em alguns REPLs, laços de leitura-avaliação-impressão, como o IPython.

2.3 PEP 626: Números de linha precisos para depuração e outras ferramentas

O PEP 626 traz números de linha mais precisos e confiáveis para ferramentas de depuração, criação de perfil e cobertura. Eventos de rastreamento, com o número de linha correto, são gerados para todas as linhas de código executadas e apenas para linhas de código que são executadas.

O atributo `f_lineno` de objetos de quadro sempre conterá o número de linha esperado.

O atributo `co_lnotab` de objetos de código foi descontinuado e será removido no 3.12. O código que precisa ser convertido do deslocamento para o número da linha deve usar o novo método `co_lines()`.

2.4 PEP 634: Correspondência de padrão estrutural

A correspondência de padrão estrutural foi adicionada na forma de uma *instrução de correspondência* e *instruções de caso* de padrões com ações associadas. Os padrões consistem em sequências, mapeamentos, tipos de dados primitivos, bem como instâncias de classe. A correspondência de padrão permite que os programas extraiam informações de tipos de dados complexos, ramifiquem na estrutura de dados e apliquem ações específicas com base em diferentes formas de dados.

Sintaxe e operações

A sintaxe genérica da correspondência de padrão é:

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

Uma instrução de correspondência pega uma expressão e compara seu valor com padrões sucessivos fornecidos como um ou mais blocos de caso. Especificamente, a correspondência de padrões opera:

1. usando dados com tipo e forma (o sujeito)
2. avaliando o `sujeito` na instrução `match`
3. comparando o assunto com cada padrão em uma instrução `case` de cima para baixo até que uma correspondência seja confirmada.
4. executando a ação associada ao padrão da correspondência confirmada
5. Se uma correspondência exata não for confirmada, no último caso, um curinga `_`, se fornecido, será usado como o caso de correspondência. Se uma correspondência exata não for confirmada e não houver um caractere curinga, todo o bloco de correspondência será autônomo.

Abordagem declarativa

Os leitores podem estar cientes da correspondência de padrão por meio do exemplo simples de correspondência de um assunto (objeto de dados) a um literal (padrão) com a instrução `switch` encontrada em C, Java ou JavaScript (e muitas outras linguagens). Frequentemente, a instrução `switch` é usada para comparação de um objeto/expressão com instruções `case` contendo literais.

Exemplos mais poderosos de correspondência de padrão podem ser encontrados em linguagens como Scala e Elixir. Com a correspondência de padrão estrutural, a abordagem é “declarativa” e declara explicitamente as condições (os padrões) para que os dados correspondam.

Embora uma série “imperativa” de instruções usando instruções “if” aninhadas possa ser usada para realizar algo semelhante à correspondência de padrão estrutural, é menos clara do que a abordagem “declarativa”. Em vez disso, a abordagem “declarativa” estabelece as condições a serem atendidas para uma correspondência e é mais legível por meio de seus padrões explícitos. Embora a correspondência de padrão estrutural possa ser usada em sua forma mais simples comparando uma variável a um literal em uma instrução `case`, seu verdadeiro valor para Python reside em seu tratamento do tipo e forma do sujeito.

Padrão simples: corresponder a um literal

Vejamos este exemplo como correspondência de padrão em sua forma mais simples: um valor, o assunto, sendo correspondido a vários literais, os padrões. No exemplo abaixo, `status` é o assunto da instrução de correspondência. Os padrões são cada uma das instruções de caso, onde literais representam códigos de status de solicitação. A ação associada ao caso é executada após uma partida:

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

Se a função acima receber um `status` de 418, “I’m a teapot” será retornado. Se a função acima receber um `status` de 500, a instrução `case` com `_` irá corresponder a um curinga, e “Something’s wrong with the Internet” é retornado. Observe o último bloco: o nome da variável, `_`, atua como um *curinga* e garante que o assunto sempre corresponderá. O uso de `_` é opcional.

Você pode combinar vários literais em um único padrão usando `|` (“ou”):

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

Comportamento sem o curinga

Se modificarmos o exemplo acima removendo o último bloco `case`, o exemplo se tornará:

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

Sem o uso de `_` em uma instrução `case`, uma correspondência pode não existir. Se não houver correspondência, o comportamento é autônomo. Por exemplo, se o `status` de 500 for passado, ocorre um `no-op`.

Padrões com uma literal e variável

Os padrões podem parecer atribuições de desempacotamento e um padrão pode ser usado para vincular variáveis. Neste exemplo, um ponto de dados pode ser desempacotado em sua coordenada `x` e coordenada `y`:

```
# point is an (x, y) tuple
match point:
    case (0, 0):
        print("Origin")
    case (0, y):
        print(f"Y={y}")
    case (x, 0):
        print(f"X={x}")
    case (x, y):
        print(f"X={x}, Y={y}")
    case _:
        raise ValueError("Not a point")
```

O primeiro padrão tem dois literais, `(0, 0)`, e pode ser considerado uma extensão do padrão literal mostrado acima. Os próximos dois padrões combinam um literal e uma variável, e a variável *vincula* um valor do assunto (`ponto`). O quarto padrão captura dois valores, o que o torna conceitualmente semelhante à atribuição de desempacotamento `(x, y) = ponto`.

Padrões e classes

Se estiver usando classes para estruturar seus dados, você pode usar como padrão o nome da classe seguido por uma lista de argumentos semelhante a um construtor. Este padrão tem a capacidade de capturar atributos de classe em variáveis:

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

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

Padrões com parâmetros posicionais

Você pode usar parâmetros posicionais com algumas classes embutidas que fornecem uma ordem para seus atributos (por exemplo, classes de dados). Você também pode definir uma posição específica para atributos em padrões configurando o atributo especial `__match_args__` em suas classes. Se for definido como ("x", "y"), os seguintes padrões são todos equivalentes (e todos ligam o atributo `y` à variável `var`):

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

Padrões aninhados

Os padrões podem ser aninhados arbitrariamente. Por exemplo, se nossos dados forem uma pequena lista de pontos, eles podem ser correspondidos assim:

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

Padrões complexos e o curinga

Até este ponto, os exemplos usaram `_` sozinho na última instrução `case`. Um curinga pode ser usado em padrões mais complexos, como `('error', code, _)`. Por exemplo:

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

No caso acima, `test_variable` irá corresponder a ('erro', código, 100) e ('erro', código, 800).

Guarda

Podemos adicionar uma cláusula `if` a um padrão, conhecido como "guarda". Se a guarda for falsa, `match` tenta o próximo bloco de caso. Observe que a captura de valor ocorre antes que a guarda seja avaliada:

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

Outros recursos-chave

Vários outros recursos-chave:

- Assim como desempacotar atribuições, os padrões de tupla e lista têm exatamente o mesmo significado e realmente correspondem a sequências arbitrárias. Tecnicamente, o sujeito deve ser uma sequência. Portanto, uma exceção importante é que padrões não correspondem a iteradores. Também evita um erro comum, sequência de padrões não correspondem a strings.
- Os padrões de sequência têm suporte a curingas: `[x, y, *rest]` e `(x, y, *rest)` funcionam de forma semelhante a curingas em desempacotamentos de atribuições. O nome depois de `*` também pode ser `_`, então `(x, y, *_)` corresponde a uma sequência de pelo menos dois itens sem ligar os itens restantes.
- Padrões de mapeamento: `{"bandwidth": b, "latency": l}` captura os valores "bandwidth" e "latency" de um dicionário. Diferente dos padrões de sequência, chaves extra são ignoradas. Um curinga `**rest` também é permitido. (Mas `**_` seria redundante, então não é permitido.)
- Subpadrões podem ser capturados usando a palavra reservada `as`

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

Isso liga `x1`, `y1`, `x2`, `y2` como você esperaria sem a cláusula `as` e `p2` a todo o segundo item do sujeito.

- A maioria dos literais são comparados por igualdade. No entanto, os singletons `True`, `False` e `None` são comparados por identidade.
- Constantes nomeadas podem ser usadas em padrões. Essas constantes nomeadas devem ser nomes pontilhados para evitar que a constante seja interpretada como uma variável de captura:

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

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

Para obter as especificações completas, consulte a [PEP 634](#). A motivação e o raciocínio estão na [PEP 635](#), e um tutorial mais longo está na [PEP 636](#).

2.5 EncodingWarning opcional e opção encoding="locale"

A codificação padrão de `TextIOWrapper` e `open()` depende da plataforma e da localidade. Como o UTF-8 é usado na maioria das plataformas Unix, omitir a opção `encoding` ao abrir arquivos UTF-8 (por exemplo, JSON, YAML, TOML, Markdown) é um bug muito comum. Por exemplo:

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

Para encontrar este tipo de bug, uma `EncodingWarning` opcional é adicionada. É emitido quando `sys.flags.warn_default_encoding` é verdadeiro e a codificação padrão específica da localidade é usada.

A opção `-X warn_default_encoding` e `PYTHONWARNDEFAULTENCODING` são adicionadas para ativar o aviso.

Veja `io-text-encoding` para mais informações.

3 Novos recursos relacionados a dicas de tipo

Esta seção cobre as principais mudanças que afetam as dicas de tipo da **PEP 484** e o módulo `typing`.

3.1 PEP 604: novo operador de união de tipo

Um novo operador de união de tipo foi introduzido, o que permite a sintaxe `X | Y`. Isso fornece uma maneira mais limpa de expressar “tanto o tipo X quanto o tipo Y” ao invés de usar `typing.Union`, especialmente em dicas de tipo.

Nas versões anteriores do Python, para aplicar uma dica de tipo para funções que aceitam argumentos de vários tipos, era usado `typing.Union`:

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

As dicas de tipo agora podem ser escritas de uma maneira mais sucinta:

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

Esta nova sintaxe também é aceita como o segundo argumento para `isinstance()` e `issubclass()`:

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

Veja `types-union` e **PEP 604** para mais detalhes.

(Contribuição de Maggie Moss e Philippe Prados em [bpo-41428](#), com acréscimos por Yurii Karabas e Serhiy Stor-chaka em [bpo-44490](#).)

3.2 PEP 612: variáveis de especificação de parâmetros

Duas novas opções para melhorar as informações fornecidas para verificadores de tipo estático para `Callable` da **PEP 484** foram adicionadas ao módulo `typing`.

A primeira é a variável de especificação de parâmetro. Eles são usados para encaminhar os tipos de parâmetro de um chamável para outro chamável – um padrão comumente encontrado em funções de ordem superior e decoradores. Exemplos de uso podem ser encontrados em `typing.ParamSpec`. Anteriormente, não havia uma maneira fácil de digitar a dependência de anotação de tipos de parâmetro de maneira tão precisa.

A segunda opção é o novo operador `Concatenate`. É usado em conjunto com variáveis de especificação de parâmetro para digitar anotar um chamável de ordem superior que adiciona ou remove parâmetros de outro chamável. Exemplos de uso podem ser encontrados em `typing.Concatenate`.

Veja `typing.Callable`, `typing.ParamSpec`, `typing.Concatenate`, `typing.ParamSpecArgs`, `typing.ParamSpecKwargs` e **PEP 612** para mais detalhes.

(Contribuição de Ken Jin em [bpo-41559](#), com pequenas melhorias por Jelle Zijlstra em [bpo-43783](#). PEP escrita por Mark Mendoza.)

3.3 PEP 613: TypeAlias

PEP 484 introduziu o conceito de apelidos de tipo, exigindo apenas que fossem atribuições não anotadas de nível superior. Essa simplicidade às vezes tornava difícil para os verificadores de tipo distinguirem entre apelidos de tipo e atribuições comuns, especialmente quando referências diretas ou tipos inválidos estavam envolvidos. Compare:

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

Agora o módulo `type` tem um valor especial `TypeAlias` que permite declarar apelidos de tipo, mais explicitamente:

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

Veja **PEP 613** para mais detalhes.

(Contribuição de Mikhail Golubev em [bpo-41923](#).)

3.4 PEP 647: guardas de tipo definidas pelo usuário

`TypeGuard` foi adicionado ao módulo `typing` para anotar funções de guarda de tipo e melhorar informações fornecidas a verificadores de tipo estático durante um estreitamento de tipo. Para mais informações, veja a documentação do `TypeGuard` e a **PEP 647**.

(Contribuição de Ken Jin e Guido van Rossum em [bpo-43766](#). PEP escrita por Eric Traut.)

4 Outras mudanças na linguagem

- O tipo `int` tem um novo método `int.bit_count()`, retornando o número de unidades na expansão binária de um dado inteiro, também conhecido como contagem da população. (Contribuição de Niklas Fiekas em [bpo-29882](#).)
- As visualizações retornadas por `dict.keys()`, `dict.values()` e `dict.items()` agora têm um atributo `mapping` que fornece um objeto `types.MappingProxyType` que envolve o dicionário original. (Contribuição de Dennis Sweeney em [bpo-40890](#).)
- **PEP 618**: A função `zip()` agora tem um sinalizador opcional `strict`, usado para exigir que todos os iteráveis tenham um comprimento igual.
- Funções embutidas e de extensão que recebem argumentos inteiros não aceitam mais `Decimal`, `Fraction` e outros objetos que podem ser convertidos em inteiros apenas com uma perda (por exemplo, tem o método `__int__()`, mas não tem o método `__index__()`). (Contribuição de Serhiy Storchaka em [bpo-37999](#).)
- Se `object.__ipow__()` retorna `NotImplemented`, o operador vai corretamente recorrer ao `object.__pow__()` e `object.__rpow__()` como esperado. (Contribuição de Alex Shkop em [bpo-38302](#).)
- Expressões de atribuição agora podem ser usadas sem parênteses dentro de literais de conjuntos e compreensões de conjuntos, bem como em índices de sequência (mas não em fatias).
- As funções têm um novo atributo `__builtins__` que é usado para procurar por símbolos embutidos quando uma função é executada, em vez de procurar em `__globals__['__builtins__']`. O atributo é inicializado a partir de `__globals__["__builtins__"]` se existir; do contrário, a partir dos embutidos atuais. (Contribuição de Mark Shannon em [bpo-42990](#).)
- Duas novas funções embutidas – `aiter()` e `anext()` foram adicionadas para fornecer contrapartes assíncronas para `iter()` e `next()`, respectivamente. (Contribuição de Joshua Bronson, Daniel Pope e Justin Wang em [bpo-31861](#).)

- Métodos estáticos (`@staticmethod`) e métodos de classe (`@classmethod`) agora herdam os atributos de método (`__module__`, `__name__`, `__qualname__`, `__doc__`, `__annotations__`) e têm um novo atributo `__wrapped__`. Além disso, métodos estáticos são agora chamáveis como funções comuns. (Contribuição de Victor Stinner em [bpo-43682](#).)
- Anotações para alvos complexos (tudo além de alvos de nome simples, ou `simple name`, definidos pela [PEP 526](#)) não causam mais nenhum efeito de tempo de execução com `from __future__ import annotations`. (Contribuição de Batuhan Taskaya em [bpo-42737](#).)
- Objetos classe e módulo agora criam preguiçosamente dados vazios de anotações sob demanda. Os dicionários de anotações são armazenados em `__dict__` do objeto para compatibilidade com versões anteriores. Isso melhora as melhores práticas para trabalhar com `__annotations__`; para mais informações, veja `annotations-howto`. (Contribuição de Larry Hastings em [bpo-43901](#).)
- Anotações consistindo em `yield`, `yield from`, `await` ou expressões nomeadas agora são proibidas em `from __future__ import annotations` por causa de seus efeitos colaterais. (Contribuição de Batuhan Taskaya em [bpo-42725](#).)
- O uso de variáveis não ligadas, `super()` e outras expressões que podem alterar o processamento da tabela de símbolos como anotações são agora processadas sem efeito sob `from __future__ import annotations`. (Contribuição de Batuhan Taskaya em [bpo-42725](#).)
- Hashes de valores NaN de ambos os tipos `float` e `decimal.Decimal` agora dependem da identidade do objeto. Anteriormente, eles sempre hash para 0 mesmo que os valores NaN não sejam iguais uns aos outros. Isso causou um comportamento de tempo de execução potencialmente quadrático devido a colisões de hash excessivas ao criar dicionários e conjuntos contendo vários NaNs. (Contribuição de Raymond Hettinger em [bpo-43475](#).)
- Uma `SyntaxError` (ao invés de uma `NameError`) será levantada ao excluir a constante `__debug__`. (Contribuição de Dong-hee Na em [bpo-45000](#).)
- Exceções `SyntaxError` agora possuem atributos `end_lineno` e `end_offset`. Eles serão `None` se não forem determinados. (Contribuição de Pablo Galindo em [bpo-43914](#).)

5 Novos módulos

- Nada ainda.

6 Módulos melhorados

6.1 asyncio

Adiciona o método `connect_accepted_socket()` até então em falta. (Contribuição de Alex Grönholm em [bpo-41332](#).)

6.2 argparse

A frase enganosa “argumentos opcionais” foi substituída por “opções” na ajuda do `argparse`. Alguns testes podem exigir adaptação se eles dependerem da correspondência de saída exata. (Contribuição de Raymond Hettinger em [bpo-9694](#).)

6.3 array

O método `index()` de `array.array` agora possui os parâmetros *start* e *stop*. (Contribuição de Anders Lorentsen e Zackery Spytz em [bpo-31956](#).)

6.4 asynchat, asyncore, smtpd

Esses módulos foram marcados como descontinuados em sua documentação de módulo desde o Python 3.6. Uma exceção `DeprecationWarning` em tempo de importação agora foi adicionada a todos esses três módulos.

6.5 base64

Adiciona `base64.b32hexencode()` e `base64.b32hexdecode()` para dar suporte a Codificação Base32 com alfabeto hexa estendido.

6.6 bdb

Adiciona `clearBreakpoints()` para redefinir todos os pontos de interrupção definidos. (Contribuição de Irit Katriel em [bpo-24160](#).)

6.7 bisect

Adicionada a possibilidade de fornecer uma função *key* para as APIs no módulo `bisect`. (Contribuição de Raymond Hettinger em [bpo-4356](#).)

6.8 codecs

Adiciona uma função `codecs.unregister()` para cancelar um registro de uma função de pesquisa de codecs. (Contribuição de Hai Shi em [bpo-41842](#).)

6.9 collections.abc

Os `__args__` do genérico parametrizado para `collections.abc.Callable` agora são consistentes com `typing.Callable`. A classe genérica `collections.abc.Callable` agora achata parâmetros de tipo, de forma semelhante ao que `typing.Callable` atualmente faz. Isso significa que `collections.abc.Callable[[int, str], str]` terá `__args__` de `(int, str, str)`; anteriormente, isso era `([int, str], str)`. Para permitir esta alteração, agora é possível criar subclasse de `types.GenericAlias` e uma subclasse será retornada ao fazer um subscript do tipo `collections.abc.Callable`. Observe que uma `TypeError` pode ser levantada para formas inválidas de parametrizar `collections.abc.Callable` que podem ter passado silenciosamente no Python 3.9. (Contribuição de Ken Jin em [bpo-42195](#).)

6.10 contextlib

Adiciona um gerenciador de contexto `contextlib.aclosing()` para fechar com segurança geradores assíncronos e objetos que representam recursos liberados de forma assíncrona. (Contribuição de Joongi Kim e John Belmonte em [bpo-41229](#).)

Adiciona suporte a gerenciador de contexto assíncrono a `contextlib.nullcontext()`. (Contribuição de Tom Gringauz em [bpo-41543](#).)

Adiciona `AsyncContextDecorator`, para dar suporte ao uso de gerenciadores de contexto assíncronos como decoradores.

6.11 curses

As funções de cores estendidas adicionadas no ncurses 6.1 serão usadas transparentemente por `curses.color_content()`, `curses.init_color()`, `curses.init_pair()` e `curses.pair_content()`. Uma nova função, `curses.has_extended_color_support()`, indica se o suporte a cores estendidas é fornecido pela biblioteca ncurses subjacente. (Contribuição de Jeffrey Kintscher e Hans Petter Jansson em [bpo-36982](#).)

As constantes `BUTTON5_*` agora são expostas no módulo `curses` se forem fornecidas pela biblioteca curses subjacente. (Contribuição de Zackery Spytz em [bpo-39273](#).)

6.12 dataclasses

`__slots__`

Adicionado o parâmetro `slots` no decorador `dataclasses.dataclass()`. (Contribuição de Yuri Karabas em [bpo-42269](#))

Campos somente-nomeados

`dataclasses` agora oferece suporte a campos que são somente-nomeados no método `__init__` gerado. Há várias maneiras de especificar campos somente-nomeados.

Você pode dizer que todos os campos são somente-nomeados:

```
from dataclasses import dataclass

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

Ambos `name` e `birthday` são parâmetros somente-nomeados para o método `__init__` gerado.

Você pode especificar somente-nomeado por campo:

```
from dataclasses import dataclass

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

Aqui apenas `birthday` é somente-nomeado. Se você definir `kw_only` em campos individuais, esteja ciente de que existem regras sobre a reordenação de campos devido a campos somente-nomeados que precisam seguir campos não somente-nomeados. Consulte a documentação completa de `dataclasses` para obter detalhes.

Você também pode especificar que todos os campos a partir de um marcador `KW_ONLY` sejam somente-nomeados. Este provavelmente será o uso mais comum:

```
from dataclasses import dataclass, KW_ONLY

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

Aqui, `z` e `t` são parâmetros somente-nomeados, enquanto `x` e `y` não são. (Contribuição de Eric V. Smith em [bpo-43532](#).)

6.13 distutils

Todo o pacote `distutils` foi descontinuado, para ser removido no Python 3.12. Sua funcionalidade para especificar compilações de pacote já foi completamente substituída por pacotes de terceiros `setuptools` e `packaging`, e a maioria das outras APIs comumente usadas estão disponíveis em outro lugar na biblioteca padrão (como `platform`, `shutil`, `subprocess` ou `sysconfig`). Não há planos para migrar qualquer outra funcionalidade de `distutils`, e aplicativos que estão usando outras funções devem planejar fazer cópias privadas do código. Consulte [PEP 632](#) para discussão.

O comando descontinuado `bdist_wininst` no Python 3.8 foi removido. O comando `bdist_wheel` agora é recomendado para distribuir pacotes binários no Windows. (Contribuição de Victor Stinner em [bpo-42802](#).)

6.14 doctest

Quando um módulo não define `__loader__`, recorre a `__spec__.loader`. (Contribuição de Brett Cannon em [bpo-42133](#).)

6.15 encodings

`encodings.normalize_encoding()` agora ignora caracteres não-ASCII. (Contribuição de Hai Shi em [bpo-39337](#).)

6.16 fileinput

Adiciona os parâmetros `encoding` e `errors` a `fileinput.input()` e `fileinput.FileInput`. (Contribuição de Inada Naoki em [bpo-43712](#).)

`fileinput.hook_compressed()` agora retorna um objeto `TextIOWrapper` quando `mode` é “r” e o arquivo está compactado, como arquivos descompactados. (Contribuição de Inada Naoki em [bpo-5758](#).)

6.17 faulthandler

O módulo `faulthandler` agora detecta se um erro fatal ocorre durante a coleta do coletor de lixo. (Contribuição de Victor Stinner em [bpo-44466](#).)

6.18 gc

Adiciona ganchos de auditoria para `gc.get_objects()`, `gc.get_referrers()` e `gc.get_referents()`. (Contribuição de Pablo Galindo em [bpo-43439](#).)

6.19 glob

Adiciona os parâmetros `root_dir` e `dir_fd` em `glob()` e `iglob()`, o que permite especificar o diretório raiz para a pesquisa. (Contribuição de Serhiy Storchaka em [bpo-38144](#).)

6.20 hashlib

O módulo `hashlib` requer OpenSSL 1.1.1 ou mais recente. (Contribuição de Christian Heimes em [PEP 644](#) e [bpo-43669](#).)

O módulo `hashlib` tem suporte preliminar a OpenSSL 3.0.0. (Contribuição de Christian Heimes em [bpo-38820](#) e outras issues.)

A alternativa puramente Python de `pbkdf2_hmac()` foi descontinuada. No futuro, o PBKDF2-HMAC só estará disponível quando o Python for desenvolvido com suporte a OpenSSL. (Contribuição de Christian Heimes em [bpo-43880](#).)

6.21 hmac

O módulo `hmac` agora usa a implementação HMAC do OpenSSL internamente. (Contribuição de Christian Heimes em [bpo-40645](#).)

6.22 IDLE e idlelib

Faz o IDLE invocar `sys.excepthook()` (quando iniciado sem `-n`). Ganchos de usuário eram ignorados anteriormente. (Contribuição de Ken Hilton em [bpo-43008](#).)

Reorganiza a caixa de diálogo de configurações. Divide a aba General nas abas Windows e Shell/Ed. Move as fontes de ajuda, que estendem o menu Help, para a aba Extensions. Abre espaço para novas opções e encurta a caixa de diálogo. O último faz com que o diálogo se ajuste melhor a telas pequenas. (Contribuição de Terry Jan Reedy em [bpo-40468](#).) Move a configuração do espaço de recuo da aba Font para a nova aba Windows. (Contribuição de Mark Roseman e Terry Jan Reedy em [bpo-33962](#).)

As alterações acima foram portadas para uma versão de manutenção 3.9.

Adiciona uma barra lateral ao console. Move o prompt principal (`>>>`) para a barra lateral. Adiciona prompts secundários (`...`) à barra lateral. Clicar com o botão esquerdo e opcionalmente arrastar seleciona uma ou mais linhas de texto, como na barra lateral do número da linha do editor. Clicar com o botão direito após selecionar as linhas de texto exibe um menu de contexto com “copy with prompts”. Isso compacta os prompts da barra lateral com linhas do texto selecionado. Esta opção também aparece no menu de contexto para o texto. (Contribuição de Tal Einat em [bpo-37903](#).)

Use espaços em vez de tabulações para indentar o código interativo. Isso faz com que as entradas de código interativo “pareçam corretas”. Tornar isso viável foi a principal motivação para adicionar a barra lateral do console. Contribuição de Terry Jan Reedy em [bpo-37892](#).)

Realça as novas palavras reservadas `match`, `case` e `_` em instruções de correspondência de padrões. No entanto, este realce não é perfeito e estará incorreto em alguns casos raros, incluindo alguns `_` em padrões de `case`. (Contribuição de Tal Einat em [bpo-44010](#).)

Novo nas versões de manutenção 3.10.

Aplica realce de sintaxe em arquivos `.pyi`. (Contribuição de Alex Waygood e Terry Jan Reedy em [bpo-45447](#).)

Inclui prompts ao salvar o console com entradas e saídas. (Contribuição de Terry Jan Reedy em [gh-95191](#).)

6.23 importlib.metadata

Paridade de recursos com `importlib_metadata` 4.6 ([histórico](#)).

Pontos de entrada `importlib.metadata` agora oferecem uma experiência melhor para selecionar pontos de entrada por grupo e nome através de uma nova classe `importlib.metadata.EntryPoints`. Consulte a Nota de Compatibilidade nos documentos para obter mais informações sobre a descontinuação e uso.

Adicionada `importlib.metadata.packages_distributions()` para resolver módulos e pacotes Python de alto nível com suas `importlib.metadata.Distribution`.

6.24 inspect

Quando um módulo não define `__loader__`, recorre a `__spec__.loader`. (Contribuição de Brett Cannon em [bpo-42133](#).)

Adiciona `inspect.get_annotations()`, que calcula com segurança as anotações definidas em um objeto. Ele contorna as peculiaridades de acessar as anotações em vários tipos de objetos e faz poucas suposições sobre o objeto que examina. `inspect.get_annotations()` também pode desfazer a string corretamente de anotações em string. `inspect.get_annotations()` agora é considerada a melhor prática para acessar o dict de anotações definido em qualquer objeto Python; para mais informações sobre as melhores práticas para trabalhar com anotações, consulte [annotations-howto](#). Da mesma forma, `inspect.signature()`, `inspect.Signature.from_callable()` e `inspect.Signature.from_function()` agora chamam `inspect.get_annotations()` para recuperar anotações. Isso significa que `inspect.signature()` e `inspect.Signature.from_callable()` agora também podem remover a string de anotações em string. (Contribuição de Larry Hastings em [bpo-43817](#).)

6.25 itertools

Adiciona `itertools.pairwise()`. (Contribuição de Raymond Hettinger em [bpo-38200](#).)

6.26 linecache

Quando um módulo não define `__loader__`, recorre a `__spec__.loader`. (Contribuição de Brett Cannon em [bpo-42133](#).)

6.27 os

Adiciona suporte a `os.cpu_count()` para RTOS de VxWorks. (Contribuição de Peixing Xin em [bpo-41440](#).)

Adiciona uma nova função `os.eventfd()` e auxiliares relacionados para envolver a chamada de sistema `eventfd2` no Linux. (Contribuição de Christian Heimes em [bpo-41001](#).)

Adiciona `os.splice()` que permite mover dados entre dois descritores de arquivo sem copiar entre o espaço de endereço do kernel e o espaço de endereço do usuário, onde um dos descritores de arquivo deve se referir a um encadeamento (pipe). (Contribuição de Pablo Galindo em [bpo-41625](#).)

Adiciona `O_EVTONLY`, `O_FSYNC`, `O_SYMLINK` e `O_NOFOLLOW_ANY` para macOS. (Contribuição de Dong-hee Na em [bpo-43106](#).)

As of 3.10.15, `os.mkdir()` and `os.makedirs()` on Windows now support passing a *mode* value of `0o700` to apply access control to the new directory. This implicitly affects `tempfile.mkdtemp()` and is a mitigation for CVE-2024-4030. Other values for *mode* continue to be ignored. (Contributed by Steve Dower in [gh-118486](#).)

6.28 os.path

`os.path.realpath()` agora aceita um argumento somente-nomeado *strict*. Quando definido como `True`, a exceção `OSError` é levantada se um caminho não existe ou um loop de link simbólico é encontrado. (Contribuição de Barney Gale em [bpo-43757](#).)

6.29 pathlib

Adiciona suporte a fatiamento a `PurePath.parents`. (Contribuição de Joshua Cannon em [bpo-35498](#).)

Adiciona suporte a indexação negativa a `PurePath.parents`. (Contribuição de Yaroslav Pankovych em [bpo-21041](#).)

Adiciona o método `Path.hardlink_to` que substitui `link_to()`. O novo método tem a mesma ordem de argumentos que `symlink_to()`. (Contribuição de Barney Gale em [bpo-39950](#).)

`pathlib.Path.stat()` e `chmod()` agora aceita um argumento somente-nomeado *follow_symlinks* para consistência com as funções correspondentes no módulo `os`. (Contribuição de Barney Gale em [bpo-39906](#).)

6.30 platform

Adiciona `platform.freedesktop_os_release()` para obter a identificação do sistema operacional a partir do arquivo padrão `os-release` do [freedesktop.org](#). (Contribuição de Christian Heimes em [bpo-28468](#).)

6.31 pprint

`pprint.pprint()` agora aceita um novo argumento nomeado `underscore_numbers`. (Contribuição de sbldon em [bpo-42914](#).)

`pprint` agora pode fazer impressão bonita de instâncias de `dataclasses.dataclass`. (Contribuição de Lewis Gaul em [bpo-43080](#).)

6.32 py_compile

Adiciona a opção `--quiet` à interface de linha de comando de `py_compile`. (Contribuição de Gregory Schevchenko em [bpo-38731](#).)

6.33 pyclbr

Adiciona um atributo `end_lineno` aos objetos `Function` e `Class` na árvore retornada por `pyclbr.readline()` e `pyclbr.readline_ex()`. Isso corresponde ao `lineno` (início) existente. (Contribuição de Aviral Srivastava em [bpo-38307](#).)

6.34 shelve

O módulo `shelve` agora usa `pickle.DEFAULT_PROTOCOL` por padrão em vez do protocolo 3 do `pickle` ao criar “shelves”. (Contribuição de Zackery Spytz em [bpo-34204](#).)

6.35 statistics

Adiciona `covariance()`, `correlation()` do Pearson, e funções simples `linear_regression()`. (Contribuição de Tymoteusz Wołodźko em [bpo-38490](#).)

6.36 site

Quando um módulo não define `__loader__`, recorre a `__spec__.loader`. (Contribuição de Brett Cannon em [bpo-42133](#).)

6.37 socket

A exceção `socket.timeout` é agora um apelido de `TimeoutError`. (Contribuição de Christian Heimes em [bpo-42413](#).)

Adiciona a opção de criar soquetes MPTCP com `IPPROTO_MPTCP` (Contribuição de Rui Cunha em [bpo-43571](#).)

Adiciona a opção `IP_RECVTOS` para receber o tipo do serviço (ToS) ou campos DSCP/ECN (Contribuição de Georg Sauthoff em [bpo-44077](#).)

6.38 ssl

O módulo `ssl` requer OpenSSL 1.1.1 ou mais recente. (Contribuição de Christian Heimes em [PEP 644](#) e [bpo-43669](#).)

O módulo `ssl` tem um suporte preliminar para OpenSSL 3.0.0 e a nova opção `OP_IGNORE_UNEXPECTED_EOF`. (Contribuição de Christian Heimes em [bpo-38820](#), [bpo-43794](#), [bpo-43788](#), [bpo-43791](#), [bpo-43799](#), [bpo-43920](#), [bpo-43789](#) e [bpo-43811](#).)

Função descontinuada e o uso de constantes descontinuadas agora resultam em uma `DeprecationWarning`. `ssl.SSLContext.options` tem `OP_NO_SSLv2` e `OP_NO_SSLv3` definidos por padrão e, portanto, não consegue avisar sobre a definição do sinalizador novamente. A *seção de descontinuidade* tem uma lista de recursos descontinuados. (Contribuição de Christian Heimes em [bpo-43880](#).)

O módulo `ssl` agora tem configurações padrão mais seguras. Cifras sem *forward secrecy* ou SHA-1 MAC são desabilitadas por padrão. O nível de segurança 2 proíbe chaves fracas RSA, DH e ECC com menos de 112 bits de segurança. `SSLContext` assume como padrão a versão mínima do protocolo TLS 1.2. As configurações são baseadas na pesquisa de Hynek Schlawack. (Contribuição de Christian Heimes em [bpo-43998](#).)

Os protocolos descontinuados SSL 3.0, TLS 1.0 e TLS 1.1 não são mais oficialmente suportados. Python não os bloqueia ativamente. No entanto, as opções de compilação do OpenSSL, configurações de distro, patches de fornecedores e suítes de criptografia podem impedir um handshake bem-sucedido.

Adiciona um parâmetro `timeout` à função `ssl.get_server_certificate()`. (Contribuição de Zackery Spytz em [bpo-31870](#).)

O módulo `ssl` usa tipos de heap e inicialização multifásica. (Contribuição de Christian Heimes em [bpo-42333](#).)

Uma nova sinalização de verificação `VERIFY_X509_PARTIAL_CHAIN` foi adicionada. (Contribuição de I0x em [bpo-40849](#).)

6.39 sqlite3

Adiciona eventos de auditoria para `connect/handle()`, `enable_load_extension()` e `load_extension()`. (Contribuição de Erlend E. Aasland em [bpo-43762](#).)

6.40 sys

Adiciona o atributo `sys.orig_argv`: a lista de argumentos de linha de comando originais passada para o executável Python. (Contribuição de Victor Stinner em [bpo-23427](#).)

Adiciona `sys.stdlib_module_names`, contendo a lista de nomes de módulos da biblioteca padrão. (Contribuição de Victor Stinner em [bpo-42955](#).)

6.41 tempfile

As of 3.10.15 on Windows, the default mode `0o700` used by `tempfile.mkdtemp()` now limits access to the new directory due to changes to `os.mkdir()`. This is a mitigation for CVE-2024-4030. (Contributed by Steve Dower in [gh-118486](#).)

6.42 _thread

`_thread.interrupt_main()` agora aceita um número de sinal opcional para simular (o padrão ainda é `signal.SIGINT`). (Contribuição de Antoine Pitrou em [bpo-43356](#).)

6.43 threading

Adiciona `threading.gettrace()` e `threading.getprofile()` para obter as funções definidas por `threading.settrace()` e `threading.setprofile()`, respectivamente. (Contribuição de Mario Corchero em [bpo-42251](#).)

Adiciona `threading.__excepthook__` para permitir obtenção do valor original de `threading.excepthook()` no caso dele estar definido com um valor quebrado ou diferente. (Contribuição de Mario Corchero em [bpo-42308](#).)

6.44 traceback

As funções `format_exception()`, `format_exception_only()` e `print_exception()` podem agora receber um objeto exceção como um argumento somente-posicional. (Contribuição de Zackery Spytz e Matthias Bussonnier em [bpo-26389](#).)

6.45 types

Reintroduz as classes `types.EllipsisType`, `types.NoneType` e `types.NotImplementedType`, fornecendo um novo conjunto de tipos prontamente interpretáveis pelos verificadores de tipo. (Contribuição de Bas van Beek em [bpo-41810](#).)

6.46 typing

Para alterações principais, veja [Novos recursos relacionados a dicas de tipo](#).

O comportamento de `typing.Literal` foi alterado para ficar em conformidade com a [PEP 586](#) e para corresponder ao comportamento de verificadores de tipo estático especificados na PEP.

1. `Literal` agora elimina a duplicação de parâmetros.
2. Comparações de igualdade entre objetos `Literal` agora são independentes da ordem.
3. Comparações de `Literal` agora respeitam os tipos. Por exemplo, `Literal[0] == Literal[False]` avaliava anteriormente como `True`. Agora é `False`. Para oferecer suporte a essa mudança, o cache de tipo usado internamente agora oferece suporte a tipos de diferenciação.
4. Objetos `Literal` agora irão levantar uma exceção `TypeError` durante as comparações de igualdade se algum de seus parâmetros não for hasháveis. Observe que declarar `Literal` com parâmetros inalteráveis não acusará um erro:

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

(Contribuição de Yurii Karabas em [bpo-42345](#).)

Adiciona uma nova função `typing.is_typeddict()` para fazer introspecção se uma anotação for uma `typing.TypedDict`. (Contribuição de Patrick Reader em [bpo-41792](#).)

Subclasses de `typing.Protocol` que apenas têm variáveis de dados declaradas agora irão levantar um `TypeError` quando verificadas com `isinstance` a menos que sejam decoradas com `runtime_checkable()`. Anteriormente, essas verificações eram aprovadas silenciosamente. Os usuários devem decorar suas subclasses com o decorador `runtime_checkable()` se quiserem protocolos de tempo de execução. (Contribuição de Yurii Karabas em [bpo-38908](#).)

A importação dos submódulos `typing.io` e `typing.re` agora emitirá `DeprecationWarning`. Esses submódulos fora descontinuados desde o Python 3.8 e serão removidos em uma versão futura do Python. Qualquer coisa pertencente a esses submódulos deve ser importada diretamente de `typing`. (Contribuição de Sebastian Rittau em [bpo-38291](#).)

6.47 unittest

Adiciona novo método `new assertNoLogs()` para complementar o existente `assertLogs()`. (Contribuição de Kit Yan Choi em [bpo-39385](#).)

6.48 urllib.parse

Versões do Python anteriores ao Python 3.10 permitiam o uso de `;` e `&` como separadores de parâmetros de consulta em `urllib.parse.parse_qs()` e `urllib.parse.parse_qsl()`. Devido a questões de segurança e em conformidade com as recomendações mais recentes do W3C, isso foi alterado para permitir apenas uma única chave separadora, com `&` como padrão. Esta mudança também afeta `cgi.parse()` e `cgi.parse_multipart()` já que elas usam as funções afetadas internamente. Para obter mais detalhes, consulte a respectiva documentação. (Contribuição de Adam Goldschmidt, Senthil Kumaran e Ken Jin em [bpo-42967](#).)

A presença de caracteres de nova linha ou tab em partes de um URL permite algumas formas de ataques. Seguindo a especificação WHATWG que atualiza [RFC 3986](#), nova linha ASCII `\n`, `\r` e os caracteres de tabulação `\t` são retirados da URL pelo analisador sintático em `urllib.parse` impedindo tais ataques. Os caracteres de remoção são controlados por uma nova variável de nível de módulo `urllib.parse._UNSAFE_URL_BYTES_TO_REMOVE`. (Veja [bpo-43882](#))

6.49 xml

Adiciona uma classe `LexicalHandler` ao módulo `xml.sax.handler`. (Contribuição de Jonathan Gossage e Zackery Spytz em [bpo-35018](#).)

6.50 zipimport

Adiciona métodos relacionados à **PEP 451**: `find_spec()`, `zipimport.zipimporter.create_module()` e `zipimport.zipimporter.exec_module()`. (Contribuição de Brett Cannon em [bpo-42131](#).)

Adiciona o método `invalidate_caches()`. (Contribuição de Desmond Cheong em [bpo-14678](#).)

7 Otimizações

- Os construtores `str()`, `bytes()` e `bytearray()` estão agora mais rápido (cerca de 30–40% para objetos pequenos). (Contribuição de Serhiy Storchaka em [bpo-41334](#).)
- O módulo `runpy` agora importa menos módulos. O tempo de inicialização do comando `python3 -m module-name` é 1,4 vezes mais rápido em média. No Linux, `python3 -I -m module-name` importa 69 módulos no Python 3.9, sendo que importa apenas 51 módulos (-18) no Python 3.10. (Contribuição de Victor Stinner em [bpo-41006](#) e [bpo-41718](#).)
- A instrução `LOAD_ATTR` agora usa o novo mecanismo “cache por opcode”. É cerca de 36% mais rápido agora para atributos regulares e 44% mais rápido para slots. (Contribuição de Pablo Galindo e Yury Selivanov em [bpo-42093](#) e Guido van Rossum em [bpo-42927](#), com base em ideias implementadas originalmente em PyPy e MicroPython.)
- Ao construir o Python com `--enable-optimizations` agora, `-fno-semantic-interposition` é adicionado à linha de compilação e vinculação. Isso acelera as compilações do interpretador Python criado com `--enable-shared` com `gcc` em até 30%. Consulte [este artigo](#) para mais detalhes. (Contribuição de Victor Stinner e Pablo Galindo em [bpo-38980](#).)
- Usa um novo código de gerenciamento de buffer de saída para os módulos `bz2` / `lzma` / `zlib` e adiciona a função `.readall()` à classe `_compression.DecompressReader`. A descompressão `bz2` agora é 1,09x ~ 1.17x mais rápida, a descompressão `lzma` 1.20x ~ 1.32x mais rápida, `GzipFile.read(-1)` 1,11x ~ 1.18x mais rápida. (Contribuição de Ma Lin, revisada por Gregory P. Smith, em [bpo-41486](#).)
- Ao usar anotações em strings, dicts de anotações para funções não são mais criados quando a função é criada. Em vez disso, eles são armazenados como uma tupla de strings, e o objeto função converte lentamente isso no dict de anotações sob demanda. Essa otimização reduz pela metade o tempo de CPU necessário para definir uma função anotada. (Contribuição de Yurii Karabas e Inada Naoki em [bpo-42202](#).)
- Funções de pesquisa de substring como `str1 in str2` e `str2.find(str1)` agora às vezes usam o algoritmo de pesquisa de string “Two-Way” de Crochemore & Perrin para evitar comportamento quadrático em strings longas. (Contribuição de Dennis Sweeney em [bpo-41972](#).)
- Adiciona micro-otimizações a `_PyType_Lookup()` para melhorar o desempenho de pesquisa de cache de atributo de tipo no caso comum de acessos de cache. Isso torna o interpretador 1,04 vezes mais rápido, em média. (Contribuição de Dino Viehland em [bpo-43452](#).)
- As seguintes funções embutidas agora oferecem suporte a uma convenção de chamada de vectorcalls mais rápidos da **PEP 590**: `map()`, `filter()`, `reversed()`, `bool()` e `float()`. (Contribuição de Dong-hee Na e Jeroen Demeyer em [bpo-43575](#), [bpo-43287](#), [bpo-41922](#), [bpo-41873](#) e [bpo-41870](#).)
- O desempenho de `BZ2File` foi melhorado removendo o `RLock` interno. Isso torna `BZ2File` inseguro para threads em face a vários leitores ou gravadores simultâneos, da mesma forma que suas classes equivalentes em `gzip` e `lzma` têm sido. (Contribuição de Inada Naoki em [bpo-43785](#).)

8 Descontinuados

- Atualmente Python aceita literais numéricos imediatamente seguidos por palavras-chave, por exemplo `0in x, 1or x, 0if 1else 2`. Permite expressões confusas e ambíguas como `[0x1for x in y]` (que pode ser interpretada como `[0x1 for x in y]` ou `[0x1f or x in y]`). A partir desta versão, um aviso de descontinuidade é levantado se o literal numérico for seguido imediatamente por uma das palavras-chave `and`, `else`, `for`, `if`, `in`, `is` e `or`. Em versões futuras, ele será alterado para aviso de sintaxe e, finalmente, para erro de sintaxe. (Contribuição de Serhiy Storchaka em [bpo-43833](#).)
- A partir desta versão, haverá um esforço concentrado para começar a limpar a semântica de importação antiga que foi mantida para compatibilidade com Python 2.7. Especificamente, `find_loader()/find_module()` (substituídos por `find_spec()`), `load_module()` (substituído por `exec_module()`), `module_repr()` (que o sistema de importação cuida para você), o atributo `__package__` (substituído por `__spec__.parent`), o atributo `__loader__` (substituído por `__spec__.loader`), e o atributo `__cached__` (substituído por `__spec__.cached`) será removido lentamente (assim como outras classes e métodos em `importlib`). `ImportWarning` e/ou `DeprecationWarning` será levantada conforme apropriado para ajudar a identificar o código que precisa ser atualizado durante esta transição.
- Todo o espaço de nomes de `distutils` foi descontinuado, para ser removido no Python 3.12. Consulte a seção de *alterações do módulo* para mais informações.
- Argumentos não inteiros para `random.randrange()` foram descontinuados. A exceção `ValueError` foi descontinuada em favor de uma exceção `TypeError`. (Contribuição de Serhiy Storchaka e Raymond Hettinger em [bpo-37319](#).)
- Os vários métodos de `load_module()` de `importlib` foram documentados como descontinuados desde Python 3.6, mas agora também irão disparar um `DeprecationWarning`. Use `exec_module()` em vez disso. (Contribuição de Brett Cannon em [bpo-26131](#).)
- `zimport.zipimporter.load_module()` foi descontinuado em preferência a `exec_module()`. (Contribuição de Brett Cannon em [bpo-26131](#).)
- O uso de `load_module()` pelo sistema de importação agora dispara uma exceção `ImportWarning`, pois `exec_module()` é preferível. (Contribuição de Brett Cannon em [bpo-26131](#).)
- O uso de `importlib.abc.MetaPathFinder.find_module()` e `importlib.abc.PathEntryFinder.find_module()` pelo sistema de importação agora dispara uma exceção `ImportWarning`, pois `importlib.abc.MetaPathFinder.find_spec()` e `importlib.abc.PathEntryFinder.find_spec()` são preferidos, respectivamente. Você pode usar `importlib.util.spec_from_loader()` para ajudar no port. (Contribuição de Brett Cannon em [bpo-42134](#).)
- O uso de `importlib.abc.PathEntryFinder.find_loader()` pelo sistema de importação agora dispara uma exceção `ImportWarning`, pois `importlib.abc.PathEntryFinder.find_spec()` é preferido. Você pode usar `importlib.util.spec_from_loader()` para ajudar no port. (Contribuição de Brett Cannon em [bpo-43672](#).)
- As várias implementações de `importlib.abc.MetaPathFinder.find_module()` (`importlib.machinery.BuiltinImporter.find_module()`, `importlib.machinery.FrozenImporter.find_module()`, `importlib.machinery.WindowsRegistryFinder.find_module()`, `importlib.machinery.PathFinder.find_module()`, `importlib.abc.MetaPathFinder.find_module()`), `importlib.abc.PathEntryFinder.find_module()` (`importlib.machinery.FileFinder.find_module()`) e `importlib.abc.PathEntryFinder.find_loader()` (`importlib.machinery.FileFinder.find_loader()`) agora levantam `DeprecationWarning` e estão programadas para remoção no Python 3.12 (anteriormente, eles foram documentados como descontinuados no Python 3.4). (Contribuição de Brett Cannon em [bpo-42135](#).)
- `importlib.abc.Finder` foi descontinuada (incluindo seu único método, `find_module()`). `importlib.abc.MetaPathFinder` e `importlib.abc.PathEntryFinder` não mais herdam

da classe. Usuários devem herdar de uma dessas duas classes conforme apropriado. (Contribuição de Brett Cannon em [bpo-42135](#).)

- As descontinuações de `imp`, `importlib.find_loader()`, `importlib.util.set_package_wrapper()`, `importlib.util.set_loader_wrapper()`, `importlib.util.module_for_loader()`, `pkgutil.ImpImporter` e `pkgutil.ImpLoader` foram todas atualizadas para listar Python 3.12 como a versão programada para remoção (elas começaram a levantar `DeprecationWarning` em versões anteriores do Python). (Contribuição de Brett Cannon em [bpo-43720](#).)
- O sistema de importação agora usa o atributo `__spec__` em módulos antes de recorrer a `module_repr()` para um método `__repr__()` do módulo. A remoção do uso de `module_repr()` está programado para Python 3.12. (Contribuição de Brett Cannon em [bpo-42137](#).)
- `importlib.abc.Loader.module_repr()`, `importlib.machinery.FrozenLoader.module_repr()` e `importlib.machinery.BuiltinLoader.module_repr()` foram descontinuadas e programadas para remoção no Python 3.12. (Contribuição de Brett Cannon em [bpo-42136](#).)
- `sqlite3.OptimizedUnicode` teve a documentação removida e foi descontinuado desde o Python 3.3, quando foi tornado um apelido para `str`. Foi agora descontinuado, programado para remoção no Python 3.12. (Contribuição de Erlend E. Aasland em [bpo-42264](#).)
- A função embutida não documentada `sqlite3.enable_shared_cache` foi agora descontinuada, agendada para remoção no Python 3.12. Seu uso é fortemente desencorajado pela documentação do SQLite3. Veja [a documentação do SQLite3](#) para mais detalhes. Se um cache compartilhado deve ser usado, abra o banco de dados no modo URI usando o parâmetro de consulta `cache=shared`. (Contribuição de Erlend E. Aasland em [bpo-24464](#).)
- Os seguintes métodos de `threading` foram agora descontinuados:
 - `threading.currentThread` => `threading.current_thread()`
 - `threading.activeCount` => `threading.active_count()`
 - `threading.Condition.notifyAll` => `threading.Condition.notify_all()`
 - `threading.Event.isSet` => `threading.Event.is_set()`
 - `threading.Thread.setName` => `threading.Thread.name`
 - `threading.thread.getName` => `threading.Thread.name`
 - `threading.Thread.isDaemon` => `threading.Thread.daemon`
 - `threading.Thread.setDaemon` => `threading.Thread.daemon`(Contribuição de Jelle Zijlstra em [gh-87889](#).)
- `pathlib.Path.link_to()` foi descontinuado e programado para remoção no Python 3.12. Use `pathlib.Path.hardlink_to()` em vez disso. (Contribuição de Barney Gale em [bpo-39950](#).)
- `cgi.log()` foi descontinuado e programado para remoção no Python 3.12. (Contribuição de Inada Naoki em [bpo-41139](#).)
- Os recursos a seguir do `ssl` foram descontinuados desde o Python 3.6, Python 3.7 ou OpenSSL 1.1.0, e serão removidos no 3.11:
 - `OP_NO_SSLv2`, `OP_NO_SSLv3`, `OP_NO_TLSv1`, `OP_NO_TLSv1_1`, `OP_NO_TLSv1_2` e `OP_NO_TLSv1_3` foram substituídos por `sslSSLContext.minimum_version` e `sslSSLContext.maximum_version`.
 - `PROTOCOL_SSLv2`, `PROTOCOL_SSLv3`, `PROTOCOL_SSLv23`, `PROTOCOL_TLSv1`, `PROTOCOL_TLSv1_1`, `PROTOCOL_TLSv1_2` e `PROTOCOL_TLS` foram descontinuados em favor de `PROTOCOL_TLS_CLIENT` e `PROTOCOL_TLS_SERVER`
 - `wrap_socket()` foi substituída por `ssl.SSLContext.wrap_socket()`
 - `match_hostname()`
 - `RAND_pseudo_bytes()`, `RAND_egd()`

- Recurso de NPN como `ssl.SSLSocket.selected_npn_protocol()` e `ssl.SSLContext.set_npn_protocols()` foram substituídos por ALPN.
- A depuração de threads (variável de ambiente `PYTHONTHREADDEBUG`) foi descontinuada no Python 3.10 e será removida no Python 3.12. Este recurso exige uma construção de depuração de Python. (Contribuição de Victor Stinner em [bpo-44584](#).)
- A importação dos submódulos `typing.io` e `typing.re` agora emitirá `DeprecationWarning`. Esses submódulos serão removidos em uma versão futura do Python. Qualquer coisa pertencente a esses submódulos deve ser importada diretamente de `typing`. (Contribuição de Sebastian Rittau em [bpo-38291](#).)

9 Removidos

- Removidos os métodos especiais `__int__`, `__float__`, `__floordiv__`, `__mod__`, `__divmod__`, `__rfloordiv__`, `__rmod__` e `__rdivmod__` da classe `complex`. Eles sempre levantavam uma exceção `TypeError`. (Contribuição de Serhiy Storchaka em [bpo-41974](#).)
- O método `ParserBase.error()` do módulo privado e não documentado `_markupbase` foi removido. `html.parser.HTMLParser` é a única subclasse de `ParserBase` e sua implementação de `error()` já tinha sido removida no Python 3.5. (Contribuição de Berker Peksag em [bpo-31844](#).)
- Removido o atributo `unicodedata.ucnhash_CAPI` que foi um objeto interno `PyCapsule`. A estrutura privada relacionada `_PyUnicode_Name_CAPI` foi movida para uma API C interna. (Contribuição de Victor Stinner em [bpo-42157](#).)
- Removido o módulo `parser`, que foi descontinuado em 3.9 em razão da mudança para o novo analisador sintático GASE, bem como todos os arquivos código-fonte e cabeçalhos C que estavam sendo usados pelo analisador sintático antigo, incluindo `node.h`, `parser.h`, `graminit.h` e `grammar.h`.
- Removidas as funções de API C pública `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags` e `PyNode_Compile` que foram descontinuadas no 3.9 em razão da mudança para o novo analisador sintático GASE.
- Removido o módulo `formatter`, que foi descontinuado no Python 3.4. É um tanto obsoleto, pouco usado e não testado. Ele foi originalmente programado para ser removido no Python 3.6, mas tais remoções foram adiadas até depois do fim de vida do Python 2.7. Os usuários existentes devem copiar quaisquer classes que usam em seu código. (Contribuição de Dong-hee Na e Terry J. Reedy em [bpo-42299](#).)
- Removida a função `PyModule_GetWarningsModule()` que era inútil agora em razão do módulo `_warnings` ser convertido a um módulo embutido no 2.6. (Contribuição de Hai Shi em [bpo-42599](#).)
- Remove apelidos descontinuados para `collections-abstract-base-classes` do módulo `collections`. (Contribuição de Victor Stinner em [bpo-37324](#).)
- O parâmetro `loop` foi removido da maioria da API de alto nível do `asyncio` seguindo a descontinuidade no Python 3.8. A motivação por trás desta alteração é multifacetada:
 1. Isso simplifica a API de alto nível.
 2. As funções na API de alto nível obtêm implicitamente o laço de eventos em execução do thread atual desde o Python 3.7. Não há necessidade de passar o laço de eventos para a API na maioria dos casos de uso normais.
 3. A passagem de laço de eventos está sujeita a erros, especialmente ao lidar com laços em execução em diferentes threads.

Observe que a API de baixo nível ainda aceitará `loop`. Veja [Alterações na API Python](#) para exemplos de como substituir o código existente.

(Contribuição de Yurii Karabas, Andrew Svetlov, Yury Selivanov e Kyle Stanley em [bpo-42392](#).)

10 Portando para Python 3.10

Esta seção lista as alterações descritas anteriormente e outras correções que podem exigir alterações no seu código.

10.1 Alterações na sintaxe Python

- O aviso de descontinuação agora é emitido ao compilar a sintaxe anteriormente válida se o literal numérico for seguido imediatamente por uma palavra reservada (como em `0in x`). Em versões futuras, ele será alterado para aviso de sintaxe e, finalmente, para um erro de sintaxe. Para se livrar do aviso e tornar o código compatível com versões futuras, basta adicionar um espaço entre o literal numérico e a palavra reservada seguinte. (Contribuição de Serhiy Storchaka em [bpo-43833](#).)

10.2 Alterações na API Python

- Os parâmetros *etype* das funções `format_exception()`, `format_exception_only()` e `print_exception()` no módulo `traceback` foram renomeadas para *exc*. (Contribuição de Zackery Spytz e Matthias Bussonnier em [bpo-26389](#).)
- `atexit`: ao sair do Python, se uma função de retorno registrada com `atexit.register()` falhar, sua exceção agora é registrada nos logs. Anteriormente, apenas algumas exceções eram registradas nos logs e a última exceção era sempre ignorada silenciosamente. (Contribuição de Victor Stinner em [bpo-42639](#).)
- A classe genérica `Collections.abc.Callable` agora nivela os parâmetros de tipo, similar ao que `typing.Callable` faz atualmente. Isso significa que `collections.abc.Callable[[int, str], str]` terá `__args__` de `(int, str, str)`; anteriormente era `([int, str], str)`. O código que acessa os argumentos via `typing.get_args()` ou `__args__` precisa levar em conta esta mudança. Além disso, `TypeError` pode ser levantada para formas inválidas de parametrização `Collections.abc.Callable` que pode ter passada silenciosamente no Python 3.9. (Contribuição de Ken Jin em [bpo-42195](#).)
- `socket.htons()` e `socket.ntohs()` agora levantam `OverflowError` em vez de `DeprecationWarning` se o parâmetro dado não couber em um inteiro sem sinal de 16 bits. (Contribuição de Erlend E. Aasland em [bpo-42393](#).)
- O parâmetro `loop` foi removido da maioria da API de alto nível do `asyncio` seguindo a descontinuidade no Python 3.8.

A corrotina que atualmente se parece com isso:

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

Deve ser substituída por isso:

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

Se `foo()` for especificamente projetado para *não* executar no laço de eventos em execução da thread atual (por exemplo, executar no laço de eventos de outra thread), considere usar `asyncio.run_coroutine_threadsafe()`.

(Contribuição de Yurii Karabas, Andrew Svetlov, Yury Selivanov e Kyle Stanley em [bpo-42392](#).)

- O construtor `types.FunctionType` agora herda os embutidos atuais se o dicionário *globals* tiver nenhuma chave `"__builtins__"`, em vez de usar `{"None": None}` como embutidos: o mesmo comportamento que as funções `eval()` e `exec()`. Definir uma função com `def function(...): ...` no Python não é afetado, globais não podem ser substituídos com esta sintaxe: também herda os embutidos atuais. (Contribuição de Victor Stinner em [bpo-42990](#).)

10.3 Alterações na API C

- As funções de API C `PyParser_SimpleParseStringFlags`, `PyParser_SimpleParseStringFlagsFilename`, `PyParser_SimpleParseFileFlags`, `PyNode_Compile` e o tipo usado por essas funções, `struct _node`, foram removidos em razão da mudança para o novo analisador sintático GASE.

O código-fonte deve ser agora compilado diretamente para um objeto código usando, por exemplo, `Py_CompileString()`. O objeto código resultante pode ser então avaliado usando, por exemplo, `PyEval_EvalCode()`.

Especificamente:

- Uma chamada para `PyParser_SimpleParseStringFlags` seguida por `PyNode_Compile` pode ser substituída por `Py_CompileString()`.
- Não há substituição direta para `PyParser_SimpleParseFileFlags`. Para compilar código de um argumento `FILE *`, você vai precisar ler o arquivo em C e passar o buffer resultante para `Py_CompileString()`.
- Para compilar um arquivo de um nome de arquivo `char *` dado, abra explicitamente o arquivo, leia-o e compile o resultado. Uma forma de fazer isso é usando o módulo `io` com `PyImport_ImportModule()`, `PyObject_CallMethod()`, `PyBytes_AsString()` e `Py_CompileString()`, como esboçado abaixo. (Declarações e tratamento de erro foram omitidos.)

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

- Para objetos `FrameObject`, o membro `f_lasti` agora representa um deslocamento de código de palavra em vez de um deslocamento simples na string de bytecode. Isso significa que esse número precisa ser multiplicado por 2 para ser usado com APIs que esperam um deslocamento de byte (como `PyCode_Addr2Line()`, por exemplo). Observe também que o membro `f_lasti` de objetos `FrameObject` não é considerado estável: por favor, use `PyFrame_GetLineNumber()` em vez disso.

11 Alterações de bytecode do CPython

- A instrução `MAKE_FUNCTION` agora aceita um dicionário ou uma tupla de strings como as anotações da função. (Contribuição de Yurii Karabas e Inada Naoki em [bpo-42202](#).)

12 Mudanças na construção

- **PEP 644:** Python agora exige OpenSSL 1.1.1 ou mais novo. OpenSSL 1.0.2 não é mais suportado. (Contribuição de Christian Heimes em [bpo-43669](#).)
- As funções C99 `snprintf()` e `vsnprintf()` agora são exigidas para compilar o Python. (Contribuição de Victor Stinner em [bpo-36020](#).)
- `sqlite3` exige SQLite 3.7.15 ou superior. (Contribuição de Sergey Fedoseev e Erlend E. Aasland em [bpo-40744](#) e [bpo-40810](#).)
- O módulo `atexit` deve agora sempre ser construído como um módulo embutido. (Contribuição de Victor Stinner em [bpo-42639](#).)
- Adiciona a opção `--disable-test-modules` ao script `configure`: não constrói nem instala módulos de teste. (Contribuição de Xavier de Gaye, Thomas Petazzoni e Peixing Xin em [bpo-27640](#).)

- Adiciona a option `--with-wheel-pkg-dir=CAMINHO` ao script `./configure`. Se especificado, o módulo `ensurepip` procura por pacotes `wheel` de `setuptools` e `pip` neste diretório: se ambos estiverem presentes, esses pacotes `wheel` são usados em vez de pacotes `wheel` empacotados por `ensurepip`.

Algumas distribuições Linux possuem políticas de empacotamento recomendando evitar o empacotamento de dependências. Por exemplo, Fedora instala pacotes `wheel` no diretório `/usr/share/python-wheels/` e não instala o pacote `ensurepip._bundled`.

(Contribuição de Victor Stinner em [bpo-42856](#).)

- Adiciona uma nova opção `--without-static-libpython` de `configure` para não construir biblioteca estática `libpythonMAJOR.MINOR.a` e não instala o arquivo objeto `python.o`.

(Contribuição de Victor Stinner em [bpo-43103](#).)

- O script `configure` agora usa o utilitário `pkg-config`, se disponível, para detectar a localização dos cabeçalhos `Tcl/Tk` e bibliotecas. Como antes, esses locais podem ser explicitamente especificados com as opções de configuração `--with-tcltk-includes` e `--with-tcltk-libs`. (Contribuição de Manolis Stamatogiannakis em [bpo-42603](#).)
- Adiciona a opção `--with-openssl-rpath` ao script `configure`. A opção simplifica a construção do Python com uma instalação `OpenSSL` personalizada, por exemplo, `./configure --with-openssl=/path/to/openssl --with-openssl-rpath=auto`. (Contribuição de Christian Heimes em [bpo-43466](#).)

13 Alterações na API C

13.1 PEP 652: Mantendo a ABI estável

A ABI (interface binária de aplicação) Estável para módulos de extensão ou Python embutido agora está explicitamente definido. `stable` descreve as garantias de estabilidade da API C e ABI junto com as melhores práticas para usar a ABI estável.

(Contribuição de Petr Viktorin em [PEP 652](#) e [bpo-43795](#).)

13.2 Novas funcionalidades

- O resultado de `PyNumber_Index()` agora sempre tem o tipo exato `int`. Anteriormente, o resultado poderia ser uma instância de uma subclasse de `int`. (Contribuição de Serhiy Storchaka em [bpo-40792](#).)
- Adiciona um novo membro `orig_argv` à estrutura `PyConfig`: a lista dos argumentos originais da linha de comando passados para o executável Python. (Contribuição de Victor Stinner em [bpo-23427](#).)
- As macros `PyDateTime_DATE_GET_TZINFO()` e `PyDateTime_TIME_GET_TZINFO()` foram adicionadas para acessar os atributos `tzinfo` dos objetos `datetime.datetime` e `datetime.time`. (Contribuição de Zackery Sptz em [bpo-30155](#).)
- Adiciona uma função `PyCodec_Unregister()` para cancelar um registro de uma função de pesquisa de codecs. (Contribuição de Hai Shi em [bpo-41842](#).)
- A função `PyIter_Send()` foi adicionada para permitir o envio de valor para o iterador sem levantar a exceção `StopIteration`. (Contribuição de Vladimir Matveev em [bpo-41756](#).)
- Adiciona `PyUnicode_AsUTF8AndSize()` à API C limitada. (Contribuição de Alex Gaynor em [bpo-41784](#).)
- Adiciona a função `PyModule_AddObjectRef()`: semelhante a `PyModule_AddObject()`, mas não rouba uma referência ao valor em caso de sucesso. (Contribuição de Victor Stinner em [bpo-1635741](#).)
- Adiciona as funções `Py_NewRef()` e `Py_XNewRef()` para incrementar a contagem de referências de um objeto e retornar o objeto. (Contribuição de Victor Stinner em [bpo-42262](#).)

- As funções `PyType_FromSpecWithBases()` e `PyType_FromModuleAndSpec()` agora aceitam uma única classe como o argumento *bases*. (Contribuição de Serhiy Storchaka em [bpo-42423](#).)
 - A função `PyType_FromModuleAndSpec()` agora aceita o slot `NULL tp_doc`. (Contribuição de Hai Shi em [bpo-41832](#).)
 - A função `PyType_GetSlot()` pode aceitar tipos estáticos. (Contribuição de Hai Shi e Petr Viktorin em [bpo-41073](#).)
 - Adiciona uma nova função `PySet_CheckExact()` à C-API para verificar se um objeto é uma instância de `set`, mas não uma instância de um subtipo. (Contribuição de Pablo Galindo em [bpo-43277](#).)
 - Adiciona `PyErr_SetInterruptEx()` que permite passar um número de sinal para simular. (Contribuição de Antoine Pitrou em [bpo-43356](#).)
 - A API C limitada agora é suportada se o Python for compilado no modo de depuração (se a macro `Py_DEBUG` estiver definida). Na API C limitada, as funções `Py_INCREF()` e `Py_DECREF()` agora são implementadas como chamadas de função opacas, em vez de acessar diretamente o membro `PyObject.ob_refcnt`, se o Python for compilado no modo de depuração e a macro `Py_LIMITED_API` tiver como alvo o Python 3.10 ou mais recente. Tornou-se possível suportar a API C limitada no modo de depuração porque a estrutura `PyObject` é a mesma no modo de lançamento e depuração desde o Python 3.8 (consulte [bpo-36465](#)).
- A API C limitada ainda não é suportada na compilação especial `--with-trace-refs` (macro `Py_TRACE_REFS`). (Contribuição de Victor Stinner em [bpo-43688](#).)
- Adiciona a função `Py_Is(x, y)` para testar se o objeto *x* é o objeto *y*, o mesmo que `x is y` em Python. Adiciona também as funções `Py_IsNone()`, `Py_IsTrue()`, `Py_IsFalse()` para testar se um objeto é, respectivamente, o singleton `None`, o singleton `True` ou o singleton `False`. (Contribuição de Victor Stinner em [bpo-43753](#).)
 - Adiciona novas funções para controlar o coletor de lixo do código C: `PyGC_Enable()`, `PyGC_Disable()`, `PyGC_IsEnabled()`. Estas funções permitem ativar, desativar e consultar o estado do coletor de lixo do código C sem precisar importar o módulo `gc`.
 - Adiciona um novo sinalizador de tipo `Py_TPFLAGS_DISALLOW_INSTANTIATION` para impedir a criação de instâncias de tipo. (Contribuição de Victor Stinner em [bpo-43916](#).)
 - Adiciona um novo sinalizador de tipo `Py_TPFLAGS_IMMUTABLETYPE` para criar objetos de tipo imutável: atributos de tipo não podem ser definidos nem excluídos. (Contribuição de Victor Stinner e Erlend E. Aasland em [bpo-43908](#).)

13.3 Portando para Python 3.10

- A macro `PY_SSIZE_T_CLEAN` deve agora ser definida para usar os formatos `PyArg_ParseTuple()` e `Py_BuildValue()` que usam `#`: `es#`, `et#`, `s#`, `u#`, `y#`, `z#`, `U#` e `Z#`. Veja `arg-parsing` e [PEP 353](#). (Contribuição de Victor Stinner em [bpo-40943](#).)
- Como `Py_REFCNT()` é alterado para a função estática em linha, `Py_REFCNT(obj) = new_refcnt` deve ser substituído por `Py_SET_REFCNT(obj, new_refcnt)`: veja `Py_SET_REFCNT()` (disponível desde o Python 3.9). Para compatibilidade com versões anteriores, esta macro pode ser usada:

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

(Contribuição de Victor Stinner em [bpo-39573](#).)

- A chamada de `PyDict_GetItem()` sem a GIL retida era permitido por razões históricas. Isso não é mais permitido. (Contribuição de Victor Stinner em [bpo-40839](#).)
- `PyUnicode_FromUnicode(NULL, size)` e `PyUnicode_FromStringAndSize(NULL, size)` levantam `DeprecationWarning` agora. Use `PyUnicode_New()` para alocar objeto `Unicode` sem dados iniciais. (Contribuição de Inada Naoki em [bpo-36346](#).)

- A estrutura privada `_PyUnicode_Name_CAPI` da API `PyCapsule` `unicodedata.ucnhash_CAPI` foi movida para a API C interna. (Contribuição de Victor Stinner em [bpo-42157](#).)
- As funções `Py_GetPath()`, `Py_GetPrefix()`, `Py_GetExecPrefix()`, `Py_GetProgramFullPath()`, `Py_GetPythonHome()` e `Py_GetProgramName()` agora retornam `NULL` se chamadas antes de `Py_Initialize()` (antes do Python ser inicializado). Use a nova API de init-config para obter a init-path-config. (Contribuição de Victor Stinner em [bpo-42260](#).)
- As macros `PyList_SET_ITEM()`, `PyTuple_SET_ITEM()` e `PyCell_SET()` não podem mais ser usadas como valor-l ou valor-r. Por exemplo, `x = PyList_SET_ITEM(a, b, c)` e `PyList_SET_ITEM(a, b, c) = x` agora falham com um erro do compilador. Isso previne bugs como o teste `if (PyList_SET_ITEM(a, b, c) < 0) ...` (Contribuição de Zackery Spytz e Victor Stinner em [bpo-30459](#).)
- Os arquivos de API não limitada `odictobject.h`, `parser_interface.h`, `picklebufobject.h`, `pyarena.h`, `pyctype.h`, `pydebug.h`, `pyfpe.h` e `pytime.h` foram movidos para o diretório `Include/cpython`. Esses arquivos não devem ser incluídos diretamente, pois já estão incluídos em `Python.h`; veja `api-includes`. Se eles foram incluídos diretamente, considere incluir `Python.h` em vez disso. (Contribuição de Nicholas Sim em [bpo-35134](#).)
- Use o sinalizador de tipo `Py_TPFLAGS_IMMUTABLETYPE` para criar objetos de tipo imutável. Não confie em `Py_TPFLAGS_HEAPTYPE` para decidir se um objeto de tipo é mutável ou não; verifique se `Py_TPFLAGS_IMMUTABLETYPE` está definido. (Contribuição de Victor Stinner e Erlend E. Aasland em [bpo-43908](#).)
- A função não documentada `Py_FrozenMain` foi removida da API limitada. A função é útil principalmente para compilações personalizadas do Python. (Contribuição de Petr Viktorin em [bpo-26241](#).)

13.4 Descontinuados

- A função `PyUnicode_InternImmortal()` agora foi descontinuada e será removida no Python 3.12: use `PyUnicode_InternInPlace()` em vez disso. (Contribuição de Victor Stinner em [bpo-41692](#).)

13.5 Removidos

- Removidas as funções `Py_UNICODE_str*` que manipulam strings `Py_UNICODE*`. (Contribuição de Inada Naoki em [bpo-41123](#).)
 - `Py_UNICODE_strlen`: use `PyUnicode_GetLength()` ou `PyUnicode_GET_LENGTH`
 - `Py_UNICODE_strcat`: use `PyUnicode_CopyCharacters()` ou `PyUnicode_FromFormat()`
 - `Py_UNICODE_strcpy`, `Py_UNICODE_strncpy`: use `PyUnicode_CopyCharacters()` ou `PyUnicode_Substring()`
 - `Py_UNICODE_strcmp`: use `PyUnicode_Compare()`
 - `Py_UNICODE_strncmp`: use `PyUnicode_Tailmatch()`
 - `Py_UNICODE_strchr`, `Py_UNICODE_strrchr`: use `PyUnicode_FindChar()`
- Removida `PyUnicode_GetMax()`. Migre para as novas APIs ([PEP 393](#)). (Contribuição de Inada Naoki em [bpo-41103](#).)
- Removida `PyLong_FromUnicode()`. Migre para `PyLong_FromUnicodeObject()`. (Contribuição de Inada Naoki em [bpo-41103](#).)
- Removida `PyUnicode_AsUnicodeCopy()`. Use `PyUnicode_AsUCS4Copy()` ou `PyUnicode_AsWideCharString()` (Contribuição de Inada Naoki em [bpo-41103](#).)
- Removida a variável `_Py_CheckRecursionLimit`: foi substituída por `ceval.recursion_limit` da estrutura `PyInterpreterState`. (Contribuição de Victor Stinner em [bpo-41834](#).)

- Removidas as macros não documentadas `Py_ALLOW_RECURSION` e `Py_END_ALLOW_RECURSION` e o campo `recursion_critical` da estrutura `PyInterpreterState`. (Contribuição de Serhiy Storcha em [bpo-41936](#).)
- Removida a função `PyOS_InitInterrupts()` não documentada. Inicializar o Python já instala implicitamente manipuladores de sinal: veja `PyConfig.install_signal_handlers`. (Contribuição de Victor Stinner em [bpo-41713](#).)
- Remova a função `PyAST_Validate()`. Não é mais possível construir um objeto AST (tipo `mod_ty`) com a API C pública. A função já foi excluída da API C limitada (**PEP 384**). (Contribuição de Victor Stinner em [bpo-43244](#).)
- Remove o arquivo de cabeçalho `symtable.h` e as funções não documentadas:

- `PyST_GetScope()`
- `PySymtable_Build()`
- `PySymtable_BuildObject()`
- `PySymtable_Free()`
- `Py_SymtableString()`
- `Py_SymtableStringObject()`

A função `Py_SymtableString()` fazia parte da ABI estável por engano, mas não pôde ser usada, porque o arquivo de cabeçalho `symtable.h` foi excluído da API C limitada.

Use o módulo Python `symtable` em vez disso. (Contribuição de Victor Stinner em [bpo-43244](#).)

- Remove `PyOS_ReadlineFunctionPointer()` dos cabeçalhos limitados da API C e de `python3.dll`, a biblioteca que fornece a ABI estável no Windows. Como a função recebe um argumento `FILE*`, sua estabilidade ABI não pode ser garantida. (Contribuição de Petr Viktorin em [bpo-43868](#).)
- Remove os arquivos de cabeçalho `ast.h`, `asdl.h` e `Python-ast.h`. Essas funções não estavam documentadas e excluídas da API C limitada. A maioria dos nomes definidos por esses arquivos de cabeçalho não foram prefixados por `Py`, portanto, podem criar conflitos de nomes. Por exemplo, `Python-ast.h` definiu uma macro `Yield` que estava em conflito com o nome `Yield` usado pelo cabeçalho `<winbase.h>` do Windows. Use o módulo Python `ast` em vez disso. (Contribuição de Victor Stinner em [bpo-43244](#).)
- Remove as funções do compilador e do analisador usando o tipo `struct _mod`, porque a API AST C pública foi removida:

- `PyAST_Compile()`
- `PyAST_CompileEx()`
- `PyAST_CompileObject()`
- `PyFuture_FromAST()`
- `PyFuture_FromASTObject()`
- `PyParser_ASTFromFile()`
- `PyParser_ASTFromFileObject()`
- `PyParser_ASTFromFilename()`
- `PyParser_ASTFromString()`
- `PyParser_ASTFromStringObject()`

Essas funções não estavam documentadas e excluídas da API C limitada. (Contribuição de Victor Stinner em [bpo-43244](#).)

- Remove o arquivo de cabeçalho `pyarena.h` com funções:

- `PyArena_New()`
- `PyArena_Free()`

- `PyArena_Malloc()`
- `PyArena_AddPyObject()`

Essas funções não estavam documentadas, excluídas da API C limitada e eram usadas apenas internamente pelo compilador. (Contribuição de Victor Stinner em [bpo-43244](#).)

- O membro `PyThreadState.use_tracing` foi removido para otimizar o Python. (Contribuição de Mark Shannon em [bpo-43760](#).)

14 Recursos de segurança notáveis no 3.10.7

Converter entre `int` e `str` em bases diferentes de 2 (binário), 4, 8 (octal), 16 (hexadecimal) ou 32 como base 10 (decimal) agora levanta uma exceção `ValueError` se o número de dígitos em forma de string estiver acima de um limite para evitar possíveis ataques de negação de serviço devido à complexidade algorítmica. Esta é uma mitigação para [CVE-2020-10735](#). Este limite pode ser configurado ou desabilitado por variável de ambiente, sinalizador de linha de comando ou APIs de `sys`. Veja a documentação de limitação de comprimento de conversão de string inteira. O limite padrão é de 4300 dígitos em forma de string.

15 Recursos de segurança notáveis no 3.10.8

O módulo descontinuado `mailcap` agora se recusa a injetar texto não seguro (nomes de arquivos, tipos MIME, parâmetros) em comandos shell. Em vez de usar esse texto, ele avisará e agirá como se uma correspondência não fosse encontrada (ou para comandos de teste, como se o teste tivesse falhado). (Contribuição de Petr Viktorin em [gh-98966](#).)

16 Notable Changes in 3.10.12

16.1 tarfile

- Os métodos de extração em `tarfile` e `shutil.unpack_archive()`, têm um novo argumento *filter* que permite limitar recursos do tar que podem ser surpreendentes ou perigosos, como criar arquivos fora do diretório de destino. Veja `tarfile-extraction-filter` para detalhes. No Python 3.12, usar sem o argumento *filter* mostrará um `DeprecationWarning`. No Python 3.14, o padrão mudará para `'data'`. (Contribuição de Petr Viktorin em [PEP 706](#).)

17 Notable changes in 3.10.15

17.1 ipaddress

- Corrigido o comportamento de `is_global` e `is_private` em `IPv4Address`, `IPv6Address`, `IPv4Network` e `IPv6Network`.

17.2 email

- Cabeçalhos com novas linhas embutidas agora são colocados entre aspas na saída.
O `generator` agora se recusará a serializar (escrever) cabeçalhos que são dobrados ou delimitados incorretamente, de modo que eles seriam analisados como vários cabeçalhos ou unidos a dados adjacentes. Se você precisar desativar esse recurso de segurança, defina `verify_generated_headers`. (Contribuição de Bas Bloemsaat e Petr Viktorin em [gh-121650](#).)
- `email.utils.getaddresses()` e `email.utils.parseaddr()` agora retornam tuplas de 2 elementos `(' ', '')` em mais situações onde endereços de e-mail inválidos são encontrados, em vez de valores potencialmente imprecisos. Um parâmetro `strict` opcional foi adicionado a essas duas funções: use `strict=False` para obter o comportamento antigo, aceitando entradas malformadas. `getattr(email.utils, 'supports_strict_parsing', False)` pode ser usado para verificar se o parâmetro `strict` está disponível. (Contribuição de Thomas Dwyer e Victor Stinner para [gh-102988](#) para melhorar a correção de CVE-2023-27043.)

18 Notable changes in 3.10.18

18.1 os.path

- The `strict` parameter to `os.path.realpath()` accepts a new value, `os.path.ALLOW_MISSING`. If used, errors other than `FileNotFoundError` will be re-raised; the resulting path can be missing but it will be free of symlinks. (Contributed by Petr Viktorin for CVE 2025-4517.)

18.2 tarfile

- `data_filter()` now normalizes symbolic link targets in order to avoid path traversal attacks. (Contributed by Petr Viktorin in [gh-127987](#) and CVE 2025-4138.)
- `extractall()` now skips fixing up directory attributes when a directory was removed or replaced by another kind of file. (Contributed by Petr Viktorin in [gh-127987](#) and CVE 2024-12718.)
- `extract()` and `extractall()` now (re-)apply the extraction filter when substituting a link (hard or symbolic) with a copy of another archive member, and when fixing up directory attributes. The former raises a new exception, `LinkFallbackError`. (Contributed by Petr Viktorin for CVE 2025-4330 and CVE 2024-12718.)
- `extract()` and `extractall()` no longer extract rejected members when `errorlevel()` is zero. (Contributed by Matt Prodani and Petr Viktorin in [gh-112887](#) and CVE 2025-4435.)

Índice

P

Propostas Estendidas Python

- PEP 353, [32](#)
- PEP 384, [34](#)
- PEP 393, [33](#)
- PEP 451, [25](#)
- PEP 484, [13](#), [14](#)
- PEP 526, [15](#)
- PEP 586, [24](#)
- PEP 590, [25](#)
- PEP 597, [3](#)
- PEP 604, [3](#), [13](#)
- PEP 612, [3](#), [13](#)
- PEP 613, [3](#), [14](#)
- PEP 617, [4](#)
- PEP 618, [3](#), [14](#)
- PEP 623, [3](#)
- PEP 624, [3](#)
- PEP 626, [3](#)
- PEP 632, [3](#), [18](#)
- PEP 634, [3](#), [12](#)
- PEP 635, [3](#), [12](#)
- PEP 636, [3](#), [12](#)
- PEP 644, [3](#), [19](#), [22](#), [30](#)
- PEP 647, [3](#), [14](#)
- PEP 652, [31](#)
- PEP 706, [35](#)

PYTHONTHREADDEBUG, [28](#)

PYTHONWARNDEFAULTENCODING, [12](#)

R

RFC

- RFC 3986, [24](#)

V

váriavel de ambiente

- PYTHONTHREADDEBUG, [28](#)

- PYTHONWARNDEFAULTENCODING, [12](#)