What's New in Python

Release 3.12.3

A. M. Kuchling

maio 03, 2024

Python Software Foundation
Email: docs@python.org

Sumário

Editor
Adam Turner

Este artigo explica os novos recursos no Python 3.12, em comparação com 3.11. Python 3.12 foi lançado em 2 de outubro de 2023. Veja changelog para uma lista completa de mudanças.

Ver também:

PEP 693 – Agendamento de lançamento do Python 3.12

1 Resumo – Destaques da versão

Python 3.12 é a versão estável mais recente da linguagem de programação Python, com uma combinação de alterações na linguagem e na biblioteca padrão. As alterações da biblioteca se concentram na limpeza de APIs descontinuadas, usabilidade e correção. É importante notar que o pacote `distutils` foi removido da biblioteca padrão. O suporte ao sistema de arquivos em `os` e `pathlib` teve uma série de melhorias e vários módulos têm melhor desempenho.

As mudanças de linguagem se concentram na usabilidade, já que f-strings tiveram muitas limitações removidas e as sugestões 'Did you mean …' continuam a melhorar. A nova *sintaxe de parâmetro type* e a instrução `type` melhoram a ergonomia para usar tipos genéricos e apelidos de tipos com verificadores de tipo estáticos.

Este artigo não tenta fornecer uma especificação completa de todos os novos recursos, mas fornece uma visão geral conveniente. Para detalhes completos, você deve consultar a documentação, como Referência da Biblioteca e Referência da Linguagem. Se você quiser entender a implementação completa e a justificativa do design para uma mudança, consulte a PEP para um novo recurso específico; mas observe que as PEPs geralmente não são mantidas atualizadas depois que um recurso é totalmente implementado.

Novos recursos de sintaxe:

- *PEP 695*, sintaxe de parâmetro de tipo e a instrução `type`

Novos recursos de gramática:

- *PEP 701*, f-strings na gramática

Melhorias no interpretador:

- *PEP 684*, um único GIL por interpretador
- *PEP 669*, monitoramento de baixo impacto
- *Aprimoradas as sugestões 'Did you mean …'* para as exceções `NameError`, `ImportError` e `SyntaxError`

Melhorias no modelo de dados Python:

- *PEP 688*, usando o protocolo de buffer do Python

Melhorias significativas na biblioteca padrão:

- A classe `pathlib.Path` agora oferece suporte a fazer subclasses
- O módulo `os` recebeu vários aprimoramentos para suporte ao Windows
- Uma interface de linha de comando foi adicionada ao módulo `sqlite3`
- As verificações de `isinstance()` para `protocolos verificáveis em tempo de execução` desfruta de um aumento de velocidade entre duas e 20 vezes
- O pacote `asyncio` teve vários aprimoramentos desempenho, com alguns benchmarks mostrando um aumento de 75% na velocidade.
- Uma interface de linha de comando foi adicionada ao módulo `uuid`

- Devido às alterações na *PEP 701*, a produção de tokens por meio do módulo `tokenize` é até 64% mais rápida.

Melhorias de segurança:

- Substitui as implementações embutidas do `hashlib` de SHA1, SHA3, SHA2-384, SHA2-512 e MD5 pelo código formalmente verificado do projeto HACL*. Essas implementações embutidas permanecem como fallbacks que são usados somente quando o OpenSSL não as fornece.

Melhorias na API C:

- *PEP 697*, tier instável de API C
- *PEP 683*, objetos imortal

Melhorias na implementação do CPython:

- *PEP 709*, inlining de compreensão
- Suporte do CPython para o perfilador Linux `perf`
- Implementa proteção contra estouro de pilha em plataformas suportadas

New typing features:

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

Descontinuações, remoções ou restrições importantes:

- **PEP 623**: Remove `wstr` os objetos Unicode na API C do Python, reduzindo o tamanho de cada objeto `str` em pelo menos 8 bytes.
- **PEP 632**: Remove o pacote `distutils`. Consulte o guia de migração para obter conselhos sobre a substituição das APIs fornecidas por ele. O pacote de terceiros Setuptools continua fornecendo `distutils`, se você ainda precisar dele no Python 3.12 e posterior.
- gh-95299: Não pré-instala `setuptools` em ambientes virtuais criados com `venv`. Isso significa que `distutils`, `setuptools`, `pkg_resources` e `easy_install` não estarão mais disponíveis por padrão; para acessá-los, execute `pip install setuptools` no ambiente virtual ativado.
- Os módulos `asynchat`, `asyncore` e `imp` foram removidos, juntamente com vários *apelidos de métodos* de `unittest.TestCase`.

2 Novas funcionalidades

2.1 PEP 695: Sintaxe do parâmetro de tipo

As classes e funções genéricas sob a **PEP 484** foram declaradas usando uma sintaxe detalhada que deixou o escopo dos parâmetros de tipo pouco claro e exigiu declarações explícitas de variação.

PEP 695 apresenta uma maneira nova, mais compacta e explícita de criar classes genéricas e funções:

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

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

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

Além disso, a PEP introduz uma nova maneira de declarar apelidos de tipos usando a instrução `type`, que cria uma instância de `TypeAliasType`:

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

Os apelidos de tipo também podem ser genéricos:

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

A nova sintaxe permite declarar os parâmetros `TypeVarTuple` e `ParamSpec`, bem como os parâmetros `TypeVar` com limites ou restrições:

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

O valor dos apelidos de tipo e os limites e restrições das variáveis de tipo criadas por meio dessa sintaxe são avaliados somente sob demanda (consulte avaliação preguiçosa). Isso significa que os apelidos de tipo podem se referir a outros tipos definidos posteriormente no arquivo.

Os parâmetros de tipo declarados por meio de uma lista de parâmetros de tipo são visíveis no escopo da declaração e em quaisquer escopos aninhados, mas não no escopo externo. Por exemplo, eles podem ser usados nas anotações de tipo para os métodos de uma classe genérica ou no corpo da classe. Entretanto, não podem ser usadas no escopo do módulo depois que a classe é definida. Consulte type-params para obter uma descrição detalhada da semântica de tempo de execução dos parâmetros de tipo.

Para dar suporte a essa semântica de escopo, um novo tipo de escopo é introduzido, o escopo de anotação. Os escopos de anotação se comportam, em sua maior parte, como escopos de função, mas interagem de forma diferente com os escopos de classe. No Python 3.13, anotações também serão avaliadas em escopos de anotação.

Consulte **PEP 695** para obter mais detalhes.

(PEP escrita por Eric Traut. Implementação por Jelle Zijlstra, Eric Traut e outros em gh-103764).

2.2 PEP 701: Formalização sintática de f-strings

PEP 701 resolve algumas restrições no uso de f-strings. Componentes de expressão dentro de f-strings agora podem ser qualquer expressão válida do Python, incluindo strings reutilizando a mesma aspa que a f-string contida, expressões multi-linhas, comentários, barras invertidas e sequências de escape unicode. Vamos cobri-los em detalhes:

- Reuso de aspas: no Python 3.11, reusar as mesmas aspas que a f-string que contém levanta um `SyntaxError`, forçando o usuário a usar outras aspas disponíveis (como usar aspas duplas ou triplas se a f-string usa aspas simples). No Python 3.12, agora você pode fazer coisas como esta:

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

  Observe que, antes dessa alteração, não havia limite explícito de como f-strings podem ser aninhadas, mas o fato de as aspas de string não poderem ser reusadas dentro do componente de expressão de f-strings tornava impossível aninhar f-strings arbitrariamente. Na verdade, esta é a f-string mais aninhada que poderia ser escrita:

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

  Como agora f-strings podem conter qualquer expressão Python válida dentro de componentes de expressão, agora é possível aninhar f-strings arbitrariamente:

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

- Expressões e comentários multilinhas: no Python 3.11, as expressões f-string devem ser definidas em uma única linha, ainda que a expressão dentro de f-string externas possa normalmente abranger várias linhas (como listas literais sendo definidas em várias linhas), tornando-as mais difíceis de ler. No Python 3.12, agora você pode definir f-strings abrangendo várias linhas e adicionar comentários inline:

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

- Contrabarra e caracteres unicode: antes do Python 3.12, as expressões f-string não podiam conter nenhum caractere `\`. Isso também afetou as sequências de escape unicode (como `\N{snowman}`), pois elas contêm a parte `\N` que anteriormente não podia fazer parte dos componentes de expressão de f-strings. Agora, você pode definir expressões como esta:

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

Veja **PEP 701** para mais detalhes.

Como um efeito colateral positivo de como esse recurso foi implementado (analisando f-strings com o **analisador GASE ou PEG**), agora as mensagens de erro para f-strings são mais precisas e incluem o local exato do erro. Por exemplo, no Python 3.11, a seguinte f-string levanta um `SyntaxError`:

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

mas a mensagem de erro não inclui o local exato do erro dentro da linha e também tem a expressão artificialmente cercada por parênteses. No Python 3.12, como as f-strings são analisadas com o analisador GASE, as mensagens de erro podem ser mais precisas e mostrar a linha inteira:

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

(Contribuição de Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou, Cristián Maureira-Fredes e Marta Gómez em gh-102856. PEP escrito por Pablo Galindo, Batuhan Taskaya, Lysandros Nikolaou e Marta Gómez).

2.3 PEP 684: Um GIL por interpretador

A **PEP 684** introduz um GIL por interpretador, para que subinterpretadores possam agora ser criados com um GIL único por interpretador. Isso permite que programas Python aproveitem ao máximo vários núcleos de CPU. Isso está atualmente disponível apenas por meio da API C, embora uma API Python seja **esperada para 3.13**.

Use a nova função `Py_NewInterpreterFromConfig()` para criar um interpretador com seu próprio GIL:

```
PyInterpreterConfig config = {
    .check_multi_interp_extensions = 1,
    .gil = PyInterpreterConfig_OWN_GIL,
};
```

(continua na próxima página)

(continuação da página anterior)

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

Para obter mais exemplos de como usar a API C para subinterpretadores com um GIL por interpretador, consulte Modules/_xxsubinterpretersmodule.c.

(Contribuição de Eric Snow em gh-104210, etc.)

2.4 PEP 669: Monitoramento de baixo impacto para CPython

A **PEP 669** define uma nova `API` para perfis, depuradores e outras ferramentas monitorarem eventos no CPython. Ela abrange uma ampla gama de eventos, incluindo chamadas, retornos, linhas, exceções, saltos e mais. Isso significa que você só paga pelo que usa, fornecendo suporte para depuradores e ferramentas de cobertura com sobrecarga quase zero. Veja `sys.monitoring` para mais detalhes.

(Contribuição de Mark Shannon em gh-103082.)

2.5 PEP 688: Tornando o protocolo de buffer acessível no Python

A **PEP 688** apresenta uma maneira de usar o protocolo de buffer do código Python. As classes que implementam o método `__buffer__()` agora podem ser usadas como tipos de buffer.

O novo `collections.abc.Buffer` ABC fornece uma maneira padrão de representar objetos de buffer, por exemplo, em anotações de tipo. A nova enumeração `inspect.BufferFlags` representa os sinalizadores que podem ser usados para personalizar a criação do buffer. (Contribuição de Jelle Zijlstra em gh-102500.)

2.6 PEP 709: Fazendo inline de compreensão

As compreensões de dicionário, lista e conjunto agora estão embutidas, em vez de criar um novo objeto função de uso único para cada execução da compreensão. Isso acelera a execução de uma compreensão em até duas vezes. Veja **PEP 709** para mais detalhes.

As variáveis de iteração de compreensão permanecem isoladas e não sobrescrevem uma variável de mesmo nome no escopo externo, nem são visíveis após a compreensão. O uso de inlining resulta em algumas mudanças visíveis de comportamento:

- Não há mais um quadro separado para a compreensão em tracebacks (situação da pilha de execução), e o rastreamento/criação de perfil não mostra mais a compreensão como uma chamada de função.
- O módulo `symtable` não produzirá mais tabelas de símbolos filhos para cada compreensão; em vez disso, os locais da compreensão serão incluídos na tabela de símbolos da função pai.
- Chamar `locals()` dentro de uma compreensão agora inclui variáveis de fora da compreensão e não inclui mais a variável sintética `.0` para o "argumento" de compreensão.
- Uma compreensão que itera diretamente sobre `locals()` (por exemplo, `[k for k in locals()]`) pode ver "RuntimeError: dictionary changed size during iteration" quando executada sob rastreamento (por exemplo, medição de cobertura de código). Esse é o mesmo comportamento já observado, por exemplo, em `for k in locals():`. Para evitar o erro, primeiro crie uma lista de chaves para iterar: `keys = list(locals()); [k for k in keys]`.

(Contribuição de Carl Meyer and Vladimir Matveev em **PEP 709**.)

2.7 Mensagens de erro melhoradas

- Os módulos da biblioteca padrão agora são potencialmente sugeridos como parte das mensagens de erro exibidas pelo interpretador quando uma exceção `NameError` é levantada ao nível superior. (Contribuição de Pablo Galindo em gh-98254.)

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

- Melhora a sugestão de erro para exceções `NameError` para instâncias. Agora, se uma exceção `NameError` for levantada em um método e a instância tiver um atributo que é exatamente igual ao nome na exceção, a sugestão incluirá `self.<NOME>` em vez da correspondência mais próxima no escopo do método. (Contribuição de Pablo Galindo em gh-99139.)

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

- Melhora a mensagem de erro de `SyntaxError` quando o usuário digita `import x from y` ao invés de `from y import x`. (Contribuição de Pablo Galindo em gh-98931.)

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

- Exceções `ImportError` levantadas a partir de instruções `from <módulo> import <nome>` com falha agora incluem sugestões para o valor de `<nome>` com base nos nomes disponíveis em `<módulo>`. (Contribuição de Pablo Galindo em gh-91058.)

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

3 Novos recursos relacionados a dicas de tipo

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

3.1 PEP 692: Usando `TypedDict` para tipagem mais precisa de `**kwargs`

A tipagem de `**kwargs` em uma assinatura de função conforme introduzido por **PEP 484** permitia anotações válidas apenas nos casos em que todos os `**kwargs` eram do mesmo tipo.

A **PEP 692** especifica uma maneira mais precisa de definir tipos de `**kwargs`, contando com dicionários tipados:

```
from typing import TypedDict, Unpack

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

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

Veja **PEP 692** para mais detalhes.

(Contribuição de Franek Magiera em gh-103629.)

3.2 PEP 698: Substitui decorador para tipagem estática

Um novo decorador `typing.override()` foi adicionado ao módulo `typing`. Ele indica aos verificadores de tipo que o método se destina a substituir um método em uma superclasse. Isso permite que os verificadores de tipo detectem erros onde um método destinado a substituir algo em uma classe base não o faz de fato.

Exemplo:

```
from typing import override

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

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

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

Consulte **PEP 698** para obter mais detalhes.

(Contribuição de Steven Troxler em gh-101561.)

4 Outras mudanças na linguagem

- O analisador sintático agora exibe `SyntaxError` ao analisar o código-fonte que contém bytes nulos. (Contribuição de Pablo Galindo em gh-96670.)
- Um par de caracteres de barra invertida que não seja uma sequência de escape válida agora levanta uma exceção `SyntaxWarning`, em vez de `DeprecationWarning`. Por exemplo, `re.compile("\d+\.\d+")` agora emite um `SyntaxWarning` (`"\d"` é uma sequência de escape inválida, use strings brutas para expressão regular: `re.compile(r"\d+\.\d+")`). Em uma versão futura do Python, `SyntaxError` acabará sendo levantada, em vez de `SyntaxWarning`. (Contribuição de Victor Stinner em gh-98401.)
- Escapes octais com valor maior do que `0o377` (ex: `"\477"`), descontinuados no Python 3.11, agora produzem um `SyntaxWarning`, em vez de `DeprecationWarning`. Em uma versão futura do Python, eles serão eventualmente um `SyntaxError`. (Contribuição de Victor Stinner em gh-98401.)
- As variáveis usadas na parte de destino das compreensões que não são armazenadas agora podem ser usadas em expressões de atribuição (`:=`). Por exemplo, em `[(b := 1) for a, b.prop in some_iter]`, a atribuição a `b` agora é permitida. Observe que a atribuição a variáveis armazenadas em na parte de destino das compreensões (como `a`) ainda não é permitida, conforme **PEP 572**. (Contribuição de Nikita Sobolev em gh-100581.)
- As exceções levantadas em um método `__set_name__` de uma classe ou um tipo não são mais envolvidas por um `RuntimeError`. As informações de contexto são adicionadas à exceção como uma nota **PEP 678**. (Contribuição de Irit Katriel em gh-77757.)
- Quando uma construção `try-except*` lida com toda a `ExceptionGroup` e levanta uma outra exceção, essa exceção não é mais envolvida em uma `ExceptionGroup`. Também alterada na versão 3.11.4. (Contribuição de Irit Katriel em gh-103590.)
- O coletor de lixo agora é executado somente no mecanismo de quebra de avaliação do laço de avaliação de bytecode do Python em vez de alocações de objetos. O coletor de lixo também pode ser executado quando `PyErr_CheckSignals()` é chamado, de modo que as extensões C que precisam ser executadas por um longo período sem executar nenhum código Python também têm a chance de executar o coletor de lixo periodicamente. (Contribuição de Pablo Galindo em gh-97922.)
- Todos os chamáveis internos e de extensão que esperam parâmetros booleanos agora aceitam argumentos de qualquer tipo, em vez de apenas `bool` e `int`. (Contribuição de Serhiy Storchaka em gh-60203.)
- `memoryview` agora oferece suporte ao tipo meio ponto flutuante (o código de formato “e”). (Contribuição de Donghee Na e Antoine Pitrou em gh-90751.)
- Objetos `slice` agora são hasheáveis, permitindo que sejam usados como chaves de dicionário e itens de conjunto. (Contribuição de Will Bradshaw, Furkan Onder e Raymond Hettinger em gh-101264.)
- `sum()` agora usa a soma de Neumaier para melhorar a precisão e comutatividade ao somar valores flutuantes ou misturas de ints e floats. (Contribuição de Raymond Hettinger em gh-100425.)
- `ast.parse()` agora exibe `SyntaxError` em vez de `ValueError` ao analisar o código-fonte que contém bytes nulos. (Contribuição de Pablo Galindo em gh-96670.)
- Os métodos de extração em `tarfile` e `shutil.unpack_archive()` têm um novo argumento *filter* que permite limitar os recursos do tar que podem ser surpreendentes ou perigosos, como a criação de arquivos fora do diretório de destino. Consulte arquivos de extração de de arquivos tar para obter detalhes. No Python 3.14, o padrão mudará para `'data'`. (Contribuição de Petr Viktorin em **PEP 706**.)
- Instâncias de `types.MappingProxyType` agora são hasheáveis se o mapeamento subjacente for hasheável (Contribuição de Serhiy Storchaka em gh-87995.)
- Adiciona suporte ao perfilador perf por meio da nova variável de ambiente `PYTHONPERFSUPPORT` e da opção de linha de comando `-X perf`, bem como das novas funções `sys.activate_stack_trampoline()`, `sys.deactivate_stack_trampoline()` e `sys.is_stack_trampoline_active()`. (Design de Pablo Galindo. Contribuição de Pablo Galindo e Christian Heimes com contribuições de Gregory P. Smith [Google] e Mark Shannon em gh-96123).

5 Novos módulos

- Nenhum.

6 Módulos melhorados

6.1 array

- A classe `array.array` agora suporta subscrição, tornando-a um tipo genérico. (Contribuição de Jelle Zijlstra em gh-98658.)

6.2 asyncio

- O desempenho da gravação em soquetes no `asyncio` foi significativamente aprimorado. O `asyncio` agora evita cópias desnecessárias ao gravar em soquetes e usa o `sendmsg()` se a plataforma o suportar. (Contribuição de Kumar Aditya em gh-91166.)
- Adiciona as funções `asyncio.eager_task_factory()` e `asyncio.create_eager_task_factory()` para permitir a opção de um laço de eventos na execução de tarefas ansiosas, tornando alguns casos de uso de 2x a 5x mais rápidos. (Contribuição de Jacob Bower e Itamar Oren em gh-102853, gh-104140 e gh-104138)
- No Linux, `asyncio` usa `asyncio.PidfdChildWatcher` por padrão se `os.pidfd_open()` estiver disponível e funcional em vez de `asyncio.ThreadedChildWatcher`. (Contribuição de Kumar Aditya em gh-98024.)
- O laço de eventos agora usa o melhor observador filho disponível para cada plataforma (`asyncio.PidfdChildWatcher` se suportado e `asyncio.ThreadedChildWatcher` caso contrário), portanto, configurar manualmente um observador filho não é recomendado. (Contribuição de Kumar Aditya em gh-94597.)
- Adiciona o parâmetro *loop_factory* a `asyncio.run()` para permitir a especificação de uma fábrica de laço de eventos personalizada. (Contribuição de Kumar Aditya em gh-99388.)
- Adiciona implementação C de `asyncio.current_task()` para aceleração de 4x-6x. (Contribuição de Itamar Oren e Pranav Thulasiram Bhat em gh-100344.)
- `asyncio.iscoroutine()` agora retorna `False` para geradores, pois `asyncio` não tem suporte a corrotinas legadas baseadas em geradores. (Contribuição de Kumar Aditya em gh-102748.)
- `asyncio.wait()` e `asyncio.as_completed()` agora aceita geradores que produzem tarefas. (Contribuição de Kumar Aditya em gh-78530.)

6.3 calendar

- Adiciona enums `calendar.Month` e `calendar.Day` definindo meses do ano e dias da semana. (Contribuição de Prince Roshan em gh-103636.)

6.4 csv

- Adiciona os sinalizadores `csv.QUOTE_NOTNULL` e `csv.QUOTE_STRINGS` para fornecer um controle mais refinado de `None` e strings vazias por objetos `csv.writer`.

6.5 dis

- Opcodes de pseudo-instruções (que são usados pelo compilador, mas não aparecem no bytecode executável) agora são expostos no módulo `dis`. `HAVE_ARGUMENT` ainda é relevante para opcodes reais, mas não é útil para pseudo-instruções. Use a nova coleção `dis.hasarg`. (Contribuição de Irit Katriel em gh-94216.)
- Adiciona a coleção `dis.hasexc` para indicar instruções que definem um manipulador de exceções. (Contribuição de Irit Katriel em gh-94216.)

6.6 fractions

- Objetos do tipo `fractions.Fraction` agora têm suporte a formatação estilo float. (Contribuição de Mark Dickinson em gh-100161.)

6.7 importlib.resources

- `importlib.resources.as_file()` agora tem suporta a diretórios de recursos. (Contribuição de Jason R. Coombs em gh-97930.)
- Renomeia o primeiro parâmetro de `importlib.resources.files()` para *anchor*. (Contribuição de Jason R. Coombs em gh-100598.)

6.8 inspect

- Adiciona `inspect.markcoroutinefunction()` para marcar funções de sincronização que retornam uma corrotina para uso com `inspect.iscoroutinefunction()`. (Contribuição de Carlton Gibson em gh-99247.)
- Adiciona `inspect.getasyncgenstate()` e `inspect.getasyncgenlocals()` para determinar o estado atual dos geradores assíncronos. (Contribuição de Thomas Krennwallner em gh-79940.)
- O desempenho de `inspect.getattr_static()` foi consideravelmente melhorado. A maioria das chamadas para a função deve ser pelo menos 2x mais rápida do que no Python 3.11. (Contribuição de Alex Waygood em gh-103193.)

6.9 itertools

- Adiciona `itertools.batched()` para coletar tuplas de tamanhos iguais, onde o último lote pode ser menor que o resto. (Contribuição de Raymond Hettinger em gh-98363.)

6.10 math

- Adiciona `math.sumprod()` para calcular uma soma de produtos. (Contribuição de Raymond Hettinger em gh-100485.)
- Estende `math.nextafter()` para incluir um argumento *steps* para subir ou descer várias etapas ao mesmo tempo. (Contribuição de Matthias Goergens, Mark Dickinson e Raymond Hettinger em gh-94906.)

6.11 os

- Adiciona `os.PIDFD_NONBLOCK` para abrir um descritor de arquivo para um processo com `os.pidfd_open()` em modo sem bloqueio. (Contribuição de Kumar Aditya em gh-93312.)
- `os.DirEntry` agora inclui um método `os.DirEntry.is_junction()` para verificar se a entrada é uma junção. (Contribuição de Charles Machalow em gh-99547.)
- Adiciona funções `os.listdrives()`, `os.listvolumes()` e `os.listmounts()` no Windows para enumerar unidades, volumes e pontos de montagem. (Contribuição de Steve Dower em gh-102519.)
- `os.stat()` e `os.lstat()` agora são mais precisos no Windows. O campo `st_birthtime` agora será preenchido com a hora de criação do arquivo, e `st_ctime` foi descontinuado, mas ainda contém a hora de criação (mas no futuro vai retornar a última alteração de metadados, para consistência com outras plataformas). `st_dev` pode ter até 64 bits e `st_ino` até 128 bits dependendo do seu sistema de arquivos, e `st_rdev` é sempre definido como zero em vez de valores incorretos. Ambas as funções podem ser significativamente mais rápidas nas lançamentos mais recentes do Windows. (Contribuição de Steve Dower em gh-99726.)

6.12 os.path

- Adiciona `os.path.isjunction()` para verificar se um determinado caminho é uma junção. (Contribuição de Charles Machalow em gh-99547.)
- Adiciona `os.path.splitroot()` para dividir um caminho em uma tríade `(drive, root, tail)`. (Contribuição de Barney Gale em gh-101000.)

6.13 pathlib

- Adiciona suporte para fazer subclasse de `pathlib.PurePath` e `pathlib.Path`, além de suas variantes específicas do Posix e do Windows. As subclasses podem substituir o método `pathlib.PurePath.with_segments()` para passar informações entre instâncias de caminho.
- Adiciona `pathlib.Path.walk()` para percorrer as árvores de diretórios e gerar todos os nomes de arquivos ou diretórios dentro delas, semelhante a `os.walk()`. (Contribuição de Stanislav Zmiev em gh-90385.)
- Adiciona o parâmetro opcional *walk_up* a `pathlib.PurePath.relative_to()` para permitir a inserção de entradas `..` no resultado; este comportamento é mais consistente com `os.path.relpath()`. (Contribuição de Domenico Ragusa em gh-84538.)
- Adiciona `pathlib.Path.is_junction()` como proxy para `os.path.isjunction()`. (Contribuição de Charles Machalow em gh-99547.)
- Adiciona o parâmetro opcional *case_sensitive* a `pathlib.Path.glob()`, `pathlib.Path.rglob()` e `pathlib.PurePath.match()` para fazer correspondência com distinção entre maiúsculas e minúsculas do caminho, permitindo uma precisão mais precisa controle sobre o processo de correspondência.

6.14 pdb

- Adiciona variáveis de conveniência para armazenar valores temporariamente para a sessão de depuração e fornece acesso rápido a valores como o quadro atual ou o valor de retorno. (Contribuição de Tian Gao em gh-103693.)

6.15 random

- Adiciona `random.binomialvariate()`. (Contribuição de Raymond Hettinger em gh-81620.)
- Adiciona um padrão de `lambd=1.0` a `random.expovariate()`. (Contribuição de Raymond Hettinger em gh-100234.)

6.16 shutil

- `shutil.make_archive()` agora passa o argumento *root_dir* para arquivadores personalizados que o suportam. Neste caso, ele não altera mais temporariamente o diretório de trabalho atual do processo para *root_dir* para realizar o arquivamento. (Contribuição de Serhiy Storchaka em gh-74696.)
- `shutil.rmtree()` agora aceita um novo argumento *onexc* que é um tratador de erros como *onerror* mas que espera uma instância de exceção em vez de um trio *(typ, val, tb)*. *onerror* foi descontinuado. (Contribuição de Irit Katriel em gh-102828.)
- `shutil. which()` agora consulta a variável de ambiente *PATHEXT* para encontrar correspondências dentro de *PATH* no Windows mesmo quando o *cmd* fornecido inclui um componente de diretório. (Contribuição de Charles Machalow em gh-103179.)

  `shutil. which()` chamará `NeedCurrentDirectoryForExePathW` ao consultar executáveis no Windows para determinar se o diretório de trabalho atual deve ser anexado ao caminho de pesquisa. (Contribuição de Charles Machalow em gh-103179.)

  `shutil.which()` vai retornar um caminho que corresponde ao *cmd* com um componente de `PATHEXT` antes de uma correspondência direta em outro lugar no caminho de pesquisa no Windows. (Contribuição de Charles Machalow em gh-103179.)

6.17 sqlite3

- Adiciona uma interface de linha de comando. (Contribuição de Erlend E. Aasland em gh-77617.)
- Adiciona o atributo `sqlite3.Connection.autocommit` a `sqlite3.Connection` e o parâmetro *autocommit* a `sqlite3.connect()` para controlar a **PEP 249** compatível com tratamento de transação. (Contribuição de Erlend E. Aasland em gh-83638.)
- Adiciona o parâmetro somente-nomeado *entrypoint* a `sqlite3.Connection.load_extension()`, para substituir o ponto de entrada da extensão SQLite. (Contribuição de Erlend E. Aasland em gh-103015.)
- Adiciona `sqlite3.Connection.getconfig()` e `sqlite3.Connection.setconfig()` a `sqlite3.Connection` para fazer alterações na configuração de uma conexão de banco de dados. (Contribuição de Erlend E. Aasland em gh-103489.)

6.18 statistics

- Estende `statistics.correlation()` para incluir como um método `ranked` para calcular a correlação de Spearman de dados classificados. (Contribuição de Raymond Hettinger em gh-95861.)

6.19 sys

- Adiciona o espaço de nomes `sys.monitoring` para expor a nova API de monitoramento *PEP 669*. (Contribuição de Mark Shannon em gh-103082.)
- Adiciona `sys.activate_stack_trampoline()` e `sys.deactivate_stack_trampoline()` para ativar e desativar trampolins do perfilador de pilha e `sys.is_stack_trampoline_active()` para consultar se os trampolins do perfilador de pilha estão ativos. (Contribuição de Pablo Galindo e Christian Heimes com contribuições de Gregory P. Smith [Google] e Mark Shannon em gh-96123.)
- Adiciona `sys.last_exc` que contém a última exceção não tratada que foi levantada (para casos de uso de depuração post-mortem). Descontinuar os três campos que possuem as mesmas informações em seu formato legado: `sys.last_type`, `sys.last_value` e `sys.last_traceback`. (Contribuição de Irit Katriel em gh-102778.)
- `sys._current_exceptions()` agora retorna um mapeamento de thread-id para uma instância de exceção, ao invés de uma tupla `(typ, exc, tb)`. (Contribuição de Irit Katriel em gh-103176.)
- `sys.setrecursionlimit()` e `sys.getrecursionlimit()`. O limite de recursão agora se aplica apenas ao código Python. As funções integradas não usam o limite de recursão, mas são protegidas por um mecanismo diferente que evita que a recursão cause uma falha na máquina virtual.

6.20 tempfile

- A função `tempfile.NamedTemporaryFile` tem um novo parâmetro opcional *delete_on_close* (Contribuição de Evgeny Zorin em gh-58451.)
- `tempfile.mkdtemp()` agora sempre retorna um caminho absoluto, mesmo se o argumento fornecido para o parâmetro *dir* for um caminho relativo.

6.21 threading

- Adiciona `threading.settrace_all_threads()` e `threading.setprofile_all_threads()` que permitem definir funções de rastreamento e criação de perfil em todos os threads em execução além do que está chamando. (Contribuição de Pablo Galindo em gh-93503.)

6.22 tkinter

- `tkinter.Canvas.coords()` agora nivela seus argumentos. Agora ele aceita não apenas coordenadas como argumentos separados (`x1, y1, x2, y2, ...`) e uma sequência de coordenadas (`[x1, y1, x2, y2, ...]`), mas também coordenadas agrupadas em pares (`(x1, y1), (x2, y2), ...` e `[(x1, y1), (x2, y2), ...]`), como métodos `create_*()`. (Contribuição de Serhiy Storchaka em gh-94473.)

6.23 tokenize

- O módulo `tokenize` inclui as mudanças introduzidas em **PEP 701**. (Contribuição de Marta Gómez Macías e Pablo Galindo em gh-102856.) Veja *Portando para Python 3.12* para mais informações sobre as mudanças no módulo `tokenize`.

6.24 types

- Adiciona `types.get_original_bases()` para permitir uma introspecção adicional de user-defined-generics quando subclassificado. (Contribuição de James Hilton-Balfe e Alex Waygood em gh-101827.)

6.25 typing

- As verificações de `isinstance()` com `protocolos verificáveis em tempo de execução` agora use `inspect.getattr_static()` em vez de `hasattr()` para verificar se existem atributos. Isso significa que descritores e métodos `__getattr__()` não são mais avaliados inesperadamente durante verificações `isinstance()` em protocolos verificáveis em tempo de execução. No entanto, também pode significar que alguns objetos que costumavam ser considerados instâncias de um protocolo verificável em tempo de execução podem não ser mais considerados instâncias desse protocolo no Python 3.12+ e vice-versa. É improvável que a maioria dos usuários seja afetada por esta mudança. (Contribuição de Alex Waygood em gh-102433.)

- Os membros de um protocolo verificável em tempo de execução agora são considerados "congelados" em tempo de execução assim que a classe é criada. Fazer alterações em atributos em tempo de execução em um protocolo verificável em tempo de execução ainda vai funcioanr, mas não terá impacto nas verificações de `isinstance()` comparando objetos com o protocolo. Por exemplo:

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

  Esta mudança foi feita para acelerar as verificações de `isinstance()` em protocolos verificáveis em tempo de execução.

- O perfil de desempenho de verificações de `isinstance()` com `protocolos verificáveis em tempo de execução` mudou significativamente. A maioria das verificações `isinstance()` a esses protocolos com apenas alguns membros devem ser pelo menos 2x mais rápidas que na versão 3.11, e algumas podem ser 20x mais rápidas ou mais. No entanto, as verificações de `isinstance()` em protocolos com muito mais membros podem ser mais lentas do que no Python 3.11. (Contribuição de Alex Waygood em gh-74690 e gh-103193.)

- Todas as classes `typing.TypedDict` e `typing.NamedTuple` agora possuem o atributo `__orig_bases__`. (Contribuição de Adrian Garcia Badaracco em gh-103699.)

- Adiciona o parâmetro `frozen_default` a `typing.dataclass_transform()`. (Contribuição de Erik De Bonte em gh-99957.)

6.26 unicodedata

- O banco de dados Unicode foi atualizado para a versão 15.0.0. (Contribuição de Benjamin Peterson em gh-96734).

6.27 unittest

Adiciona uma opção de linha de comando `--durations`, mostrando os N casos de teste mais lentos:

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

(0.000 durations hidden.  Use -v to show these durations.)
----------------------------------------------------------------------
Ran 158 tests in 9.869s

OK (skipped=3)
```

(Contribuição de Giampaolo Rodola em gh-48330)

6.28 uuid

- Adiciona uma interface de linha de comando. (Contribuição de Adam Chhina em gh-88597.)

7 Otimizações

- Remove os membros `wstr` e `wstr_length` dos objetos Unicode. Reduz o tamanho do objeto em 8 ou 16 bytes na plataforma de 64 bits. (**PEP 623**) (Contribuição de Inada Naoki em gh-92536.)
- Adiciona suporte experimental para usar o otimizador binário BOLT no processo de construção, o que melhora o desempenho de 1 a 5%. (Contribuição de Kevin Modzelewski em gh-90536 e ajuste de Donghee Na em gh-101525)
- Acelera a substituição de expressões regulares (funções `re.sub()` e `re.subn()` e métodos `re.Pattern` correspondentes) para strings de substituição contendo referências de grupo em 2 ou 3 vezes . (Contribuição de Serhiy Storchaka em gh-91524.)
- Acelera a criação de `asyncio.Task` adiando a dispendiosa formatação de strings. (Contribuição de Itamar Oren em gh-103793.)
- As funções `tokenize.tokenize()` e `tokenize.generate_tokens()` são até 64% mais rápidas como efeito colateral das mudanças necessárias para cobrir **PEP 701** no módulo `tokenize` . (Contribuição de Marta Gómez Macías e Pablo Galindo em gh-102856.)
- Acelera chamadas de métodos `super()` e carregamentos de atributos através da nova instrução `LOAD_SUPER_ATTR`. (Contribuição de Carl Meyer e Vladimir Matveev em gh-103497.)

8 Alterações de bytecode do CPython

- Remove a instrução `LOAD_METHOD`. Ela foi mesclada em `LOAD_ATTR`. `LOAD_ATTR` agora se comportará como a antiga instrução `LOAD_METHOD` se o bit inferior de seu oparg estiver definido. (Contribuição de Ken Jin em gh-93429.)
- Remova as instruções `JUMP_IF_FALSE_OR_POP` e `JUMP_IF_TRUE_OR_POP`. (Contribuição de Irit Katriel em gh-102859.)
- Remove a instrução `PRECALL`. (Contribuição de Mark Shannon em gh-92925.)
- Adiciona as instruções `BINARY_SLICE` e `STORE_SLICE`. (Contribuição de Mark Shannon em gh-94163.)
- Adiciona as instruções `CALL_INTRINSIC_1`. (Contribuição de Mark Shannon em gh-99005.)
- Adiciona a instrução `CALL_INTRINSIC_2`. (Contribuição de Irit Katriel em gh-101799.)
- Adiciona a instrução `CLEANUP_THROW`. (Contribuição de Brandt Bucher em gh-90997.)
- Adiciona a instrução `END_SEND`. (Contribuição de Mark Shannon em gh-103082.)
- Adiciona a instrução `LOAD_FAST_AND_CLEAR` como parte da implementação de **PEP 709**. (Contribuição de Carl Meyer em gh-101441.)
- Adiciona a instrução `LOAD_FAST_CHECK`. (Contribuição de Dennis Sweeney em gh-93143.)
- Adiciona os opcodes `LOAD_FROM_DICT_OR_DEREF`, `LOAD_FROM_DICT_OR_GLOBALS` e `LOAD_LOCALS` como parte da implementação da **PEP 695**. Remove o opcode `LOAD_CLASSDEREF`, que pode ser substituído por `LOAD_LOCALS` mais `LOAD_FROM_DICT_OR_DEREF`. (Contribuição de Jelle Zijlstra em gh-103764.)
- Adiciona a instrução `LOAD_SUPER_ATTR`. (Contribuição de Carl Meyer e Vladimir Matveev em gh-103497.)
- Adiciona a instrução `RETURN_CONST`. (Contribuição de Wenyang Wang em gh-101632.)

9 Ferramentas e daemons

- Remove o diretório `Tools/demo/` que continha scripts antigos de demonstração. Uma cópia pode ser encontrada no projeto old-demos. (Contribuição de Victor Stinner em gh-97681.)
- Remove scripts de exemplo desatualizados do diretório `Tools/scripts/`. Uma cópia pode ser encontrada no projeto old-demos. (Contribuição de Victor Stinner em gh-97669.)

10 Descontinuados

- `argparse`: Os parâmetros *type*, *choices* e *metavar* de `argparse.BooleanOptionalAction` foram descontinuados e serão removidos na versão 3.14. (Contribuição de Nikita Sobolev em gh-92248.)
- `ast`: Os seguintes recursos `ast` foram descontinuados na documentação desde Python 3.8, agora fazem com que um `DeprecationWarning` seja emitido em tempo de execução quando eles são acessados ou usados, e serão removidos no Python 3.14:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`

  Usa `ast.Constant` em vez disso. (Contribuição de Serhiy Storchaka em gh-90953.)

- asyncio:
  - As classes filhas do observador asyncio.MultiLoopChildWatcher, asyncio.FastChildWatcher, asyncio.AbstractChildWatcher e asyncio.SafeChildWatcher foram descontinuadas e serão removidas no Python 3.14. (Contribuição de Kumar Aditya em gh-94597.)
  - asyncio.set_child_watcher(), asyncio.get_child_watcher(), asyncio.AbstractEventLoopPolicy.set_child_watcher() e asyncio.AbstractEventLoopPolicy.get_child_watcher() foram descontinuados e serão removidos no Python 3.14. (Contribuição de Kumar Aditya em gh-94597.)
  - O método get_event_loop() da política de laço de eventos padrão agora emite um DeprecationWarning se não houver nenhum laço de eventos atual definido e decidir criar um. (Contribuição de Serhiy Storchaka e Guido van Rossum em gh-100160.)
- calendar: As constantes calendar.January e calendar.February foram descontinuadas e substituídas por calendar.JANUARY e calendar.FEBRUARY. (Contribuição de Prince Roshan em gh-103636.)
- collections.abc: collections.abc.ByteString foi descontinuado. Prefira Sequence ou collections.abc.Buffer. Para uso em tipagem, prefira uma união, como bytes | bytearray ou collections.abc.Buffer. (Contribuição de Shantanu Jain em gh-91896.)
- datetime: utcnow() e utcfromtimestamp() da classe datetime.datetime foram descontinuados e serão removidos em uma versão futura. Em vez disso, use objetos conscientes do fuso horário para representar datas e horas em UTC: respectivamente, chame now() e fromtimestamp() com o parâmetro *tz* definido como datetime.UTC. (Contribuição de Paul Ganssle em gh-103857.)
- email: Descontinua o parâmetro *isdst* em email.utils.localtime(). (Contribuição de Alan Williams em gh-72346.)
- importlib.abc: Foram descontinuadas as seguintes classes, programadas para remoção no Python 3.14:
  - importlib.abc.ResourceReader
  - importlib.abc.Traversable
  - importlib.abc.TraversableResources

  Em vez disso, use classes importlib.resources.abc:
  - importlib.resources.abc.Traversable
  - importlib.resources.abc.TraversableResources

  (Contribuição de Jason R. Coombs e Hugo van Kemenade em gh-93963.)
- itertools: Descontinua o suporte para operações de cópia, deepcopy e pickle, que é não documentado, ineficiente, historicamente cheio de bugs e inconsistente. Isso será removido na versão 3.14 para uma redução significativa no volume de código e na carga de manutenção. (Contribuição de Raymond Hettinger em gh-101588.)
- multiprocessing: No Python 3.14, o método de início padrão de multiprocessing será alterado para um mais seguro no Linux, BSDs e outras plataformas POSIX não-macOS onde 'fork' é atualmente o padrão (gh-84559). Adicionar um aviso em tempo de execução sobre isso foi considerado muito perturbador, pois não se espera que a maior parte do código se importe. Use as APIs get_context() ou set_start_method() para especificar explicitamente quando seu código *requer* 'fork'. Veja contextos e métodos de início.
- pkgutil: pkgutil.find_loader() e pkgutil.get_loader() foram descontinuados e serão removidos no Python 3.14; use importlib.util.find_spec() em vez disso. (Contribuição de Nikita Sobolev em gh-97850.)
- pty: O módulo tem duas funções master_open() e slave_open() não documentadas que foram descontinuados desde Python 2, mas só ganharam um DeprecationWarning adequado em 3.12. Remova-os em 3.14. (Contribuição de Soumendra Ganguly e Gregory P. Smith em gh-85984.)

- os:
  - Os campos st_ctime retornados por os.stat() e os.lstat() no Windows foram descontinuados. Em um lançamento futuro, eles conterão o horário da última alteração de metadados, consistente com outras plataformas. Por enquanto, eles ainda contêm o horário de criação, que também está disponível no novo campo st_birthtime. (Contribuição de Steve Dower em gh-99726.)
  - Nas plataformas POSIX, os.fork() agora pode levantar uma exceção DeprecationWarning quando puder detectar a chamada de um processo multithread. Sempre houve uma incompatibilidade fundamental com a plataforma POSIX ao fazer isso. Mesmo que esse código *pareça* funcionar. Adicionamos o aviso para aumentar a conscientização, pois os problemas encontrados pelo código ao fazer isso estão se tornando mais frequentes. Veja a documentação os.fork() para mais detalhes junto com esta discussão sobre fork ser incompatível com threads para *por que* estamos agora trazendo à tona esse antigo problema de compatibilidade de plataforma para os desenvolvedores.

  Quando este aviso aparece devido ao uso de multiprocessing ou concurrent.futures, a correção é usar um método de início de multiprocessing diferente, como "spawn" ou "forkserver".
- shutil: O argumento *onerror* de shutil.rmtree() foi descontinuado; use *onexc* em vez disso. (Contribuição de Irit Katriel em gh-102828.)
- sqlite3:
  - adaptadores e conversores padrão agora foram descontinuados. Em vez disso, use sqlite3-adapter-converter-recipes e adapte-os às suas necessidades. (Contribuição de Erlend E. Aasland em gh-90016.)
  - Em execute(), DeprecationWarning agora é emitida quando espaços reservados nomeados são usados junto com parâmetros fornecidos como sequência em vez de como dict. A partir do Python 3.14, usar espaços reservados nomeados com parâmetros fornecidos como uma sequência vai levantar uma ProgrammingError. (Contribuição de Erlend E. Aasland em gh-101698.)
- sys: Os campos sys.last_type, sys.last_value e sys.last_traceback foram descontinuados. Use sys.last_exc em vez disso. (Contribuição de Irit Katriel em gh-102778.)
- tarfile: Extrair arquivos tar sem especificar *filter* foi descontinuado até Python 3.14, quando o filtro 'data' se tornará o padrão. Veja tarfile-extraction-filter para detalhes.
- typing:
  - typing.Hashable e typing.Sized, apelidos forem collections.abc.Hashable e collections.abc.Sized respectivamente, foram descontinuados. (gh-94309.)
  - typing.ByteString, descontinuado desde Python 3.9, agora faz com que um DeprecationWarning seja emitida quando é usado. (Contribuição de Alex Waygood em gh-91896.)
- xml.etree.ElementTree: O módulo agora emite DeprecationWarning ao testar o valor verdadeiro de um xml.etree.ElementTree.Element. Antes, a implementação Python emitia FutureWarning, e a implementação C não emitia nada. (Contribuição de Jacob Walls em gh-83122.)
- As assinaturas de 3 argumentos (tipo, valor, traceback) de coroutine throw(), generator throw() e async generator throw() foram descontinuados e podem ser removidos em uma versão futura do Python. Use as versões de argumento único dessas funções. (Contribuição de Ofey Chan em gh-89874.)
- DeprecationWarning agora é levantada quando __package__ em um módulo difere de __spec__.parent (anteriormente era ImportWarning). (Contribuição de Brett Cannon em gh-65961.)
- Definir __package__ ou __cached__ em um módulo foi descontinuado e deixará de ser definido ou levado em consideração pelo sistema de importação no Python 3.14. (Contribuição de Brett Cannon em gh-65961.)
- O operador de inversão bit a bit (~) em bool foi descontinuado. Isso levantará um erro no Python 3.14. Use not para negação lógica de bools. No raro caso em que você realmente precisa da inversão bit a bit do int subjacente, converta para int explicitamente: ~int(x). (Contribuição de Tim Hoffmann em gh-103487.)

- Acessar `co_lnotab` em objetos de código foi descontinuado no Python 3.10 via **PEP 626**, mas só obteve um `DeprecationWarning` adequado em 3.12, portanto será removido em 3.14. (Contribuição de Nikita Sobolev em gh-101866.)

10.1 Remoção pendente no Python 3.13

Os módulos e APIs a seguir foram descontinuados em lançamentos anteriores do Python e serão removidos no Python 3.13.

Módulos (veja **PEP 594**):

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

Outros módulos:

- `lib2to3` e o programa **`2to3`** (gh-84540)

APIs:

- `configparser.LegacyInterpolation` (gh-90765)
- `locale.resetlocale()` (gh-90817)
- `turtle.RawTurtle.settiltangle()` (gh-50096)
- `unittest.findTestCases()` (gh-50096)
- `unittest.getTestCaseNames()` (gh-50096)
- `unittest.makeSuite()` (gh-50096)
- `unittest.TestProgram.usageExit()` (gh-67048)
- `webbrowser.MacOSX` (gh-86421)
- Encadeamento do descritor de `classmethod` (gh-89519)
- Métodos descontinuados de `importlib.resources`:

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

  Use `importlib.resources.files()` em vez disso. Confira importlib-resources: Migrando do legado , em inglês (gh-106531)

10.2 Remoção pendente em Python 3.14

As APIs a seguir foram descontinuadas e serão removidas no Python 3.14.

- `argparse`: Os parâmetros *type*, *choices* e *metavar* de `argparse.BooleanOptionalAction`
- `ast`:
  - `ast.Num`
  - `ast.Str`
  - `ast.Bytes`
  - `ast.NameConstant`
  - `ast.Ellipsis`
- `asyncio`:
  - `asyncio.MultiLoopChildWatcher`
  - `asyncio.FastChildWatcher`
  - `asyncio.AbstractChildWatcher`
  - `asyncio.SafeChildWatcher`
  - `asyncio.set_child_watcher()`
  - `asyncio.get_child_watcher()`,
  - `asyncio.AbstractEventLoopPolicy.set_child_watcher()`
  - `asyncio.AbstractEventLoopPolicy.get_child_watcher()`
- `collections.abc`: `collections.abc.ByteString`.
- `email`: o parâmetro *isdst* em `email.utils.localtime()`.
- `importlib.abc`:
  - `importlib.abc.ResourceReader`
  - `importlib.abc.Traversable`
  - `importlib.abc.TraversableResources`
- `itertools`: Suporte para operações de cópia, cópia profunda (deepcopy) e pickle.
- `pkgutil`:
  - `pkgutil.find_loader()`
  - `pkgutil.get_loader()`.
- `pty`:

  - `pty.master_open()`
  - `pty.slave_open()`
- `shutil`: O argumento *onerror* de `shutil.rmtree()`
- `typing`: `typing.ByteString`
- `xml.etree.ElementTree`: Teste do valor verdadeiro de uma `xml.etree.ElementTree.Element`.
- Os atributos `__package__` e `__cached__` em objetos de módulo.
- O atributo `co_lnotab` de objetos código.

10.3 Remoção pendente em Python 3.15

As APIs a seguir foram descontinuadas e serão removidas no Python 3.15.

APIs:

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

10.4 Remoção pendente em versões futuras

As APIs a seguir foram descontinuadas em versões anteriores do Python e serão removidas, embora atualmente não haja uma data agendada para sua remoção.

- código de formatação `'u'` do `array` (gh-57281)
- `typing.Text` (gh-92332)
- 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]`). Um aviso de sintaxe é levantado se o literal numérico for seguido imediatamente por uma das palavras-chave `and`, `else`, `for`, `if`, `in`, `is` e `or`. Em um lançamento futuro, será alterado para um erro de sintaxe. (gh-87999)

11 Removidos

11.1 asynchat e asyncore

- Esses dois módulos foram removidos de acordo com o cronograma em **PEP 594**, tendo sido descontinuados no Python 3.6. Use `asyncio` em vez disso. (Contribuição de Nikita Sobolev em gh-96580.)

11.2 configparser

- Vários nomes descontinuados no `configparser` na versão 3.2 foram removidos por gh-89336:
  - `configparser.ParsingError` não possui mais um argumento ou atributo `filename`. Use o atributo e o argumento `source`.
  - `configparser` não possui mais uma classe `SafeConfigParser`. Use o nome `ConfigParser` mais curto.
  - `configparser.ConfigParser` não possui mais um método `readfp`. Use `read_file()` em vez disso.

11.3 distutils

- Remove o pacote `distutils`. Foi descontinuado no Python 3.10 por **PEP 632** "Descontinuar o módulo distutils". Para projetos que ainda usam `distutils` e não podem ser atualizados para outra coisa, o projeto `setuptools` pode ser instalado: ele ainda fornece `distutils`. (Contribuição de Victor Stinner em gh-92584.)

11.4 ensurepip

- Remove o wheel de setuptools incluído em `ensurepip` e para de instalar setuptools em ambientes criados por `venv`.

  `pip (>= 22.1)` não requer que setuptools sejam instalados no ambiente. Pacotes baseados em `setuptools` (e baseados em `distutils`) ainda podem ser usados com `pip install`, já que o pip fornecerá `setuptools` no ambiente de construção que ele usa para construir um pacote.

  `easy_install`, `pkg_resources`, `setuptools` e `distutils` não são mais fornecidos por padrão em ambientes criados com `venv` ou inicializados com `ensurepip`, já que eles são parte do pacote `setuptools`. Para projetos que dependem destes em tempo de execução, o projeto `setuptools` deve ser declarado como uma dependência e instalado separadamente (normalmente, usando pip).

  (Contribuição de Pradyun Gedam em gh-95299.)

11.5 enum

- Remove `EnumMeta.__getattr__` de `enum`, que não é mais necessário para acesso ao atributos do enum. (Contribuição de Ethan Furman em gh-95083.)

11.6 ftplib

- Remove o atributo de classe `FTP_TLS.ssl_version` do `ftplib`: use o parâmetro *context* em seu lugar. (Contribuição de Victor Stinner em gh-94172.)

11.7 gzip

- Remove o atributo `filename` do `gzip.GzipFile` de `gzip`, descontinuado desde Python 2.6, use o atributo `name` em seu lugar. No modo de gravação, o atributo `filename` adicionou a extensão de arquivo `'.gz'` se ela não estivesse presente. (Contribuição de Victor Stinner em gh-94196.)

11.8 hashlib

- Remove a implementação Python pura de `hashlib.pbkdf2_hmac()` de `hashlib`, descontinuado em Python 3.10. Python 3.10 e mais recente requer OpenSSL 1.1.1 (**PEP 644**): esta versão do OpenSSL fornece uma implementação C de `pbkdf2_hmac()` que é mais rápida. (Contribuição de Victor Stinner em gh-94199.)

11.9 importlib

- Foi concluída a remoção de muitos alvos de `importlib` que haviam sido descontinuados anteriormente:
  - Referências e suporte a `module_repr()` foram removidos. (Contribuição de Barry Warsaw em gh-97850.)
  - `importlib.util.set_package`, `importlib.util.set_loader` e `importlib.util.module_for_loader` foram todos removidos. (Contribuição de Brett Cannon e Nikita Sobolev em gh-65961 e gh-97850.)
  - O suporte para APIs `find_loader()` e `find_module()` foi removido. (Contribuição de Barry Warsaw em gh-98040.)
  - `importlib.abc.Finder`, `pkgutil.ImpImporter` e `pkgutil.ImpLoader` foram removidos. (Contribuição de Barry Warsaw em gh-98040.)

11.10 imp

- O módulo `imp` foi removido. (Contribuição de Barry Warsaw em gh-98040.)

  Para migrar consulte a seguinte tabela de correspondência:

imp	importlib
`imp.NullImporte`	Insere `None` em `sys.path_importer_cache`
`imp.cache_from_`	`importlib.util.cache_from_source()`
`imp.find_module`	`importlib.util.find_spec()`
`imp.get_magic()`	`importlib.util.MAGIC_NUMBER`
`imp.get_suffixe`	`importlib.machinery.SOURCE_SUFFIXES`, `importlib.machinery.EXTENSION_SUFFIXES` e `importlib.machinery.BYTECODE_SUFFIXES`
`imp.get_tag()`	`sys.implementation.cache_tag`
`imp.load_module`	`importlib.import_module()`
`imp.new_module(`	`types.ModuleType(name)`
`imp.reload()`	`importlib.reload()`
`imp.source_from`	`importlib.util.source_from_cache()`
`imp.load_source`	*Veja abaixo*

  Substitua `imp.load_source()` com:

```
import importlib.util
import importlib.machinery

def load_source(modname, filename):
    loader = importlib.machinery.SourceFileLoader(modname, filename)
    spec = importlib.util.spec_from_file_location(modname, filename,␣
↪loader=loader)
    module = importlib.util.module_from_spec(spec)
```

(continua na próxima página)

(continuação da página anterior)

```
    # The module is always executed and not cached in sys.modules.
    # Uncomment the following line to cache the module.
    # sys.modules[module.__name__] = module
    loader.exec_module(module)
    return module
```

- Remove funções e atributos do `imp` sem substituições:
  - Funções não documentadas:
    * `imp.init_builtin()`
    * `imp.load_compiled()`
    * `imp.load_dynamic()`
    * `imp.load_package()`
  - `imp.lock_held()`, `imp.acquire_lock()`, `imp.release_lock()`: o esquema de bloqueio foi alterado no Python 3.3 para bloqueios por módulo.
  - Constantes de `imp.find_module()`: `SEARCH_ERROR`, `PY_SOURCE`, `PY_COMPILED`, `C_EXTENSION`, `PY_RESOURCE`, `PKG_DIRECTORY`, `C_BUILTIN`, `PY_FROZEN`, `PY_CODERESOURCE`, `IMP_HOOK`.

11.11 io

- Remove `io.OpenWrapper` e `_pyio.OpenWrapper` de `io`, descontinuados em Python 3.10: basta usar `open()` em seu lugar. A função `open()` (`io.open()`) é uma função embutida. Desde o Python 3.10, `_pyio.open()` também é um método estático. (Contribuição de Victor Stinner em gh-94169.)

11.12 locale

- Remove a função `locale.format()` de `locale`, descontinuada no Python 3.7: use `locale.format_string()` em seu lugar. (Contribuição de Victor Stinner em gh-94226.)

11.13 smtpd

- O módulo `smtpd` foi removido de acordo com o cronograma em **PEP 594**, tendo sido descontinuado em Python 3.4.7 e 3.5.4. Use o módulo PyPI aiosmtpd ou qualquer outro servidor baseado em `asyncio`. (Contribuição de Oleg Iarygin em gh-93243.)

11.14 sqlite3

- Os seguintes recursos não documentados do `sqlite3`, descontinuados no Python 3.10, foram agora removidos:
  - `sqlite3.enable_shared_cache()`
  - `sqlite3.OptimizedUnicode`

  Se um cache compartilhado deve ser usado, abra o banco de dados no modo URI usando o parâmetro de consulta `cache=shared`.

  A fábrica de texto `sqlite3.OptimizedUnicode` tem sido um apelido para `str` desde Python 3.3. O código que anteriormente definiu a fábrica de texto como `OptimizedUnicode` pode usar `str` explicitamente ou confiar no valor padrão que também é `str`.

  (Contribuição de Erlend E. Aasland em gh-92548.)

11.15 ssl

- Remove a função `ssl.RAND_pseudo_bytes()` do `ssl`, descontinuada no Python 3.6: use `os.urandom()` ou `ssl.RAND_bytes()` em seu lugar. (Contribuição de Victor Stinner em gh-94199.)
- Remove a função `ssl.match_hostname()`. Foi descontinuada no Python 3.7. OpenSSL realiza correspondência de nome de host desde Python 3.7, Python não usa mais a função `ssl.match_hostname()`. (Contribuição de Victor Stinner em gh-94199.)
- Remove a função `ssl.wrap_socket()`, descontinuada no Python 3.7: em vez desta, crie um objeto `ssl.SSLContext` e chame seu método `ssl.SSLContext.wrap_socket`. Qualquer pacote que ainda use `ssl.wrap_socket()` está quebrado e é inseguro. A função não envia uma extensão SNI TLS nem valida o nome do host do servidor. O código está sujeito a CWE-295 (validação inadequada de certificado). (Contribuição de Victor Stinner em gh-94199.)

11.16 unittest

- Remove recursos do `unittest` há muito tempo descontinuados:
  - Vários apelidos de método de `TestCase`:

Apelido descontinuado	Método	Descontinuado em
`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

    Você pode usar https://github.com/isidentical/teyit para modernizar automaticamente seus testes de unidade.
  - Método não documentado e quebrado `assertDictContainsSubset` de `TestCase` (descontinuado no Python 3.2).
  - Parâmetro não documentado *use_load_tests* de `TestLoader.loadTestsFromModule` (descontinuado e ignorado desde Python 3.2).
  - Um apelido da classe `TextTestResult`: `_TextTestResult` (descontinuado no Python 3.2).

  (Contribuição de Serhiy Storchaka em gh-89325.)

11.17 webbrowser

- Remove o suporte para navegadores descontinuados de `webbrowser`. Os navegadores removidos incluem: Grail, Mosaic, Netscape, Galeon, Skipstone, Iceape, Firebird e Firefox versões 35 e inferiores (gh-102871).

11.18 xml.etree.ElementTree

- Remove o método `ElementTree.Element.copy()` da implementação Python pura, descontinuado no Python 3.10; use a função `copy.copy()` em seu lugar. A implementação C de `xml.etree.ElementTree` não possui nenhum método `copy()`, apenas um método `__copy__()`. (Contribuição de Victor Stinner em gh-94383.)

11.19 zipimport

- Remove os métodos `find_loader()` e `find_module()` do `zipimport`, descontinuados no Python 3.10: use o método `find_spec()` em seu lugar. Veja **PEP 451** para o raciocínio. (Contribuição de Victor Stinner em gh-94379.)

11.20 Outros

- Remove a regra `suspicious` do `Makefile` da documentação e do `Doc/tools/rstlint.py`, ambas em favor do sphinx-lint. (Contribuição de Julien Palard em gh-98179.)
- Remove os parâmetros *keyfile* e *certfile* dos módulos `ftplib`, `imaplib`, `poplib` e `smtplib`, e dos módulos *key_file*, *cert_file* e *check_hostname* do módulo `http.client`, todos descontinuados desde Python 3.6. Use o parâmetro *context* (*ssl_context* em `imaplib`). (Contribuição de Victor Stinner em gh-94172.)
- Remove hacks de compatibilidade `Jython` de vários módulos e testes da stdlib. (Contribuição de Nikita Sobolev em gh-99482.)
- Remove o sinalizador `_use_broken_old_ctypes_structure_semantics_` do módulo `ctypes`. (Contribuição de Nikita Sobolev em gh-99285.)

12 Portando para Python 3.12

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

12.1 Alterações na API Python

- Regras mais rigorosas agora são aplicadas para referências numéricas de grupos e nomes de grupos em expressões regulares. Apenas a sequência de dígitos ASCII agora é aceita como referência numérica. O nome do grupo em padrões de bytes e strings de substituição agora pode conter apenas letras e dígitos ASCII e sublinhado. (Contribuição de Serhiy Storchaka em gh-91760.)
- Remove a funcionalidade `randrange()` descontinuada desde o Python 3.10. Anteriormente, `randrange(10.0)` era convertido sem perdas para `randrange(10)`. Agora, ele levanta um `TypeError`. Além disso, a exceção levantada para valores não inteiros como `randrange(10.5)` ou `randrange('10')` foi alterada de `ValueError` para `TypeError`. Isso também evita bugs onde `randrange(1e25)` selecionaria silenciosamente um intervalo maior que `randrange(10**25)`. (Originalmente sugerido por Serhiy Storchaka gh-86388.)
- `argparse.ArgumentParser` alterou a codificação e o tratador de erros para ler argumentos do arquivo (por exemplo, opção `fromfile_prefix_chars`) da codificação de texto padrão (por exemplo, `locale.getpreferredencoding(False)`) para tratador de erros e codificação do sistema de arquivos. Os arquivos de argumento devem ser codificados em UTF-8 em vez de ANSI Codepage no Windows.

- Remove o módulo `smtpd` baseado em `asyncore` descontinuado no Python 3.4.7 e 3.5.4. Um substituto recomendado é o módulo PyPI baseado em `asyncio` aiosmtpd.
- `shlex.split()`: Passar `None` para o argumento *s* agora levanta uma exceção, ao invés de ler `sys.stdin`. O recurso foi descontinuado no Python 3.9. (Contribuição de Victor Stinner em gh-94352.)
- O módulo `os` não aceita mais caminhos bytes ou similar, como os tipos `bytearray` e `memoryview`: apenas o tipo exato `bytes` é aceito para strings de bytes. (Contribuição de Victor Stinner em gh-98393.)
- `syslog.openlog()` e `syslog.closelog()` agora falham se usadas em subinterpretadores. `syslog.syslog()` ainda pode ser usada em subinterpretadores, mas agora somente se `syslog.openlog()` já tiver sido chamada no interpretador principal. Estas novas restrições não se aplicam ao interpretador principal, então apenas um pequeno grupo de usuários poderá ser afetado. Essa mudança ajuda no isolamento do interpretador. Além disso, `syslog` é um wrapper em torno de recursos globais de processo, que são melhor gerenciados a partir do interpretador principal. (Contribuição de Donghee Na em gh-99127.)
- O comportamento não documentado de bloqueio de `cached_property()` foi removido, porque bloqueava todas as instâncias da classe, levando a uma alta contenção de bloqueio. Isso significa que uma função getter de propriedade em cache agora pode ser executada mais de uma vez para uma única instância, se duas threads competirem. Para a maioria das propriedades simples em cache (por exemplo, aquelas que são idempotentes e simplesmente calculam um valor com base em outros atributos da instância), isso será suficiente. Se a sincronização for necessária, implemente o bloqueio na função getter de propriedade em cache ou em torno de pontos de acesso multithread.
- `sys._current_exceptions()` agora retorna um mapeamento de thread-id para uma instância de exceção, ao invés de uma tupla `(typ, exc, tb)`. (Contribuição de Irit Katriel em gh-103176.)
- Ao extrair arquivos tar usando `tarfile` ou `shutil.unpack_archive()`, passe o argumento *filter* para limitar recursos que podem surpreender ou ser perigosos. Veja tarfile-extraction-filter para detalhes.
- A saída das funções `tokenize.tokenize()` e `tokenize.generate_tokens()` agora foi alterada devido às mudanças introduzidas na **PEP 701**. Isso significa que os tokens `STRING` não são mais emitidos para f-strings e os tokens descritos em **PEP 701** agora são produzidos: `FSTRING_START`, `FSTRING_MIDDLE` e `FSTRING_END` agora são emitidos para partes "string" de f-strings, além dos tokens apropriados para a tokenização nos componentes da expressão. Por exemplo, para a f-string `f"start {1+1} end"` a versão antiga do tokenizer emitida:

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

enquanto a nova versão emite:

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

Além disso, pode haver algumas pequenas alterações comportamentais como consequência das alterações necessárias para oferecer suporte à **PEP 701**. Algumas dessas mudanças incluem:

  - O atributo `type` dos tokens emitidos ao tokenizar alguns caracteres Python inválidos, como `!`, mudou de `ERRORTOKEN` para `OP`.
  - Strings incompletas de linha única agora também levantam `tokenize.TokenError` como fazem strings multilinhas incompletas.
  - Algum código Python incompleto ou inválido agora levanta `tokenize.TokenError` em vez de retornar tokens `ERRORTOKEN` arbitrários ao tokenizá-lo.
  - Misturar tabulações e espaços como indentação no mesmo arquivo não é mais suportado e vai levantar uma `TabError`.

- O módulo `threading` agora espera que o módulo `_thread` tenha um atributo `_is_main_interpreter`. É uma função sem argumentos que retorna `True` se o interpretador atual for o interpretador principal.

  Qualquer biblioteca ou aplicação que forneça um módulo `_thread` personalizado deve fornecer `_is_main_interpreter()`. (Veja gh-112826.)

13 Alterações de construção

- Python não usa mais `setup.py` para construir módulos de extensão C compartilhados. Parâmetros de construção como cabeçalhos e bibliotecas são detectados no script `configure`. As extensões são construídas por `Makefile`. A maioria das extensões usa `pkg-config` e recorre à detecção manual. (Contribuição de Christian Heimes em gh-93939.)
- `va_start()` com dois parâmetros, como `va_start(args, format)`, agora é necessário para construir Python. `va_start()` não é mais chamado com um único parâmetro. (Contribuição de Kumar Aditya em gh-93207.)
- CPython agora usa a opção ThinLTO como política de otimização de tempo de vincualção padrão se o compilador Clang aceitar o sinalizador. (Contribuição de Donghee Na em gh-89536.)
- Adiciona a variável `COMPILEALL_OPTS` em `Makefile` para substituir as opções `compileall` (padrão: `-j0`) em `make install`. Também fundiu os 3 comandos `compileall` em um único comando para construir arquivos .pyc para todos os níveis de otimização (0, 1, 2) de uma só vez. (Contribuição de Victor Stinner em gh-99289.)
- Adiciona trios de plataforma para LoongArch de 64 bits:
  - loongarch64-linux-gnusf
  - loongarch64-linux-gnuf32
  - loongarch64-linux-gnu

  (Contribuição de Zhang Na em gh-90656.)
- `PYTHON_FOR_REGEN` agora requer Python 3.10 ou mais novo.
- Autoconf 2.71 e aclocal 1.16.4 agora são necessários para regerar `!configure`. (Contribuição de Christian Heimes em gh-89886.)
- Construções do Windows e instaladores do macOS em python.org agora usam OpenSSL 3.0.

14 Alterações na API C

14.1 Novas funcionalidades

- **PEP 697**: Introduz o tier de API C Instável, destinada a ferramentas de baixo nível como depuradores e compiladores JIT. Esta API pode mudar em cada versão secundária do CPython sem avisos de descontinuação. Seu conteúdo é marcado pelo prefixo `PyUnstable_` nos nomes.

  Construtores de objeto código:

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

  Armazenamento extra para objetos código (**PEP 523**):

  - `PyUnstable_Eval_RequestCodeExtraIndex()` (renomeado de `_PyEval_RequestCodeExtraIndex`)

  - `PyUnstable_Code_GetExtra()` (renomeado de `_PyCode_GetExtra`)
  - `PyUnstable_Code_SetExtra()` (renomeado de `_PyCode_SetExtra`)

  Os nomes originais continuarão disponíveis até que a respectiva API seja alterada.

  (Contribuição de Petr Viktorin em gh-101101.)

- **PEP 697**: Adiciona uma API para estender tipos cujo layout de memória de instância é opaco:
  - `PyType_Spec.basicsize` pode ser zero ou negativo para especificar a herança ou extensão do tamanho da classe base.
  - `PyObject_GetTypeData()` e `PyType_GetTypeDataSize()` adicionadas para permitir acesso a dados de instância específicos da subclasse.
  - `Py_TPFLAGS_ITEMS_AT_END` e `PyObject_GetItemData()` adicionadas para permitir a extensão segura de certos tipos de tamanho variável, incluindo `PyType_Type`.
  - `Py_RELATIVE_OFFSET` adicionada para permitir a definição de `membros` em termos de uma estrutura específica da subclasse.

  (Contribuição de Petr Viktorin em gh-103509.)

- Adiciona à API C limitada a nova função `PyType_FromMetaclass()`, que generaliza a `PyType_FromModuleAndSpec()` existente usando um argumento de metaclasse adicional. (Contribuição de Wenzel Jakob em gh-93012.)

- API para criação de objetos que podem ser chamados usando o protocolo vectorcall foi adicionada à API Limitada:
  - `Py_TPFLAGS_HAVE_VECTORCALL`
  - `PyVectorcall_NARGS()`
  - `PyVectorcall_Call()`
  - `vectorcallfunc`

  O sinalizador `Py_TPFLAGS_HAVE_VECTORCALL` agora é removido de uma classe quando o método `__call__()` da classe é reatribuído. Isso torna vectorcall seguro para uso com tipos mutáveis (ou seja, tipos de heap sem o sinalizador imutável, `Py_TPFLAGS_IMMUTABLETYPE`). Tipos mutáveis que não substituem `tp_call` agora herdam o sinalizador `Py_TPFLAGS_HAVE_VECTORCALL`. (Contribuição de Petr Viktorin em gh-93274.)

  Os sinalizadores `Py_TPFLAGS_MANAGED_DICT` e `Py_TPFLAGS_MANAGED_WEAKREF` foram adicionados. Isso permite que as classes de extensões ofereçam suporte a objetos `__dict__` e weakrefs com menos trabalho, usando menos memória e com acesso mais rápido.

- API para realizar chamadas usando o protocolo vectorcall foi adicionada à API Limitada:
  - `PyObject_Vectorcall()`
  - `PyObject_VectorcallMethod()`
  - `PY_VECTORCALL_ARGUMENTS_OFFSET`

  Isto significa que ambas as extremidades de entrada e saída do protocolo de chamada vetorial estão agora disponíveis na API Limitada. (Contribuição de Wenzel Jakob em gh-98586.)

- Adiciona duas novas funções públicas, `PyEval_SetProfileAllThreads()` e `PyEval_SetTraceAllThreads()`, que permitem definir funções de rastreamento e perfilação em todos os threads em execução, além do de chamada. (Contribuição de Pablo Galindo em gh-93503.)

- Adiciona a nova função `PyFunction_SetVectorcall()` à API C que define o campo vectorcall de um determinado `PyFunctionObject`. (Contribuição de Andrew Frost em gh-92257.)

- A API C agora permite registrar funções de retorno de chamada via `PyDict_AddWatcher()`, `PyDict_Watch()` e APIs relacionadas a serem chamadas sempre que um dicionário é modificado. Destina-se a ser usado para otimizar interpretadores, compiladores JIT ou depuradores. (Contribuição de Carl Meyer em gh-91052.)

- Adiciona a API `PyType_AddWatcher()` e `PyType_Watch()` para registrar retornos de chamada para receber notificações sobre alterações em um tipo. (Contribuição de Carl Meyer em gh-91051.)
- Adiciona APIs `PyCode_AddWatcher()` e `PyCode_ClearWatcher()` para registrar retornos de chamada para receber notificação sobre criação e destruição de objetos de código. (Contribuição de Itamar Oren em gh-91054.)
- Adiciona as funções `PyFrame_GetVar()` e `PyFrame_GetVarString()` para obter uma variável de quadro por seu nome. (Contribuição de Victor Stinner em gh-91248.)
- Adiciona `PyErr_GetRaisedException()` e `PyErr_SetRaisedException()` para salvar e restaurar a exceção atual. Essas funções retornam e aceitam um único objeto de exceção, em vez dos argumentos triplos dos agora obsoletos `PyErr_Fetch()` e `PyErr_Restore()`. Isso é menos sujeito a erros e um pouco mais eficiente. (Contribuição de Mark Shannon em gh-101578.)
- Adiciona `_PyErr_ChainExceptions1`, que usa uma instância de exceção, para substituir a API legada `_PyErr_ChainExceptions`, que agora foi descontinuada. (Contribuição de Mark Shannon em gh-101578.)
- Adiciona `PyException_GetArgs()` e `PyException_SetArgs()` como funções de conveniência para recuperar e modificar o `args` passado para o construtor da exceção. (Contribuição de Mark Shannon em gh-101578.)
- Adiciona `PyErr_DisplayException()`, que usa uma instância de exceção, para substituir a API legada `PyErr_Display()`. (Contribuição de Irit Katriel em gh-102755).
- **PEP 683**: Introduz *Objetos Imortais*, que permite que objetos ignorem contagens de referências e alterações relacionadas à API C:
  - **`_Py_IMMORTAL_REFCNT`: A contagem de referências que define um objeto** como imortal.
  - `_Py_IsImmortal` Verifica se um objeto possui a contagem de referências imortal.
  - **`PyObject_HEAD_INIT` Isso agora inicializará a contagem de referências para** `_Py_IMMORTAL_REFCNT` quando usado com `Py_BUILD_CORE`.
  - **`SSTATE_INTERNED_IMMORTAL` Um identificador para objetos unicode internalizados** que são imortais.
  - **`SSTATE_INTERNED_IMMORTAL_STATIC` Um identificador para unicode internalizado** objetos que são imortais e estáticos
  - **`sys.getunicodeinternedsize` Isso retorna o número total de unicode** objetos que foram internalizados. Isso agora é necessário para `refleak.py` rastrear corretamente contagens de referências e blocos alocados

  (Contribuição de Eddie Elizondo em gh-84436.)
- **PEP 684**: Adiciona a nova função `Py_NewInterpreterFromConfig()` e `PyInterpreterConfig`, que podem ser usadas para criar sub-interpretadores com seus próprios GILs. (Veja *PEP 684: Um GIL por interpretador* para mais informações.) (Contribuição de Eric Snow em gh-104110.)
- Na API C limitada versão 3.12, as funções `Py_INCREF()` e `Py_DECREF()` agora são implementadas como chamadas de função opacas para ocultar detalhes de implementação. (Contribuição de Victor Stinner em gh-105387.)

14.2 Portando para Python 3.12

- APIs Unicode legadas baseadas na representação de `Py_UNICODE*` foram removidas. Migre para APIs baseadas em UTF-8 ou `wchar_t*`.
- Funções de análise de argumentos como `PyArg_ParseTuple()` não mais oferecem suporte a formato baseado em `Py_UNICODE*` (por exemplo, `u`, `Z`). Migre para outros formatos para Unicode como `s`, `z`, `es` e `U`.
- `tp_weaklist` para todos os tipos embutidos estáticos é sempre `NULL`. Este é um campo somente interno em `PyTypeObject`, mas estamos apontando a mudança caso alguém acesse o campo diretamente de qualquer maneira. Para evitar quebras, considere usar a API C pública existente ou, se necessário, a macro `_PyObject_GET_WEAKREFS_LISTPTR()` (somente interna).
- Este `PyTypeObject.tp_subclasses` somente interno pode agora não ser um ponteiro de objeto válido. Seu tipo foi alterado para `void*` para refletir isso. Mencionamos isso caso alguém esteja acessando diretamente o campo somente interno.

  Para obter uma lista de subclasses, chame o método Python `__subclasses__()` (usando `PyObject_CallMethod()`, por exemplo).
- Adicionado suporte para mais opções de formatação (alinhamento à esquerda, octais, hexadecimais maiúsculos, `intmax_t`, `ptrdiff_t`, `wchar_t` strings C, largura variável e precisão) em `PyUnicode_FromFormat()` e `PyUnicode_FromFormatV()`. (Contribuição de Serhiy Storchaka em gh-98836.)
- Um caractere de formato não reconhecido em `PyUnicode_FromFormat()` e `PyUnicode_FromFormatV()` agora define um `SystemError`. Nas versões anteriores, fazia com que todo o restante da string de formato fosse copiado como estava para a string de resultado e quaisquer argumentos extras descartados. (Contribuição de Serhiy Storchaka em gh-95781.)
- Corrige o posicionamento incorreto do sinal em `PyUnicode_FromFormat()` e `PyUnicode_FromFormatV()`. (Contribuição de Philip Georgi em gh-95504.)
- Classes de extensão que desejam adicionar um `__dict__` ou slot de referência fraca devem usar `Py_TPFLAGS_MANAGED_DICT` e `Py_TPFLAGS_MANAGED_WEAKREF` em vez de `tp_dictoffset` e `tp_weaklistoffset`, respectivamente. O uso de `tp_dictoffset` e `tp_weaklistoffset` ainda é aceitado, mas não oferece suporte totalmente a herança múltipla (gh-95589), e o desempenho pode ser pior. Classes que declaram `Py_TPFLAGS_MANAGED_DICT` devem chamar `_PyObject_VisitManagedDict()` e `_PyObject_ClearManagedDict()` para percorrer e limpar os dicionários de sua instância. Para limpar weakrefs (referências fracas), chame `PyObject_ClearWeakRefs()`, como antes.
- A função `PyUnicode_FSDecoder()` não aceita mais caminhos bytes ou similar, como os tipos `bytearray` e `memoryview`: apenas o tipo exato `bytes` é aceito para strings de bytes. (Contribuição de Victor Stinner em gh-98393.)
- As macros `Py_CLEAR`, `Py_SETREF` e `Py_XSETREF` agora avaliam seus argumentos apenas uma vez. Se um argumento tiver efeitos colaterais, esses efeitos colaterais não serão mais duplicados. (Contribuição de Victor Stinner em gh-98724.)
- O indicador de erro do interpretador agora está sempre normalizado. Isso significa que `PyErr_SetObject()`, `PyErr_SetString()` e as outras funções que definem o indicador de erro agora normalizam a exceção antes de armazená-la. (Contribuição de Mark Shannon em gh-101578.)
- `_Py_RefTotal` não é mais oficial e é mantido apenas para compatibilidade com ABI. Observe que é um global interno e está disponível apenas em compilações de depuração. Se acontecer de você usá-lo, você precisará começar a usar `_Py_GetGlobalRefTotal()`.
- As seguintes funções agora selecionam uma metaclasse apropriada para o tipo recém-criado:
  - `PyType_FromSpec()`
  - `PyType_FromSpecWithBases()`
  - `PyType_FromModuleAndSpec()`

A criação de classes cujas substituições de metaclasse `tp_new` foi descontinuada e no Python 3.14+ não será permitida. Observe que essas funções ignoram `tp_new` da metaclasse, possivelmente permitindo uma inicialização incompleta.

Observe que `PyType_FromMetaclass()` (adicionada em Python 3.12) já não permite a criação de classes cuja metaclasse substitui `tp_new` (`__new__()` em Python).

Como `tp_new` substitui quase tudo que as funções `PyType_From*` fazem, os dois são incompatíveis entre si. O comportamento existente – ignorar a metaclasse em vários passos de criação do tipo – é inseguro em geral, uma vez que as (meta)classes assumem que `tp_new` foi chamado. Não existe uma solução geral simples. Um dos seguintes pode funcionar para você:

- Se você controla a metaclasse, evite usar `tp_new` nela:
  - Se a inicialização puder ser ignorada, isso pode ser feito em `tp_init`.
  - Se a metaclasse não precisa ser instanciada do Python, defina seu `tp_new` como `NULL` usando o sinalizador `Py_TPFLAGS_DISALLOW_INSTANTIATION`. Isso o torna aceitável para funções `PyType_From*`.
- Evite funções `PyType_From*`: se você não precisa de recursos específicos do C (slots ou configuração do tamanho da instância), crie tipos chamando a metaclasse.
- Se você *sabe* que `tp_new` pode ser ignorado com segurança, filtre o aviso de descontinuação usando `warnings.catch_warnings()` do Python.

- `PyOS_InputHook` e `PyOS_ReadlineFunctionPointer` não são mais chamados em subinterpretadores. Isso ocorre porque os clientes geralmente dependem do estado global de todo o processo (uma vez que esses retornos de chamada não têm como recuperar o estado do módulo de extensão).

  Isso também evita situações em que as extensões podem ser executadas em um subinterpretador que não oferecem suporte (ou no qual ainda não foram carregadas). Veja gh-104668 para mais informações.

- `PyLongObject` teve seus componentes internos alterados para melhor desempenho. Embora os internos de `PyLongObject` sejam privados, eles são usados por alguns módulos de extensão. Os campos internos não devem mais ser acessados diretamente, em vez disso, as funções da API que começam com `PyLong_...` devem ser usadas. Duas novas funções de API *instáveis* são fornecidas para acesso eficiente ao valor de `PyLongObject`s que cabe em uma única palavra de máquina:
  - `PyUnstable_Long_IsCompact()`
  - `PyUnstable_Long_CompactValue()`
- Alocadores personalizados, definidos via `PyMem_SetAllocator()`, agora precisam ser seguros para thread, independentemente do domínio de memória. Os alocadores que não possuem estado próprio, incluindo "ganchos", não são afetados. Se o seu alocador personalizado ainda não for seguro para thread e você precisar de orientação, crie um novo relatório de problema no GitHub e CC `@ericsnowcurrently`.

14.3 Descontinuados

- Em conformidade com a **PEP 699**, o campo `ma_version_tag` em `PyDictObject` foi descontinuado para módulos de extensão. Acessar este campo gerará um aviso do compilador em tempo de compilação. Este campo será removido no Python 3.14. (Contribuição de Ramvikrams e Kumar Aditya em gh-101193. PEP de Ken Jin.)
- Variáveis de configuração global descontinuadas:
  - `Py_DebugFlag`: use `PyConfig.parser_debug`
  - `Py_VerboseFlag`: use `PyConfig.verbose`
  - `Py_QuietFlag`: use `PyConfig.quiet`
  - `Py_InteractiveFlag`: use `PyConfig.interactive`
  - `Py_InspectFlag`: use `PyConfig.inspect`

  - `Py_OptimizeFlag`: use `PyConfig.optimization_level`
  - `Py_NoSiteFlag`: use `PyConfig.site_import`
  - `Py_BytesWarningFlag`: use `PyConfig.bytes_warning`
  - `Py_FrozenFlag`: use `PyConfig.pathconfig_warnings`
  - `Py_IgnoreEnvironmentFlag`: use `PyConfig.use_environment`
  - `Py_DontWriteBytecodeFlag`: use `PyConfig.write_bytecode`
  - `Py_NoUserSiteDirectory`: use `PyConfig.user_site_directory`
  - `Py_UnbufferedStdioFlag`: use `PyConfig.buffered_stdio`
  - `Py_HashRandomizationFlag`: use `PyConfig.use_hash_seed` and `PyConfig.hash_seed`
  - `Py_IsolatedFlag`: use `PyConfig.isolated`
  - `Py_LegacyWindowsFSEncodingFlag`: use `PyPreConfig.legacy_windows_fs_encoding`
  - `Py_LegacyWindowsStdioFlag`: use `PyConfig.legacy_windows_stdio`
  - `Py_FileSystemDefaultEncoding`: use `PyConfig.filesystem_encoding`
  - `Py_HasFileSystemDefaultEncoding`: use `PyConfig.filesystem_encoding`
  - `Py_FileSystemDefaultEncodeErrors`: use `PyConfig.filesystem_errors`
  - `Py_UTF8Mode`: use `PyPreConfig.utf8_mode` (see `Py_PreInitialize()`)

  A API `Py_InitializeFromConfig()` deve ser usada com `PyConfig`. (Contribuição de Victor Stinner em gh-77782.)

- A criação de `tipos imutáveis` com bases mutáveis foi descontinuada e será desativada no Python 3.14. (gh-95388)
- O cabeçalho `structmember.h` foi descontinuado, embora continue disponível e não haja planos para removê-lo.

  Seu conteúdo agora está disponível apenas incluindo `Python.h`, com um prefixo `Py` adicionado se estiver faltando:

  - `PyMemberDef`, `PyMember_GetOne()` e `PyMember_SetOne()`
  - Macros de tipo como `Py_T_INT`, `Py_T_DOUBLE`, etc. (anteriormente `T_INT`, `T_DOUBLE`, etc.)
  - Os sinalizadores `Py_READONLY` (anteriormente `READONLY`) e `Py_AUDIT_READ` (anteriormente todas em maiúsculas)

  Vários itens não são expostos em `Python.h`:

  - `T_OBJECT` (use `Py_T_OBJECT_EX`)
  - `T_NONE` (anteriormente não documentada e bastante estranha)
  - A macro `WRITE_RESTRICTED` que não faz nada.
  - As macros `RESTRICTED` e `READ_RESTRICTED`, equivalentes a `Py_AUDIT_READ`.
  - Em algumas configurações, `<stddef.h>` não está incluído em `Python.h`. Deve ser incluído manualmente ao usar `offsetof()`.

  O cabeçalho descontinuado continua a fornecer seu conteúdo original com os nomes originais. Seu código antigo pode permanecer inalterado, a menos que as macros extras e sem namespace o incomodem muito.

  (Contribuição em gh-47146 de Petr Viktorin, baseado em trabalhos anteriores de Alexander Belopolsky e Matthias Braun.)

- `PyErr_Fetch()` e `PyErr_Restore()` foram descontinuado. Use `PyErr_GetRaisedException()` e `PyErr_SetRaisedException()` em vez disso. (Contribuição de Mark Shannon em gh-101578.)
- `PyErr_Display()` foi descontinuada. Use `PyErr_DisplayException()` em vez disso. (Contribuição de Irit Katriel em gh-102755).
- `_PyErr_ChainExceptions` foi descontinuada. Use `_PyErr_ChainExceptions1` em vez disso. (Contribuição de Irit Katriel em gh-102192.)
- Usar `PyType_FromSpec()`, `PyType_FromSpecWithBases()` ou `PyType_FromModuleAndSpec()` para criar uma classe cujas substituições de metaclasse `tp_new` foi descontinuado. Chame a metaclasse.

Remoção pendente em Python 3.14

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

  A API `Py_InitializeFromConfig()` deve ser usada com `PyConfig`.
- A criação de `tipos imutáveis` com bases mutáveis (gh-95388).

Remoção pendente em Python 3.15

- `PyImport_ImportModuleNoBlock()`: use `PyImport_ImportModule()`
- Tipo `Py_UNICODE_WIDE`: use `wchar_t`
- Tipo `Py_UNICODE`: use `wchar_t`
- Funções de inicialização do Python
  - `PySys_ResetWarnOptions()`: limpa `sys.warnoptions` e `warnings.filters`
  - `Py_GetExecPrefix()`: obtém `sys.exec_prefix`
  - `Py_GetPath()`: obtém `sys.path`
  - `Py_GetPrefix()`: obtém `sys.prefix`
  - `Py_GetProgramFullPath()`: obtém `sys.executable`
  - `Py_GetProgramName()`: obtém `sys.executable`
  - `Py_GetPythonHome()`: obtém `PyConfig.home` ou a variável de ambiente `PYTHONHOME`

Remoção pendente em versões futuras

As APIs a seguir foram descontinuado e serão removidas, embora atualmente não haja uma data agendada para sua remoção.

- `Py_TPFLAGS_HAVE_FINALIZE`: desnecessária desde o Python 3.8
- `PyErr_Fetch()`: use `PyErr_GetRaisedException()`
- `PyErr_NormalizeException()`: use `PyErr_GetRaisedException()`
- `PyErr_Restore()`: use `PyErr_SetRaisedException()`
- `PyModule_GetFilename()`: use `PyModule_GetFilenameObject()`
- `PyOS_AfterFork()`: use `PyOS_AfterFork_Child()`
- `PySlice_GetIndicesEx()`: use `PySlice_Unpack()` e `PySlice_AdjustIndices()`
- `PyUnicode_AsDecodedObject()`: use `PyCodec_Decode()`
- `PyUnicode_AsDecodedUnicode()`: use `PyCodec_Decode()`
- `PyUnicode_AsEncodedObject()`: use `PyCodec_Encode()`
- `PyUnicode_AsEncodedUnicode()`: use `PyCodec_Encode()`
- `PyUnicode_READY()`: desnecessário desde o Python 3.12
- `PyErr_Display()`: use `PyErr_DisplayException()`
- `_PyErr_ChainExceptions()`: use `_PyErr_ChainExceptions1`
- Membro `PyBytesObject.ob_shash`: chame `PyObject_Hash()`
- Membro `PyDictObject.ma_version_tag`
- API do Thread Local Storage (TLS):
  - `PyThread_create_key()`: use `PyThread_tss_alloc()`
  - `PyThread_delete_key()`: use `PyThread_tss_free()`
  - `PyThread_set_key_value()`: use `PyThread_tss_set()`
  - `PyThread_get_key_value()`: use `PyThread_tss_get()`
  - `PyThread_delete_key_value()`: use `PyThread_tss_delete()`
  - `PyThread_ReInitTLS()`: desnecessário desde o Python 3.7

14.4 Removidos

- Remove o arquivo de cabeçalho `token.h`. Nunca houve nenhuma API C de tokenizador público. O arquivo de cabeçalho `token.h` foi projetado apenas para ser usado por internos do Python. (Contribuição de Victor Stinner em gh-92651.)
- As APIs Unicode legadas foram removidas. Veja a **PEP 623** para detalhes.
  - `PyUnicode_WCHAR_KIND`
  - `PyUnicode_AS_UNICODE()`
  - `PyUnicode_AsUnicode()`
  - `PyUnicode_AsUnicodeAndSize()`
  - `PyUnicode_AS_DATA()`
  - `PyUnicode_FromUnicode()`
  - `PyUnicode_GET_SIZE()`
  - `PyUnicode_GetSize()`
  - `PyUnicode_GET_DATA_SIZE()`
- Remove a macro de função `PyUnicode_InternImmortal()`. (Contribuição de Victor Stinner em gh-85858.)

15 Alterações notáveis no 3.12.4

15.1 ipaddress

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

Índice

P

V