
What's New in Python

Release 3.13.0

A. M. Kuchling

outubro 29, 2024

Python Software Foundation
Email: docs@python.org

Sumário

1	Resumo – Destaques da versão	3
2	Novas funcionalidades	5
2.1	Um melhor interpretador interativo	5
2.2	Mensagens de erro melhoradas	5
2.3	CPython com threads livres	6
2.4	Um compilador just-in-time (JIT) experimental	7
2.5	Definidas semânticas de mutação para <code>locals()</code>	8
2.6	Suporte para plataformas móveis	8
3	Outras mudanças na linguagem	9
4	Novos módulos	10
5	Módulos melhorados	10
5.1	<code>argparse</code>	10
5.2	<code>array</code>	10
5.3	<code>ast</code>	10
5.4	<code>asyncio</code>	11
5.5	<code>base64</code>	11
5.6	<code>compileall</code>	11
5.7	<code>concurrent.futures</code>	11
5.8	<code>configparser</code>	12
5.9	<code>copy</code>	12
5.10	<code>ctypes</code>	12
5.11	<code>dbm</code>	12
5.12	<code>dis</code>	12
5.13	<code>doctest</code>	13
5.14	<code>email</code>	13
5.15	<code>fractions</code>	13
5.16	<code>glob</code>	13
5.17	<code>importlib</code>	13
5.18	<code>io</code>	14
5.19	<code>ipaddress</code>	14
5.20	<code>itertools</code>	14
5.21	<code>marshal</code>	14
5.22	<code>math</code>	14
5.23	<code>mimetypes</code>	14

5.24	mmap	14
5.25	multiprocessing	14
5.26	os	15
5.27	os.path	15
5.28	pathlib	15
5.29	pdb	16
5.30	queue	16
5.31	random	16
5.32	re	16
5.33	shutil	16
5.34	site	16
5.35	sqlite3	16
5.36	ssl	16
5.37	statistics	17
5.38	subprocess	17
5.39	sys	17
5.40	tempfile	17
5.41	time	17
5.42	tkinter	18
5.43	traceback	18
5.44	types	18
5.45	typing	18
5.46	unicodedata	19
5.47	venv	19
5.48	warnings	19
5.49	xml	19
5.50	zipimport	19
6	Otimizações	19
7	Módulos e APIs removidas	20
7.1	PEP 594: remove “baterias descarregadas” da biblioteca padrão	20
7.2	2to3	21
7.3	embutidos	21
7.4	configparser	21
7.5	importlib.metadata	21
7.6	locale	21
7.7	opcode	21
7.8	pathlib	21
7.9	re	22
7.10	tkinter.tix	22
7.11	turtle	22
7.12	typing	22
7.13	unittest	22
7.14	urllib	22
7.15	webbrowser	22
8	Novas descontinuações	23
8.1	Remoção pendente no Python 3.14	25
8.2	Remoção pendente no Python 3.15	26
8.3	Remoção pendente no Python 3.16	27
8.4	Remoção pendente em versões futuras	28
9	Alterações de bytecode do CPython	30
10	Alterações na API C	30
10.1	Novas funcionalidades	30
10.2	Alterações na API C	33
10.3	Mudanças na API C Limitada	33

10.4 APIs C removidas	34
10.5 APIs C descontinuadas	35
11 Mudanças na construção	38
12 Portando para o Python 3.13	39
12.1 Alterações na API Python	39
12.2 Alterações na API C	39
13 Mudanças em teste de regressão	41
Índice	42

Editores

Adam Turner e Thomas Wouters

Este artigo explica os novos recursos no Python 3.13, em comparação com 3.12. Python 3.13 foi lançado em 7 de outubro de 2024. Veja changelog para uma lista completa de mudanças.

Ver também

PEP 719 – Cronograma de lançamento do Python 3.13

1 Resumo – Destaques da versão

O Python 3.13 é a versão estável mais recente da linguagem de programação Python, com uma mistura de mudanças na linguagem, na implementação e na biblioteca padrão. As maiores mudanças incluem um novo *interpretador interativo*, suporte experimental para execução em um *modo de threads livres* (**PEP 703**) e um *compilador Just-In-Time* (**PEP 744**).

As mensagens de erro continuam a melhorar, com tracebacks (situação da pilha de execução) agora realçados em cores por padrão. A função embutida `locals()` agora tem *semânticas definidas* para alterar o mapeamento retornado, e os parâmetros de tipo agora oferecem suporte a valores padrão.

As alterações da biblioteca contêm a remoção de APIs e módulos descontinuados, bem como as melhorias usuais em facilidade de uso e correção. Vários módulos da biblioteca padrão legados agora *foram removidos* após sua descontinuação no Python 3.11 (**PEP 594**).

Este artigo não tenta fornecer uma especificação completa de todos os novos recursos, mas fornece uma visão geral conveniente. Para detalhes completos, consulte a documentação, como Referência da Biblioteca e Referência da Linguagem. Para 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. Veja *Portando para o Python 3.13* para orientação sobre atualização a partir de versões anteriores do Python.

Melhorias no interpretador:

- Um *interpretador interativo* bastante aprimorado e *mensagens de erro aprimoradas*.
- **PEP 667**: a função embutida `locals()` agora tem *semântica definida* ao fazer a mutação do mapeamento retornado. Os depuradores do Python e ferramentas semelhantes agora podem atualizar variáveis locais de maneira mais confiável em escopos otimizados, mesmo durante a execução simultânea de código.
- **PEP 703**: CPython 3.13 tem suporte experimental para ser executado com a trava global do interpretador, ou GIL, desabilitada. Veja *CPython com threads livres* para mais detalhes.

- **PEP 744**: um *compilador JIT* básico foi adicionado. Atualmente está desativado por padrão (embora possamos ativá-lo mais tarde). As melhorias de desempenho são modestas – esperamos melhorar isso nas próximas versões.
- Suporte a cores no novo *interpretador interativo*, bem como na saída de *tracebacks* e de *doctest*. Isso pode ser desabilitado através das variáveis de ambiente `PYTHON_COLORS` e `NO_COLOR`.

Melhorias no modelo de dados Python:

- `__static_attributes__` armazena os nomes dos atributos acessados por meio de `self.X` em qualquer função no corpo da classe.
- `__firstlineno__` registra o número da primeira linha da definição de classe.

Melhorias significativas na biblioteca padrão:

- Adiciona uma nova exceção `PythonFinalizationError`, levantada quando uma operação é bloqueada durante a finalização.
- O módulo `argparse` agora oferece suporte a descontinuar opções de linha de comando, argumentos posicionais e subcomandos.
- As novas funções `base64.z85encode()` e `base64.z85decode()` oferecem suporte à codificação e decodificação de *dados Z85*.
- O módulo `copy` agora tem uma função `copy.replace()`, com suporte para muitos tipos embutidos e qualquer classe que defina o método `__replace__()`.
- O novo módulo `dbm.sqlite3` agora é o backend `dbm` padrão.
- O módulo `os` tem um conjunto de novas funções para trabalhar com os descritores de arquivo de notificação de temporizador do Linux.
- O módulo `random` agora tem uma interface de linha de comando.

Melhorias de segurança:

- `ssl.create_default_context()` define `ssl.VERIFY_X509_PARTIAL_CHAIN` e `ssl.VERIFY_X509_STRICT` como sinalizadores padrão.

Melhorias na API C:

- O slot `Py_mod_gil` agora é usado para indicar que um módulo de extensão oferece suporte à execução com a GIL desabilitada.
- A API C `PyTime` foi adicionada, fornecendo acesso aos relógios do sistema.
- `PyMutex` é um novo mutex leve que ocupa um único byte.
- Há um novo conjunto de funções para gerar eventos de monitoramento da **PEP 669** na API C.

Novos recursos de tipagem:

- **PEP 696**: parâmetros de tipo (`typing.TypeVar`, `typing.ParamSpec` e `typing.TypeVarTuple`) agora oferecem suporte a valores padrão.
- **PEP 702**: o novo decorador `warnings.deprecated()` adiciona suporte para marcar descontinuações no sistema de tipos e ambiente de execução.
- **PEP 705**: `typing.ReadOnly` pode ser usado para marcar um item de `typing.TypedDict` como somente leitura para verificadores de tipo.
- **PEP 742**: `typing.TypeIs` fornece um comportamento de estreitamento de tipo mais intuitivo, como uma alternativa ao `typing.TypeGuard`.

Suporte à plataforma:

- **PEP 730**: iOS da Apple agora é uma *plataforma oficialmente com suporte*, no **tier 3**.
- **PEP 738**: Android é agora uma *plataforma oficialmente com suporte*, no **tier 3**.
- `wasm32-wasi` agora é uma plataforma com suporte **tier 2**.

- `wasm32-emscripten` não é mais uma plataforma oficialmente com suporte.

Remoções importantes:

- **PEP 594**: as 19 “baterias descarregadas” (módulos legados da biblioteca padrão) restantes foram removidas da biblioteca padrão: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `nntplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` e `xdrlib`.
- Remove a ferramenta `2to3` e o módulo `lib2to3` (descontinuados no Python 3.11).
- Remove o módulo `tkinter.tix` (descontinuado no Python 3.6).
- Remove a função `locale.resetlocale()`.
- Remove os espaços de nomes `typing.io` e `typing.re`.
- Remove descritores encadeados de `classmethod`.

Mudanças no cronograma de lançamento:

PEP 602 (“Ciclo de Lançamento Anual para Python”) foi atualizado para estender o período de suporte total (“bug-fix”) para novos lançamentos para dois anos. Esta política atualizada significa que:

- O Python 3.9–3.12 tem um ano e meio de suporte total, seguido por três anos e meio de correções de segurança.
- Python 3.13 e posteriores têm dois anos de suporte total, seguidos de três anos de correções de segurança.

2 Novas funcionalidades

2.1 Um melhor interpretador interativo

Python agora usa um novo console interativo por padrão, com base no código do [projeto PyPy](#). Quando o usuário inicia o REPL de um terminal interativo, agora os seguintes novos recursos estão disponíveis:

- Edição multilinha com preservação do histórico.
- Suporte direto para comandos específicos do REPL, como `help`, `exit` e `quit`, sem a necessidade de chamá-los como funções.
- Prompts e tracebacks com cores habilitadas por padrão.
- Ajuda interativa para navegar usando `F1` com um histórico de comandos separado.
- Navegação no histórico usando `F2` que ignora a saída, bem como os prompts `>>>` e `....`.
- “Modo de colagem” com `F3` que facilita colar blocos de código maiores (pressione `F3` novamente para retornar ao prompt normal).

Para desabilitar o novo console interativo, defina a variável de ambiente `PYTHON_BASIC_REPL`. Para mais informações sobre o modo interativo, veja [tut-interac](#).

(Contribuição de Pablo Galindo Salgado, Łukasz Langa e Lysandros Nikolaou em [gh-111201](#) baseado no código do projeto PyPy. Suporte ao Windows foi uma contribuição de Dino Viehland e Anthony Shaw.)

2.2 Mensagens de erro melhoradas

- O interpretador agora usa cores por padrão ao exibir tracebacks no terminal. Este recurso pode ser controlado por meio da nova variável de ambiente `PYTHON_COLORS`, bem como das variáveis de ambiente canônicas `NO_COLOR` e `FORCE_COLOR`. (Contribuição de Pablo Galindo Salgado em [gh-112730](#).)
- Um erro comum é escrever um script com o mesmo nome de um módulo de biblioteca padrão. Quando isso resulta em erros, agora exibimos uma mensagem de erro mais útil:

```
$ python random.py
Traceback (most recent call last):
  File "/home/me/random.py", line 1, in <module>
    import random
```

(continua na próxima página)

```
File "/home/me/random.py", line 3, in <module>
    print(random.randint(5))
    ^^^^^^^^^^^^^^^^^
AttributeError: module 'random' has no attribute 'randint' (consider renaming
→ '/home/me/random.py' since it has the same name as the standard library_
→ module named 'random' and prevents importing that standard library module)
```

Da mesma forma, se um script tiver o mesmo nome de um módulo de terceiros que ele tentar importar e isso resultar em erros, também exibiremos uma mensagem de erro mais útil:

```
$ python numpy.py
Traceback (most recent call last):
  File "/home/me/numpy.py", line 1, in <module>
    import numpy as np
  File "/home/me/numpy.py", line 3, in <module>
    np.array([1, 2, 3])
    ^^^^^^^
AttributeError: module 'numpy' has no attribute 'array' (consider renaming '/'
→ home/me/numpy.py' if it has the same name as a library you intended to_
→ import)
```

(Contribuição de Shantanu Jain em [gh-95754](#).)

- A mensagem de erro agora tenta sugerir o argumento nomeado correto quando um argumento nomeado incorreto é passado para uma função.

```
>>> "Better error messages!".split(max_split=1)
Traceback (most recent call last):
  File "<python-input-0>", line 1, in <module>
    "Better error messages!".split(max_split=1)
    ~~~~~^~~~~~
TypeError: split() got an unexpected keyword argument 'max_split'. Did you_
→ mean 'maxsplit'?
```

(Contribuição de Pablo Galindo Salgado e Shantanu Jain em [gh-107944](#).)

2.3 CPython com threads livres

O CPython agora tem suporte experimental para execução em modo de threads livres, com a trava global do interpretador (GIL) desabilitada. Este é um recurso experimental e, portanto, não é habilitado por padrão. O modo de threads livres requer um executável diferente, geralmente chamado de `python3.13t` ou `python3.13t.exe`. Binários pré-construídos marcados como *free-threaded* podem ser instalados como parte dos instaladores oficiais Windows e macOS, ou o CPython pode ser construído a partir da fonte com a opção `--disable-gil`.

A execução de threads livres permite a utilização total do poder de processamento disponível ao executar threads em paralelo nos núcleos de CPU disponíveis. Embora nem todos os softwares se beneficiem disso automaticamente, programas projetados com threading em mente serão executados mais rapidamente em hardware multi-core. **O modo de threads livres é experimental** e o trabalho está em andamento para melhorá-lo: espere alguns bugs e um impacto substancial no desempenho de thread único. As construções de threads livres do CPython oferecem suporte à execução opcional com a GIL habilitada no tempo de execução usando a variável de ambiente `PYTHON_GIL` ou a opção de linha de comando `-X gil=1`.

Para verificar se o interpretador atual oferece suporte a threads livres, `python -VV` e `sys.version` contêm “experimental free-threading build”. A nova função `sys._is_gil_enabled()` pode ser usada para verificar se a GIL está realmente desabilitada no processo em execução.

Os módulos de extensão da API C precisam ser construídos especificamente para a construção com threads livres. Extensões que oferecem suporte à execução com GIL desabilitada devem usar o slot `Py_mod_gil`. Extensões que usam inicialização monofásica devem usar `PyUnstable_Module_SetGIL()` para indicar se oferecem suporte à

execução com a GIL desabilitada. A importação de extensões C que não usam esses mecanismos fará com que a GIL seja habilitada, a menos que a GIL tenha sido explicitamente desabilitada com a variável de ambiente `PYTHON_GIL` ou a opção `-X gil=0`. `pip 24.1` ou mais recentes é necessário para instalar pacotes com extensões C na construção de threads livres.

Este trabalho foi possível graças a muitos indivíduos e organizações, incluindo a grande comunidade de colaboradores do Python e projetos de terceiros para testar e habilitar o suporte para threads livres. Colaboradores notáveis incluem: Sam Gross, Ken Jin, Donghee Na, Itamar Oren, Matt Page, Brett Simmers, Dino Viehland, Carl Meyer, Nathan Goldbaum, Ralf Gommers, Lysandros Nikolaou e muitos outros. Muitos desses colaboradores são empregados pela Meta, que forneceu recursos de engenharia significativos para dar suporte a este projeto.

Ver também

PEP 703 “Tornando a trava global do interpretador opcional no CPython” contém justificativa e informações sobre este trabalho.

[Portando módulos de extensão para oferecer suporte a threads livres](#): um guia de portabilidade mantido pela comunidade para autores de extensões.

2.4 Um compilador just-in-time (JIT) experimental

Quando o CPython é configurado e construído usando a opção `--enable-experimental-jit`, um compilador just-in-time (JIT) é adicionado, o que pode acelerar alguns programas Python. No Windows, use `PCbuild/build.bat --experimental-jit` para habilitar o JIT ou `--experimental-jit-interpreter` para habilitar o interpretador Tier 2. Os requisitos de construção e outras informações de suporte [estão contidos em](#) `Tools/jit/README.md`.

A opção `--enable-experimental-jit` aceita esses valores (opcionais), assumindo como padrão `yes` se `--enable-experimental-jit` estiver presente sem o valor opcional.

- `no`: desativa todo o pipeline de Tier 2 e JIT.
- `yes`: habilita o JIT. Para desabilitar o JIT em tempo de execução, passe a variável de ambiente `PYTHON_JIT=0`.
- `yes-off`: constrói o JIT, mas desabilita-o por padrão. Para habilitar o JIT em tempo de execução, passe a variável de ambiente `PYTHON_JIT=1`.
- `interpreter`: habilita o interpretador Tier 2, mas desabilita o JIT. O interpretador pode ser desabilitado executando com `PYTHON_JIT=0`.

A arquitetura interna é aproximadamente a seguinte:

- Começamos com especializado *bytecode de Tier 1*. Veja O que há de novo no 3.11 para detalhes.
- Quando o bytecode Tier 1 fica quente o suficiente, ele é traduzido para uma nova representação intermediária (IR) puramente interna, também chamado de *IR Tier 2*, e às vezes chamada de micro-ops (“uops”).
- O IR Tier 2 usa a mesma máquina virtual baseada em pilha que o Tier 1, mas o formato de instrução é mais adequado para tradução em código de máquina.
- Temos vários passes de otimização para IR Tier 2, que são aplicados antes de serem interpretados ou traduzidos em código de máquina.
- Existe um interpretador de Tier 2, mas ele se destina principalmente à depuração dos estágios iniciais do pipeline de otimização. O interpretador Tier 2 pode ser habilitado configurando o Python com `--enable-experimental-jit=interpreter`.
- Quando o JIT está habilitado, o IR Tier 2 otimizado é traduzido em código de máquina, que é então executado.
- O processo de tradução de código de máquina usa uma técnica chamada *copiar e corrigir*. Não possui dependências de tempo de execução, mas há uma nova dependência de tempo de construção no LLVM.

 [Ver também](#)

[PEP 744](#)

(JIT de Brandt Bucher, inspirado em um artigo de Haoran Xu e Fredrik Kjolstad. Tier 2 IR de Mark Shannon e Guido van Rossum. Otimizador Tier 2 de Ken Jin.)

2.5 Definidas semânticas de mutação para `locals()`

Historicamente, o resultado esperado da mutação do valor de retorno de `locals()` foi deixada para as implementações individuais de Python definirem. A partir do Python 3.13, [PEP 667](#) padroniza o comportamento histórico do CPython para a maioria dos escopos de execução de código, mas altera escopos otimizados (funções, geradores, corrotinas, compreensões e expressões geradoras) para retornar explicitamente instantâneos independentes das variáveis locais atualmente atribuídas, incluindo variáveis não locais referenciadas localmente capturadas em encerramentos.

Esta mudança na semântica de `locals()` em escopos otimizados também afeta o comportamento padrão das funções de execução de código que visam implicitamente `locals()` se nenhum espaço de nomes explícito for fornecido (como `exec()` e `eval()`). Nas versões anteriores, se as alterações podiam ou não ser acessadas chamando `locals()` após chamar a função de execução de código dependia da implementação. Especificamente no CPython, esse código normalmente parece funcionar conforme desejado, mas às vezes pode falhar em escopos otimizados com base em outro código (incluindo depuradores e ferramentas de rastreamento de execução de código), redefinindo potencialmente o instantâneo compartilhado nesse escopo. Agora, o código sempre será executado em um instantâneo independente das variáveis locais em escopos otimizados e, portanto, as alterações nunca serão visíveis em chamadas subsequentes para `locals()`. Para acessar as alterações feitas nesses casos, uma referência explícita de espaço de nomes deve agora ser passada para a função relevante. Alternativamente, pode fazer sentido atualizar o código afetado para usar uma API de execução de código de nível superior que retorne o espaço de nomes de execução de código resultante (por exemplo, `runpy.run_path()` ao executar arquivos Python do disco).

Para garantir que depuradores e ferramentas semelhantes possam atualizar de forma confiável variáveis locais em escopos afetados por esta mudança, `FrameType.f_locals` agora retorna um proxy write-through para as variáveis locais e não locais referenciadas localmente do quadro nesses escopos, em vez de retornar uma instância `dict` compartilhada atualizada inconsistentemente com semântica de tempo de execução indefinida.


Veja [PEP 667](#) para mais detalhes, incluindo alterações e descontinuações relacionadas à API C. Notas de portabilidade também são fornecidas abaixo para as *APIs de Python* e as *APIs de C* afetadas.

(PEP e implementação são contribuição de Mark Shannon e Tian Gao em [gh-74929](#). Atualizações da documentação fornecidas por Guido van Rossum e Alyssa Coghlan.)

2.6 Suporte para plataformas móveis

PEP 730: iOS agora é uma plataforma com suporte conforme a [PEP 11](#), com os alvos `arm64-apple-ios` e `arm64-apple-ios-simulator` no nível 3 (dispositivos iPhone e iPad lançados após 2013 e o simulador Xcode iOS sendo executado em hardware Apple Silicon, respectivamente). `x86_64-apple-ios-simulator` (o simulador Xcode iOS sendo executado em hardware `x86_64` mais antigo) não é uma plataforma com suporte nível 3, mas terá o melhor suporte possível. (PEP escrita e implementação como contribuição de Russell Keith-Magee em [gh-114099](#).)

PEP 738: Android agora é uma plataforma com suporte conforme a [PEP 11](#), com os alvos `aarch64-linux-android` e `x86_64-linux-android` no nível 3. Os alvos de 32 bits `arm-linux-androideabi` e `i686-linux-android` não são plataformas com suporte de nível 3, mas terão o melhor suporte possível. (PEP escrita e implementação como contribuição de Malcolm Smith em [gh-116622](#).)

 [Ver também](#)

[PEP 730](#), [PEP 738](#)

3 Outras mudanças na linguagem

- O compilador agora remove espaços em branco comuns de cada linha em uma docstring. Isso reduz o tamanho do cache de bytecode (como arquivos `.pyc`), com reduções no tamanho do arquivo de cerca de 5%, por exemplo, em `sqlalchemy.orm.session` do SQLAlchemy 2.0. Essa alteração afeta ferramentas que usam docstrings, como `doctest`.

```
>>> def spam():
...     """
...         Esta é uma docstring com
...         espaços em branco no começo.
...
...         Ela até mesmo tem vários parágrafos!
...     """
...
>>> spam.__doc__
'\n\nEsta é uma docstring com\n\n espaços em branco no começo.\n\n\nEla até m_
-> mesmo tem vários parágrafos!\n\n'
```

(Contribuição de Inada Naoki em [gh-81283](#).)

- Escopos de anotação dentro dos escopos de classe agora pode conter lambdas e compreensões. As compreensões localizadas nos escopos de classe não são incorporadas ao escopo pai.

```
class C[T]:
    type Alias = lambda: T
```

(Contribuição de Jelle Zijlstra em [gh-109118](#) e [gh-118160](#).)

- Instruções `future` não são mais acionadas por importações relativas do módulo `__future__`, o que significa que instruções no formato `from __future__ import ...` agora são simplesmente importações relativas padrão, sem recursos especiais ativados. (Contribuição de Jeremiah Gabriel Pascual em [gh-118216](#).)
- Declarações `global` agora são permitidas em blocos `except` quando esse `global` é usado no bloco `else`. Anteriormente, isso erroneamente levantava uma exceção `SyntaxError`. (Contribuição de Irit Katriel em [gh-111123](#).)
- Adiciona `PYTHON_FROZEN_MODULES`, uma nova variável de ambiente que determina se os módulos congelados são ignorados pelo maquinário de importação, equivalente à opção de linha de comando `-X frozen_modules`. (Contribuição de Yilei Yang em [gh-111374](#).)
- Adicionado suporte ao perfilador `perf` funcionando sem `ponteiros de quadro` através da nova variável de ambiente `PYTHON_PERF_JIT_SUPPORT` e opção de linha de comando `-X perf_jit`. (Contribuição de Pablo Galindo em [gh-118518](#).)
- A localização de um arquivo `.python_history` pode ser alterada por meio da nova variável de ambiente `PYTHON_HISTORY`. (Contribuição de Levi Sabah, Zackery Spytz e Hugo van Kemenade em [gh-73965](#).)
- As classes têm um novo atributo `__static_attributes__`. Isso é preenchido pelo compilador, com uma tupla de nomes de atributos da classe que são atribuídos através de `self.<name>` de qualquer função em seu corpo. (Contribuição de Irit Katriel em [gh-115775](#).)
- O compilador agora cria um atributo `__firstlineno__` em classes com o número da primeira linha da definição de classe. (Contribuição de Serhiy Storchaka em [gh-118465](#).)
- As funções embutidas `exec()` e `eval()` agora aceitam os argumentos `globals` e `locals` como nomeados. (Contribuição de Raphael Gaschignard em [gh-105879](#).)
- A função embutida `compile()` agora aceita um novo sinalizador, `ast.PyCF_OPTIMIZED_AST`, que é semelhante a `ast.PyCF_ONLY_AST` exceto que a AST retornada é otimizada de acordo com o valor do argumento `optimize`. (Contribuição de Irit Katriel em [gh-108113](#).)
- Adiciona um atributo `__name__` em objetos `property`. (Contribuição de Eugene Toder em [gh-101860](#).)

- Adiciona `PythonFinalizationError`, uma nova exceção derivada de `RuntimeError` e usada para sinalizar quando as operações são bloqueadas durante a finalização. Os seguintes chamáveis agora levantam `PythonFinalizationError`, em vez de `RuntimeError`:

```
- _thread.start_new_thread()
- os.fork()
- os.forkpty()
- subprocess.Popen
```

(Contribuição de Victor Stinner em [gh-114570](#).)

- Permite que o argumento `count` de `str.replace()` seja um argumento nomeado. (Contribuição de Hugo van Kemenade em [gh-106487](#).)
- Muitas funções agora emitem um aviso se um valor booleano for passado como argumento do descritor de arquivo. Isso pode ajudar a detectar alguns erros mais cedo. (Contribuição de Serhiy Storchaka em [gh-82626](#).)
- Adicionados atributos `name` e `mode` para objetos arquivo ou similar compactados e arquivados nos módulos `bz2`, `lzma`, `tarfile` e `zipfile`. (Contribuição de Serhiy Storchaka em [gh-115961](#).)

4 Novos módulos

- `dbm.sqlite3`: um backend SQLite para `dbm`. (Contribuição de Raymond Hettinger e Erlend E. Aasland em [gh-100414](#).)

5 Módulos melhorados

5.1 argparse

- Adiciona o parâmetro *deprecated* aos métodos `add_argument()` e `add_parser()`, o qual permite descontinuar opções de linha de comando, argumentos posicionais e subcomandos. (Contribuição de Serhiy Storchaka em [gh-83648](#).)

5.2 array

- Adiciona o código do tipo `'w'` (`Py_UCS4`) para caracteres Unicode. Ele deve ser usado no lugar do descontinuado código do tipo `'u'`. (Contribuição de Inada Naoki em [gh-80480](#).)
- Registra `array.array` como `MutableSequence` implementando o método `clear()`. (Contribuição de Mike Zimin em [gh-114894](#).)

5.3 ast

- Os construtores dos tipos de nós no módulo `ast` agora são mais rígidos nos argumentos que aceitam, com um comportamento mais intuitivo quando os argumentos são omitidos.

Se um campo opcional em um nó AST não for incluído como argumento ao construir uma instância, o campo agora será definido como `None`. Da mesma forma, se um campo de lista for omitido, esse campo será agora definido como uma lista vazia, e se um campo `expr_context` for omitido, o padrão será `Load()`. (Anteriormente, em todos os casos, o atributo estaria ausente na instância do nó de AST recém-construída.)

Em todos os outros casos, nos quais um argumento é omitido, o construtor de nós vai emitir uma `DeprecationWarning`. Isso vai levantar uma exceção no Python 3.15. Da mesma forma, passar um argumento nomeado para o construtor que não mapeia para um campo no nó AST agora está descontinuado e vai levantar uma exceção no Python 3.15.

Estas mudanças não se aplicam às subclasses definidas pelo usuário de `ast.AST`, a menos que a classe opte pelo novo comportamento definindo o mapeamento `AST._field_types`.

(Contribuição de Jelle Zijlstra em [gh-105858](#), [gh-117486](#) e [gh-118851](#).)

- `ast.parse()` agora aceita um argumento opcional *optimize* que é passado para `compile()`. Isto torna possível obter um AST otimizado. (Contribuição de Irit Katriel em [gh-108113](#).)

5.4 asyncio

- `asyncio.as_completed()` agora retorna um objeto que é ao mesmo tempo um iterador assíncrono e um iterator simples de aguardáveis. Os aguardáveis gerados pela iteração assíncrona incluem tarefas originais ou objetos futuros que foram passados, facilitando a associação dos resultados às tarefas que estão sendo concluídas. (Contribuição de Justin Arthur em [gh-77714](#).)
- `asyncio.loop.create_unix_server()` agora removerá automaticamente o soquete Unix quando o servidor for fechado. (Contribuição de Pierre Ossman em [gh-111246](#).)
- `DatagramTransport.sendto()` agora enviará datagramas de comprimento zero se for chamado com um objeto de bytes vazios. O controle de fluxo de transporte agora também leva em conta o cabeçalho do datagrama ao calcular o tamanho do buffer. (Contribuição de Jamie Phan em [gh-115199](#).)
- Adiciona `Queue.shutdown` e `QueueShutDown` para gerenciar a terminação da fila. (Contribuição de Laurie Opperman e Yves Duprat em [gh-104228](#).)
- Adiciona os métodos `Server.close_clients()` e `Server.abort_clients()`, que fecham de forma mais forçada um servidor asyncio. (Contribuição de Pierre Ossman em [gh-113538](#).)
- Aceita uma tupla de separadores em `StreamReader.readuntil()`, parando quando qualquer um deles for encontrado. (Contribuição de Bruce Merry em [gh-81322](#).)
- Melhora o comportamento de `TaskGroup` quando um cancelamento externo colide com um cancelamento interno. Por exemplo, quando dois grupos de tarefas estão aninhados e ambos experimentam uma exceção em uma tarefa filho simultaneamente, é possível que o grupo de tarefas externo seja interrompido, porque seu cancelamento interno foi engolido pelo grupo de tarefas interno.

No caso em que um grupo de tarefas é cancelado externamente e também deve levantar `ExceptionGroup`, ele agora chamará o método `cancel()` da tarefa pai. Isso garante que uma `CancelledError` será levantada no próximo `await`, para que o cancelamento não seja perdido.

Um benefício adicional dessas mudanças é que os grupos de tarefas agora preservam a contagem de cancelamentos (`cancelling()`).

Para lidar com alguns casos extremos, `uncancel()` agora pode redefinir o sinalizador não documentado `_must_cancel` quando a contagem de cancelamentos chegar a zero.

(Inspirado por um relatório de problema relatado por Arthur Tacca em [gh-116720](#).)

- Quando `TaskGroup.create_task()` é chamado em um `TaskGroup` inativo, a corrotina fornecida será fechada (o que evita que um `RuntimeWarning` sobre a corrotina fornecida nunca seja aguardado). (Contribuição de Arthur Tacca e Jason Zhang em [gh-115957](#).)

5.5 base64

- Adiciona as funções `z85encode()` e `z85decode()` para codificar bytes como dados Z85 e decodificar dados codificados em Z85 para bytes. (Contribuição de Matan Perelman em [gh-75299](#).)

5.6 compileall

- O número padrão de threads e processos de trabalho é agora selecionado usando `os.process_cpu_count()` em vez de `os.cpu_count()`. (Contribuição de Victor Stinner em [gh-109649](#).)

5.7 concurrent.futures

- O número padrão de threads e processos de trabalho é agora selecionado usando `os.process_cpu_count()` em vez de `os.cpu_count()`. (Contribuição de Victor Stinner em [gh-109649](#).)

5.8 configparser

- `ConfigParser` agora tem suporte para seções sem nome, o que permite pares de chave-valor de nível superior. Isso pode ser habilitado com o novo parâmetro `allow_unnamed_section`. (Contribuição de Pedro Sousa Lacerda em [gh-66449](#).)

5.9 copy

- A nova função `replace()` e o protocolo `replace` tornam a criação de cópias modificadas de objetos muito mais simples. Isso é especialmente útil ao trabalhar com objetos imutáveis. Os seguintes tipos dão suporte à função `replace()` e implementam o protocolo `replace`:

- `collections.namedtuple()`
- `dataclasses.dataclass`
- `datetime.datetime`, `datetime.date`, `datetime.time`
- `inspect.Signature`, `inspect.Parameter`
- `types.SimpleNamespace`
- objetos código

Qualquer classe definida pelo usuário também pode ter suporte `copy.replace()` definindo o método `__replace__()`. (Contribuição de Serhiy Storchaka em [gh-108751](#).)

5.10 ctypes

- Como consequência da refatoração interna necessária, a inicialização de metaclasses internas agora acontece em `__init__` em vez de `__new__`. Isso afeta projetos que fazem subclasse dessas metaclasses internas para fornecer inicialização personalizada. De forma geral:

- A lógica personalizada que foi feita em `__new__` após chamar `super().__new__` deve ser movida para `__init__`.
- Para criar uma classe, chame a metaclasses, não apenas o método `__new__` da metaclasses.

Veja [gh-124520](#) para discussão e links para mudanças em alguns projetos afetados.

- Os objetos `ctypes.Structure` têm um novo atributo `_align_` que permite que o alinhamento da estrutura que está sendo empacotada para/da memória seja especificado explicitamente. (Contribuição de Matt Sanderson em [gh-112433](#))

5.11 dbm

- Adiciona `dbm.sqlite3`, um novo módulo que implementa um backend SQLite, e torna-o o backend padrão `dbm`. (Contribuição de Raymond Hettinger e Erlend E. Aasland em [gh-100414](#).)
- Permite a remoção de todos os itens do banco de dados por meio dos novos métodos `gdbm.clear()` e `ndbm.clear()`. (Contribuição de Donghee Na em [gh-107122](#).)

5.12 dis

- Altera a saída das funções do módulo `dis` para mostrar rótulos lógicos para alvos de salto e manipuladores de exceção, em vez de deslocamentos. Os deslocamentos podem ser adicionados com a nova opção de linha de comando `-O` ou o argumento `show_offsets`. (Contribuição de Irit Katriel em [gh-112137](#).)
- `get_instructions()` não representa mais entradas de cache como instruções separadas. Em vez disso, ele os retorna como parte de `Instruction`, no novo campo `cache_info`. O argumento `show_caches` para `get_instructions()` foi descontinuado e não tem mais nenhum efeito. (Contribuição de Irit Katriel em [gh-112962](#).)

5.13 doctest

- A saída `doctest` é agora colorida por padrão. Isso pode ser controlado através da nova variável de ambiente `PYTHON_COLORS`, bem como das variáveis de ambiente canônicas `NO_COLOR` e `FORCE_COLOR`. Veja também `using-on-controlling-color`. (Contribuição de Hugo van Kemenade em [gh-117225](#).)
- O método `DocTestRunner.run()` agora conta o número de testes ignorados. Adiciona os atributos `DocTestRunner.skips` e `TestResults.skipped`. (Contribuição de Victor Stinner em [gh-108794](#).)

5.14 email

- Cabeçalhos com novas linhas embutidas são agora colocadas 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](#).)
- `getaddresses()` e `parseaddr()` agora retornam pares `(' ', '')` em mais situações onde endereços de e-mail inválidos são encontrados em vez de valores potencialmente imprecisos. As duas funções têm agora um novo parâmetro opcional `strict` (padrão `True`). Para obter o comportamento antigo (aceitar entrada malformada), `strict=False`. `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 [CVE 2023-27043](#).)

5.15 fractions

- Os objetos `Fraction` agora oferecem suporte às regras padrão de minilinguagem de especificação de formato para preenchimento, alinhamento, manipulação de sinais, largura mínima e agrupamento. (Contribuição de Mark Dickinson em [gh-111320](#).)

5.16 glob

- Adiciona `translate()`, uma função que converte uma especificação de caminho com curingas estilo shell em uma expressão regular. (Contribuição de Barney Gale em [gh-72904](#).)

5.17 importlib

- As seguintes funções em `importlib.resources` agora permitem acessar um diretório (ou árvore) de recursos, usando múltiplos argumentos posicionais (os argumentos `encoding` e `errors` nas funções de leitura de texto agora são somente-nomeados):

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

Essas funções não estão mais descontinuadas e não estão programadas para remoção. (Contribuição de Petr Viktorin em [gh-116608](#).)

- A `contents()` permanece descontinuada em favor da API completa `Traversable`. No entanto, agora não há planos para removê-la. (Contribuição de Petr Viktorin em [gh-116608](#).)

5.18 io

- O finalizador `IOBase` agora registra quaisquer erros levantados pelo método `close()` com `sys.unraisablehook`. Anteriormente, os erros eram ignorados silenciosamente por padrão e registrados apenas em Modo de Desenvolvimento do Python ou ao usar uma construção de depuração do Python. (Contribuição de Victor Stinner em [gh-62948](#).)

5.19 ipaddress

- Adiciona a propriedade `IPv4Address.ipv6_mapped`, que retorna o endereço IPv6 mapeado para IPv4. (Contribuição de Charles Machalow em [gh-109466](#).)
- Corrige os comportamentos `is_global` e `is_private` em `IPv4Address`, `IPv6Address`, `IPv4Network` e `IPv6Network`. (Contribuição de Jakub Stasiak em [gh-113171](#).)

5.20 itertools

- `batched()` tem um novo parâmetro `strict`, que levanta uma `ValueError` se o lote final for menor que o tamanho do lote especificado. (Contribuição de Raymond Hettinger em [gh-113202](#).)

5.21 marshal

- Adiciona o parâmetro `allow_code` nas funções do módulo. Passar `allow_code=False` evita a serialização e desserialização de objetos de código que são incompatíveis entre versões do Python. (Contribuição de Serhiy Storchaka em [gh-113626](#).)

5.22 math

- A nova função `fma()` realiza operações fundidas de multiplicação e adição. Ela calcula $x * y + z$ com apenas uma única rodada e, portanto, evita qualquer perda intermediária de precisão. Ele envolve a função `fma()` fornecida pelo C99 e segue a especificação da operação IEEE 754 “fusedMultiplyAdd” para casos especiais. (Contribuição de Mark Dickinson e Victor Stinner em [gh-73468](#).)

5.23 mimetypes

- Adiciona a função `guess_file_type()` para adivinhar um tipo MIME de um caminho do sistema de arquivos. Usar caminhos com `guess_type()` agora está suavemente descontinuada. (Contribuição de Serhiy Storchaka em [gh-66543](#).)

5.24 mmap

- `mmap` agora está protegida contra travamentos no Windows quando a memória mapeada está inacessível devido a erros no sistema de arquivos ou violações de acesso. (Contribuição de Jannis Weigend em [gh-118209](#).)
- `mmap` tem um novo método `seekable()` que pode ser usado quando um objeto pesquisável arquivo ou similar é necessário. O método `seek()` agora retorna a nova posição absoluta. (Contribuição de Donghee Na e Sylvie Liberman em [gh-111835](#).)
- O novo parâmetro `trackfd` somente para UNIX para `mmap` controla a duplicação do descritor de arquivo; se falso, o descritor de arquivo especificado por `fileno` não será duplicado. (Contribuição de Zackery Spytz e Petr Viktorin em [gh-78502](#).)

5.25 multiprocessing

- O número padrão de threads e processos de trabalho é agora selecionado usando `os.process_cpu_count()` em vez de `os.cpu_count()`. (Contribuição de Victor Stinner em [gh-109649](#).)

5.26 os

- Adiciona a função `process_cpu_count()` para obter o número de núcleos lógicos de CPU utilizáveis pela thread de chamada do processo atual. (Contribuição de Victor Stinner em [gh-109649](#).)
- `cpu_count()` e `process_cpu_count()` podem ser substituídos através da nova variável de ambiente `PYTHON_CPU_COUNT` ou da nova opção de linha de comando `-X cpu_count`. Esta opção é útil para usuários que precisam limitar os recursos da CPU de um sistema contêiner sem precisar modificar código da aplicação ou o contêiner em si. (Contribuição de Donghee Na em [gh-109595](#).)
- Adiciona uma interface de baixo nível aos *descritores de arquivo de temporizador* do Linux via `timerfd_create()`, `timerfd_settime()`, `timerfd_gettime_ns()`, `timerfd_gettime()`, `timerfd_gettime_ns()`, `TFD_NONBLOCK`, `TFD_CLOEXEC`, `TFD_TIMER_ABSTIME` e `TFD_TIMER_CANCEL_ON_SET` (Contribuição de Masaru Tsuchiyama em [gh-108277](#).)
- `lchmod()` e o argumento `follow_symlinks` em `chmod()` estão agora disponíveis no Windows. Observe que o valor padrão de `follow_symlinks` em `lchmod()` é `False` no Windows. (Contribuição de Serhiy Storchaka em [gh-59616](#).)
- `fchmod()` e suporte para descritores de arquivo em `chmod()` estão agora disponíveis no Windows. (Contribuição de Serhiy Storchaka em [gh-113191](#).)
- No Windows, `mkdir()` e `makedirs()` agora oferecem suporte a passar um valor *mode* de `0o700` para aplicar controle de acesso ao novo diretório. Isso afeta implicitamente `tempfile.mkdtemp()` e é uma mitigação para [CVE 2024-4030](#). Outros valores para *mode* continuam a ser ignorados. (Contribuição de Steve Dower em [gh-118486](#).)
- `posix_spawn()` agora aceita `None` para o argumento *env*, o que faz com que o processo recém-gerado use o ambiente de processo atual. (Contribuição de Jakub Kulik em [gh-113119](#).)
- `posix_spawn()` agora pode usar o atributo `POSIX_SPAWN_CLOSEFROM` no parâmetro *file_actions* em plataformas que oferecem suporte a `posix_spawn_file_actions_addclosefrom_np()`. (Contribuição de Jakub Kulik em [gh-113117](#).)

5.27 os.path

- Adiciona `isreserved()` para verificar se um caminho está reservado no sistema atual. Esta função está disponível apenas no Windows. (Contribuição de Barney Gale em [gh-88569](#).)
- No Windows, `isabs()` não considera mais caminhos que começam com exatamente uma barra (`\` ou `/`) como absolutos. (Contribuição de Barney Gale e Jon Foster em [gh-44626](#).)
- `realpath()` agora resolve nomes de arquivos no estilo MS-DOS, mesmo que o arquivo não esteja acessível. (Contribuição de Moonsik Park em [gh-82367](#).)

5.28 pathlib

- Adiciona `UnsupportedOperation`, que é levantada em vez de `NotImplementedError` quando uma operação de caminho não é possível. (Contribuição de Barney Gale em [gh-89812](#).)
- Adiciona um novo construtor para criar objetos `Path` a partir de URIs ‘file’ (`file:///`), `Path.from_uri()`. (Contribuição de Barney Gale em [gh-107465](#).)
- Adiciona `PurePath.full_match()` para corresponder caminhos com curingas do tipo shell, incluindo o curinga recursivo `***`. (Contribuição de Barney Gale em [gh-73435](#).)
- Adiciona o atributo de classe `PurePath.parser` para armazenar a implementação de `os.path` usada para análise e junção de caminhos de baixo nível. Isso vai ser `posixpath` ou `ntpath`.
- Adiciona o argumento somente-nomeado `recurse_symlinks` a `Path.glob()` e `rglob()`. (Contribuição de Barney Gale em [gh-77609](#).)
- `Path.glob()` e `rglob()` agora retornam arquivos e diretórios quando recebem um padrão que termina com `***`. Anteriormente, apenas os diretórios eram retornados. (Contribuição de Barney Gale em [gh-70303](#).)

- Adiciona o argumento somente-nomeado *follow_symlinks* a `Path.is_file`, `Path.is_dir`, `Path.owner()` e `Path.group()`. (Contribuição de Barney Gale em [gh-105793](#) e Kamil Turek em [gh-107962](#).)

5.29 pdb

- `breakpoint()` e `set_trace()` agora entram no depurador imediatamente em vez de na próxima linha de código a ser executada. Esta mudança evita que o depurador quebre fora do contexto quando `breakpoint()` estiver posicionado no final do contexto. (Contribuição de Tian Gao em [gh-118579](#).)
- `sys.path[0]` não mais é substituído pelo diretório do script que está sendo depurado quando `sys.flags.safe_path` está definido. (Contribuição de Tian Gao e Christian Walther em [gh-111762](#).)
- `zipapp` agora é suportado como alvo de depuração. (Contribuição de Tian Gao em [gh-118501](#).)
- Adiciona a capacidade de mover entre exceções encadeadas durante a depuração post-mortem em `pm()` usando o novo comando `exceptions [número_exc]` para `Pdb`. (Contribuição de Matthias Bussonnier em [gh-106676](#).)
- Expressões e instruções cujo prefixo é um comando `pdb` agora são identificadas e executadas corretamente. (Contribuição de Tian Gao em [gh-108464](#).)

5.30 queue

- Adiciona `Queue.shutdown` e `ShutDown` para gerenciar a terminação da fila. (Contribuição de Laurie Opperman e Yves Duprat em [gh-104750](#).)

5.31 random

- Adiciona uma interface de linha de comando. (Contribuição de Hugo van Kemenade em [gh-118131](#).)

5.32 re

- Renomeia `re.error` para `PatternError` para maior clareza. `re.error` é mantido para compatibilidade com versões anteriores.

5.33 shutil

- Suporte aos argumentos nomeados *dir_fd* e *follow_symlinks* em `chown()`. (Contribuição de Berker Peksag e Tahia K em [gh-62308](#))

5.34 site

- Os arquivos `.pth` agora são decodificados primeiro usando UTF-8 e depois usando codificação da localidade se a decodificação UTF-8 falhar. (Contribuição de Inada Naoki em [gh-117802](#).)

5.35 sqlite3

- Uma `ResourceWarning` agora é emitida se um objeto `Connection` não for explicitamente fechado. (Contribuição de Erlend E. Aasland em [gh-105539](#).)
- Adiciona o parâmetro somente-nomeado *filter* a `Connection.iterdump()` para filtrar objetos de banco de dados para despejo. (Contribuição de Mariusz Felisiak em [gh-91602](#).)

5.36 ssl

- A API `create_default_context()` agora inclui `VERIFY_X509_PARTIAL_CHAIN` e `VERIFY_X509_STRICT` em seus sinalizadores padrão.

Nota

`VERIFY_X509_STRICT` pode rejeitar certificados pré-**RFC 5280** ou malformados que a implementação OpenSSL subjacente poderia aceitar. Embora não seja recomendado desativar isso, você pode fazer isso usando:

```
import ssl

ctx = ssl.create_default_context()
ctx.verify_flags &= ~ssl.VERIFY_X509_STRICT
```

(Contribuição de William Woodruff em [gh-112389](#).)

5.37 statistics

- Adiciona `kde()` para estimativa de densidade do núcleo. Isto torna possível estimar uma função de densidade de probabilidade contínua a partir de um número fixo de amostras discretas. (Contribuição de Raymond Hettinger em [gh-115863](#).)
- Adiciona `kde_random()` para amostragem de uma função de densidade de probabilidade estimada criada por `kde()`. (Contribuição de Raymond Hettinger em [gh-115863](#).)

5.38 subprocess

- O módulo `subprocess` agora usa a função `posix_spawn()` em mais situações.

Notavelmente, quando `close_fds` é `True` (o padrão), `posix_spawn()` será usado quando a biblioteca C fornecer `posix_spawn_file_actions_addclosefrom_np()`, que inclui versões recentes do Linux, FreeBSD e Solaris. No Linux, isso deve ter um desempenho semelhante ao código existente baseado em `vfork()` do Linux.

Um botão de controle privado `subprocess._USE_POSIX_SPAWN` pode ser definido como `False` se você precisar forçar `subprocess` a nunca usar `posix_spawn()`. Por favor, informe o motivo e os detalhes da plataforma no rastreador de problemas se você definir isso para que possamos melhorar nossa lógica de seleção de API para todos. (Contribuído por Jakub Kulik em [gh-113117](#).)

5.39 sys

- Adiciona a função `_is_interned()` para testar se uma string foi internada. Não há garantia de que esta função exista em todas as implementações do Python. (Contribuição de Serhiy Storchaka em [gh-78573](#).)

5.40 tempfile

- No Windows, o modo padrão `0o700` usado por `tempfile.mkdtemp()` agora limita o acesso ao novo diretório devido a alterações em `os.mkdir()`. Esta é uma mitigação para **CVE 2024-4030**. (Contribuição de Steve Dower em [gh-118486](#).)

5.41 time

- No Windows, `monotonic()` agora usa o relógio `QueryPerformanceCounter()` para uma resolução de 1 microssegundo, em vez do relógio `GetTickCount64()` que tem uma resolução de 15.6 milissegundos. (Contribuição de Victor Stinner em [gh-88494](#).)
- No Windows, `time()` agora usa o relógio `GetSystemTimePreciseAsFileTime()` para uma resolução de 1 microssegundo, em vez do relógio `GetSystemTimeAsFileTime()` que tem uma resolução de 15.6 milissegundos. (Contribuição de Victor Stinner em [gh-63207](#).)

5.42 tkinter

- Adiciona métodos de widget tkinter: `tk_busy_hold()`, `tk_busy_configure()`, `tk_busy_cget()`, `tk_busy_forget()`, `tk_busy_current()` e `tk_busy_status()`. (Contribuição de Miguel, klappnase e Serhiy Storchaka em [gh-72684](#).)
- O método `wm_attributes()` de widget tkinter agora aceita o nome do atributo sem o prefixo negativo para obter atributos de janela (por exemplo, `w.wm_attributes('alpha')`) e permite especificar atributos e valores para definir como argumentos nomeados (por exemplo, `w.wm_attributes(alpha=0.5)`). (Contribuição de Serhiy Storchaka em [gh-43457](#).)
- `wm_attributes()` agora pode retornar atributos como um `dict`, usando o novo parâmetro somente-nomeado opcional `return_python_dict`. (Contribuição de Serhiy Storchaka em [gh-43457](#).)
- `Text.count()` agora pode retornar um `int` simples quando o novo parâmetro somente-nomeado opcional `return_ints` é usado. Caso contrário, a contagem única é retornada como uma tupla de 1 elemento ou `None`. (Contribuição de Serhiy Storchaka em [gh-97928](#).)
- Suporte ao tipo de elemento “vsapi” no método `element_create()` de `tkinter.ttk.Style`. (Contribuição de Serhiy Storchaka em [gh-68166](#).)
- Adiciona o método `after_info()` para widgets do Tkinter. (Contribuição de Cheryl Sabella em [gh-77020](#).)
- Adiciona um novo método `copy_replace()` a `PhotoImage` para copiar uma região de uma imagem para outra, possivelmente com zoom de pixel, subamostragem ou ambos. (Contribuição de Serhiy Storchaka em [gh-118225](#).)
- Adiciona parâmetro `from_coords` para os métodos `copy()`, `zoom()` e `subsample()` da classe `PhotoImage`. Adiciona parâmetros `zoom` e `subsample` para o método `copy()` da classe `PhotoImage`. (Contribuição de Serhiy Storchaka em [gh-118225](#).)
- Adiciona os métodos `PhotoImage.read()` para ler uma imagem de um arquivo e `data()` para obter os dados da imagem. Adiciona os parâmetros `background` e `grayscale` ao método `write()`. (Contribuição de Serhiy Storchaka em [gh-118271](#).)

5.43 traceback

- Adiciona o atributo `exc_type_str` a `TracebackException`, que contém uma exibição de string do `exc_type`. Descontinua o atributo `exc_type`, que contém o próprio objeto de tipo. Adicione o parâmetro `save_exc_type` (padrão `True`) para indicar se `exc_type` deve ser salvo. (Contribuição de Irit Katriel em [gh-112332](#).)
- Adiciona um novo parâmetro somente-nomeado `show_group` a `TracebackException`. `format_exception_only()` para formatar (recursivamente) as exceções aninhadas de uma instância `BaseExceptionGroup`. (Contribuição de Irit Katriel em [gh-105292](#).)

5.44 types

- `SimpleNamespace` agora pode receber um único argumento posicional para inicializar os argumentos do espaço de nomes. Este argumento deve ser um mapeamento ou um iterável de pares chave-valor. (Contribuição de Serhiy Storchaka em [gh-108191](#).)

5.45 typing

- **PEP 705:** adiciona `ReadOnly`, uma construção especial de tipagem para marcar um item `TypedDict` como somente leitura para verificadores de tipo.
- **PEP 742:** adiciona `TypeIs`, uma construção de tipagem que pode ser usada para instruir um verificador de tipos sobre como restringir um tipo.
- Adiciona `NoDefault`, um objeto sinalizador usado para representar os padrões de alguns parâmetros no módulo `typing`. (Contribuição de Jelle Zijlstra em [gh-116126](#).)
- Adicione `get_protocol_members()` para retornar o conjunto de membros que definem uma `typing.Protocol`. (Contribuição de Jelle Zijlstra em [gh-104873](#).)

- Adiciona `is_protocol()` para verificar se uma classe é uma `Protocol`. (Contribuição de Jelle Zijlstra em [gh-104873](#).)
- `ClassVar` agora pode ser aninhado em `Final`, e vice-versa. (Contribuição de Mehdi Drissi em [gh-89547](#).)

5.46 unicodedata

- Atualiza o banco de dados Unicode para a versão 15.1.0. (Contribuição de James Gerity em [gh-109559](#).)

5.47 venv

- Adiciona suporte para criar arquivos ignorados pelo gerenciamento de controle de fontes (SCM) no diretório de um ambiente virtual. Por padrão, o Git é compatível. Isso está implementado como ativado por padrão por meio da API, que pode ser estendido para oferecer suporte a outros SCMs (`EnvBuilder` e `create()`) e desativado por padrão por meio da CLI, usando `--without-scm-ignore-files`. (Contribuição de Brett Cannon em [gh-108125](#).)

5.48 warnings

- **PEP 702:** O novo decorador `warnings.deprecated()` fornece uma maneira de comunicar desconituações para um verificador de tipo estático e avisar sobre o uso de classes e funções obsoletas. Um `DeprecationWarning` também pode ser emitido quando uma função ou classe decorada é usada em tempo de execução. (Contribuição de Jelle Zijlstra em [gh-104003](#).)

5.49 xml

- Permite controlar o adiamento da nova análise do Expat $\geq 2.6.0$ (**CVE 2023-52425**) adicionando cinco novos métodos:

```
- xml.etree.ElementTree.XMLParser.flush()
- xml.etree.ElementTree.XMLPullParser.flush()
- xml.parsers.expat.xmlparser.GetReparseDeferralEnabled()
- xml.parsers.expat.xmlparser.SetReparseDeferralEnabled()
- xml.sax.expatreader.ExpatParser.flush()
```

(Contribuição de Sebastian Pipping em [gh-115623](#).)

- Adiciona o método `close()` para o iterador retornado por `iterparse()` para limpeza explícita. (Contribuição de Serhiy Storchaka em [gh-69893](#).)

5.50 zipimport

- Adiciona suporte para arquivos no formato ZIP64. Todo mundo adora dados enormes, certo? (Contribuição de Tim Hatch em [gh-94146](#).)

6 Otimizações

- Vários módulos de biblioteca padrão tiveram seus tempos de importação significativamente melhorados. Por exemplo, o tempo de importação do módulo `typing` foi reduzido em cerca de um terço pela remoção de dependências em `re` e `contextlib`. Outros módulos para aproveitar a aceleração no tempo de importação incluem `email.utils`, `enum`, `functools`, `importlib.metadata` e `threading`. (Contribuição de Alex Waygood, Shantanu Jain, Adam Turner, Daniel Hollas e outros em [gh-109653](#).)
- `textwrap.indent()` agora é cerca de 30% mais rápido do que antes para entradas grandes. (Contribuição de Inada Naoki em [gh-107369](#).)
- O módulo `subprocess` agora a função `posix_spawn()` em mais situações, incluindo quando `close_fds` é `True` (o padrão) em muitas plataformas. Isto deve fornecer um aumento notável de desempenho ao iniciar

processos no FreeBSD e Solaris. Veja a seção de *subprocess* acima para detalhes. (Contribuição de Jakub Kulik em [gh-113117](#).)

7 Módulos e APIs removidas

7.1 PEP 594: remove “baterias descarregadas” da biblioteca padrão

PEP 594 propôs remover 19 módulos da biblioteca padrão, coloquialmente chamados de “baterias descarregadas” devido ao seu status histórico, obsoleto ou inseguro. Todos os módulos a seguir foram descontinuados no Python 3.11 e agora foram removidos:

- `aifc`
- `audioop`
- `chunk`
- `cgi` e `cgitb`
 - `cgi.FieldStorage` normalmente pode ser substituído por `urllib.parse.parse_qs()` para solicitações GET e HEAD, e o módulo `email.message` ou a biblioteca `multipart` para solicitações POST e PUT.
 - `cgi.parse()` pode ser substituído chamando `urllib.parse.parse_qs()` diretamente na string de consulta desejada, a menos que a entrada seja `multipart/form-data`, que deve ser substituída conforme descrito abaixo para `cgi.parse_multipart()`.
 - `cgi.parse_header()` pode ser substituído pela funcionalidade do pacote `email`, que implementa os mesmos RFCs MIME. Por exemplo, com `email.message.EmailMessage`:

```
from email.message import EmailMessage

msg = EmailMessage()
msg['content-type'] = 'application/json; charset="utf8"'
main, params = msg.get_content_type(), msg['content-type'].params
```

- `cgi.parse_multipart()` pode ser substituído pela funcionalidade no pacote `email`, que implementa os mesmos RFCs de MIME, ou pela biblioteca `multipart`. Por exemplo, as classes `email.message.EmailMessage` e `email.message.Message`.
- `crypt` e a extensão privada `_crypt`. O módulo `hashlib` pode ser uma substituição apropriada quando simplesmente fazer hash de um valor é necessário. Caso contrário, várias bibliotecas de terceiros no PyPI estão disponíveis:
 - `bcrypt`: hashing de senha moderno para seu software e servidores.
 - `passlib`: framework abrangente de hash de senha com suporte para mais de 30 esquemas.
 - `argon2-cffi`: algoritmo seguro de hashing de senha Argon2.
 - `legacrypt`: envólucro ao `ctypes` para a chamada POSIX da biblioteca de criptografia e funcionalidade associada.
 - `crypt_r`: fork do módulo `crypt`, um envólucro para a chamada da biblioteca `crypt_r(3)` e funcionalidade associada.
- `imghdr`: as bibliotecas `filetype`, `puremagic` ou `python-magic` devem ser usadas como substituições. Por exemplo, a função `puremagic.what()` pode ser usada para substituir a função `imghdr.what()` para todos os formatos de arquivo que tinham suporte de `imghdr`.
- `mailcap`: use o módulo `mimetypes`.
- `msilib`
- `nis`
- `nntplib`: use a biblioteca `pynntp` do PyPI.

- `ossaudiodev`: para reprodução de áudio, use a biblioteca `pygame` do PyPI.
- `pipes`: use o módulo `subprocess`.
- `sndhdr`: as bibliotecas `filetype`, `puremagic` ou `python-magic` devem ser usadas como substitutas.
- `spwd`: use a biblioteca `python-pam` do PyPI.
- `sunau`
- `telnetlib`: use as bibliotecas `telnetlib3` ou `Exscript` do PyPI.
- `uu`: use o módulo `base64` como uma alternativa moderna.
- `xdrlib`

(Contribuição de Victor Stinner e Zachary Ware em [gh-104773](#) e [gh-104780](#).)

7.2 2to3

- Remove o programa `2to3` e o módulo `lib2to3`, já descontinuados no Python 3.11. (Contribuição de Victor Stinner em [gh-104780](#).)

7.3 embutidos

- Remove suporte aos descritores `classmethod` encadeados (introduzidos em [gh-63272](#)). Eles não podem mais serem usados para agrupar outros descritores como `property`. O design central desse recurso apresentava falhas e levou a problemas. Para “passar” um `classmethod`, considere usar o atributo `__wrapped__` que foi adicionado no Python 3.10. (Contribuição de Raymond Hettinger em [gh-89519](#).)
- Chamar `frame.clear()` em um quadro suspenso levanta `RuntimeError` (como sempre foi o caso para um quadro em execução). (Contribuição de Irit Katriel em [gh-79932](#).)

7.4 configparser

- Remove a classe não documentada `LegacyInterpolation`, descontinuada na docstring desde Python 3.2 e em tempo de execução desde Python 3.11. (Contribuição de Hugo van Kemenade em [gh-104886](#).)

7.5 importlib.metadata

- Remove o acesso descontinuado a subscrito (`__getitem__()`) para objetos `EntryPoint`. (Contribuição de Jason R. Coombs em [gh-113175](#).)

7.6 locale

- Remove a função `locale.resetlocale()`, descontinuada no Python 3.11. Use `locale.setlocale(locale.LC_ALL, "")`. (Contribuição de Victor Stinner em [gh-104783](#).)

7.7 opcode

- Move `opcode.ENABLE_SPECIALIZATION` para `_opcode.ENABLE_SPECIALIZATION`. Este campo foi adicionado na versão 3.12, nunca foi documentado e não se destina ao uso externo. (Contribuição de Irit Katriel em [gh-105481](#).)
- Remove `opcode.is_pseudo()`, `opcode.MIN_PSEUDO_OPCODE` e `opcode.MAX_PSEUDO_OPCODE`, que foram adicionados no Python 3.12, mas nunca foram documentados nem expostos através de `dis`, e não foram planejados para serem usados externamente. (Contribuição de Irit Katriel em [gh-105481](#).)

7.8 pathlib

- Remove a capacidade de usar objetos `Path` como gerenciadores de contexto. Essa funcionalidade foi descontinuada e não tinha efeito desde o Python 3.9. (Contribuição de Barney Gale em [gh-83863](#).)

7.9 re

- Remove a função `re.template()` não documentada, descontinuada e quebrada e o sinalizador `re.TEMPLATE / re.T`. (Contribuição de Serhiy Storchaka e Nikita Sobolev em [gh-105687](#).)

7.10 tkinter.tix

- Remove o módulo `tkinter.tix`, descontinuado no Python 3.6. A biblioteca Tix de terceiros que o módulo empacota não é mantida. (Contribuição de Zachary Ware em [gh-75552](#).)

7.11 turtle

- Remove o método `RawTurtle.settiltangle()`, descontinuado na documentação desde o Python 3.1 e em tempo de execução desde o Python 3.11. (Contribuição de Hugo van Kemenade em [gh-104876](#).)

7.12 typing

- Remove os espaços de nomes `typing.io` e `typing.re`, descontinuados desde o Python 3.8. Os itens nesses espaços de nomes podem ser importados diretamente do módulo `typing`. (Contribuição de Sebastian Rittau em [gh-92871](#).)
- Remove o método de argumento nomeado de criação de tipos `TypedDict`, descontinuados no Python 3.11. (Contribuição de Tomas Roun em [gh-104786](#).)

7.13 unittest

- Remove as seguintes funções `unittest`, descontinuadas no Python 3.11:

```
- unittest.findTestCases()
- unittest.makeSuite()
- unittest.getTestCaseNames()
```

Em vez delas, use os métodos de `TestLoader`:

```
- loadTestsFromModule()
- loadTestsFromTestCase()
- getTestCaseNames()
```

(Contribuição de Hugo van Kemenade em [gh-104835](#).)

- Remove o método `TestProgram.usageExit()` não testado e não documentado, descontinuado no Python 3.11. (Contribuição de Hugo van Kemenade em [gh-104992](#).)

7.14 urllib

- Remove os parâmetros `cafile`, `capath` e `cadefault` da função `urllib.request.urlopen()`, descontinuada no Python 3.6. Em vez disso, use o parâmetro `context` com uma instância de `SSLContext`. A função `ssl.SSLContext.load_cert_chain()` pode ser usada para carregar certificados específicos ou deixe `ssl.create_default_context()` selecionar os certificados de autoridade certificadora (AC) confiável do sistema operacional. (Contribuição de Victor Stinner em [gh-105382](#).)

7.15 webbrowser

- Remove a classe `MacOSX` não testada e não documentada, descontinuada no Python 3.11. Use a classe `MacOSXOSAScript` (introduzida no Python 3.2). (Contribuição de Hugo van Kemenade em [gh-104804](#).)
- Remove o atributo descontinuado `MacOSXOSAScript._name`. Use o atributo `MacOSXOSAScript.name`. (Contribuição de Nikita Sobolev em [gh-105546](#).)

8 Novas descontinuações

- Funções definidas pelo usuário:
 - A atribuição ao atributo `__code__` de uma função onde o tipo do novo objeto código não corresponde ao tipo da função está descontinuada. Os diferentes tipos são: função simples, gerador, gerador assíncrono e corrotina. (Contribuição de Irit Katriel em [gh-81137](#).)
- `array`:
 - Descontinua o código de formato `'u'` (`wchar_t`). Este código de formato foi descontinuado na documentação desde o Python 3.3 e será removido no Python 3.16. Use o código de formato `'w'` (`Py_UCS4`) para caracteres Unicode. (Contribuição de Hugo van Kemenade em [gh-80480](#).)
- `ctypes`:
 - Descontinua a função não documentada `SetPointerType()`, que será removida no Python 3.15. (Contribuição de Victor Stinner em [gh-105733](#).)
 - Descontinua suavemente a função `ARRAY()` em favor da multiplicação `type * length`. (Contribuição de Victor Stinner em [gh-105733](#).)
- `decimal`:
 - Descontinua o especificador de formato de `Decimal` não padrão e não documentado `'N'`, que só é suportado na implementação C do módulo `decimal`. (Contribuição de Serhiy Storchaka em [gh-89902](#).)
- `dis`:
 - Descontinua o separador `HAVE_ARGUMENT`. Em vez disso, Verifique a associação em `hasarg`. (Contribuição de Irit Katriel em [gh-109319](#).)
- `getopt` e `optparse`:
 - Ambos os módulos agora estão suavemente descontinuados, com `argparse` preferido para novos projetos. Esta é uma nova descontinuação suave para o módulo `getopt`, enquanto o módulo `optparse` já estava *de fato* suavemente descontinuado. (Contribuição de Victor Stinner em [gh-106535](#).)
- `gettext`:
 - Descontinua números não inteiros como argumentos para funções e métodos que consideram formas plurais no módulo `gettext`, ainda que nenhuma tradução tenha sido encontrada. (Contribuição de Serhiy Storchaka em [gh-88434](#).)
- `glob`:
 - Descontinua as funções não documentadas `glob0()` e `glob1()`. Em vez disso, use `glob()` e passe um objeto caminho ou similar especificando o diretório raiz para o parâmetro `root_dir`. (Contribuição de Barney Gale em [gh-117337](#).)
- `http.server`:
 - Descontinua `CGIHTTPRequestHandler`, a ser removido no Python 3.15. Servidores HTTP CGI baseados em processos estão fora de moda há muito tempo. Este código estava desatualizado, sem manutenção e raramente era usado. Ele tem um alto potencial para bugs de segurança e funcionalidade. (Contribuição de Gregory P. Smith em [gh-109096](#).)
 - Descontinua o sinalizador `--cgi` para a interface de linha de comando `python -m http.server`, a ser removido no Python 3.15. (Contribuição de Gregory P. Smith em [gh-109096](#).)
- `mimetypes`:
 - Descontinua suavemente argumentos caminho de arquivo para `guess_type()`, use `guess_file_type()` em vez disso. (Contribuição de Serhiy Storchaka em [gh-66543](#).)
- `re`:

- Descontinua a passagem dos argumentos opcionais *maxsplit*, *count* ou *flags* como argumentos posicionais para as funções de nível de módulo `split()`, `sub()` e `subn()`. Esses parâmetros se tornarão somente-nomeados em uma versão futura do Python. (Contribuição de Serhiy Storchaka em [gh-56166](#).)
- `pathlib`:
 - Descontinua `PurePath.is_reserved()`, a ser removido no Python 3.15. Use `os.path.isreserved()` para detectar caminhos reservados no Windows. (Contribuição de Barney Gale em [gh-88569](#).)
- `platform`:
 - Descontinua `java_ver()`, a ser removida no Python 3.15. Esta função é útil apenas para suporte a Jython, tem uma API confusa e é amplamente não testada. (Contribuição de Nikita Sobolev em [gh-116349](#).)
- `pydoc`:
 - Descontinua a função não documentada `ispackage()`. (Contribuição de Zackery Spytz em [gh-64020](#).)
- `sqlite3`:
 - Descontinua a passagem de mais de um argumento posicional para a função `connect()` e o construtor `Connection`. Os parâmetros restantes se tornarão somente-nomeados no Python 3.15. (Contribuição de Erlend E. Aasland em [gh-107948](#).)
 - Descontinua a passagem de nome, número de argumentos e o chamável como argumentos nomeados para `Connection.create_function()` e `Connection.create_aggregate()`. Esses parâmetros se tornarão somente-posicionais no Python 3.15. (Contribuição de Erlend E. Aasland em [gh-108278](#).)
 - Descontinua a passagem do chamável de função de retorno como argumento nomeado para os métodos `set_authorizer()`, `set_progress_handler()` e `set_trace_callback()` da classe `Connection`. Os chamáveis de função de retorno se tornarão somente-posicionais no Python 3.15. (Contribuição de Erlend E. Aasland em [gh-108278](#).)
- `sys`:
 - Descontinua a função `_enablelegacywindowsfsencoding()`, para ser removida no Python 3.16. Em vez disso, use a variável de ambiente `PYTHONLEGACYWINDOWSFSENCODING`. (Contribuição de Inada Naoki em [gh-73427](#).)
- `tarfile`:
 - Descontinua o atributo `TarFile.tarfile` não documentado e não utilizado, a ser removido no Python 3.16. (Contribuição em [gh-115256](#).)
- `traceback`:
 - Descontinua o atributo `TracebackException.exc_type`. Em vez disso, use `TracebackException.exc_type_str`. (Contribuição de Irit Katriel em [gh-112332](#).)
- `typing`:
 - Descontinua a não-documentada sintaxe de argumento nomeado para criação de classes `NamedTuple` (por exemplo, `Point = NamedTuple("Point", x=int, y=int)`), para ser removido no Python 3.15. Em vez disso, use as sintaxes baseada em classe ou a funcional. (Contribuição de Alex Waygood in [gh-105566](#).)
 - Descontinua a omissão do parâmetro *fields* ao criar uma classe `NamedTuple` ou `typing.TypedDict` e descontinua a passagem de `None` para o parâmetro *fields* de ambos os tipos. O Python 3.15 exigirá uma sequência válida para o parâmetro *fields*. Para criar uma classe `NamedTuple` com zero campos, use `class NT(NamedTuple): pass` ou `NT = NamedTuple("NT", ())`. Para criar uma classe `TypedDict` com zero campos, use `class TD(TypedDict): pass` ou `TD = TypedDict("TD", {})`. (Contribuição de Alex Waygood em [gh-105566](#) e [gh-105570](#).)
 - Descontinua o decorador de função `typing.no_type_check_decorator()` para ser removido no Python 3.15. Depois de oito anos no módulo `typing`, ele ainda não foi implementado por nenhum dos principais verificadores de tipos. (Contribuição de Alex Waygood em [gh-106309](#).)

- Descontinua `typing.AnyStr`. No Python 3.16, isso será removido de `typing.__all__`, e uma `DeprecationWarning` será levantada em tempo de execução quando isso for importado ou acessado. Será removido completamente em Python 3.18. Em vez disso, use a nova sintaxe de parâmetro de tipo. (Contribuição de Michael The em [gh-107116](#).)
- `wave`:
 - Descontinua os métodos `getmark()`, `setmark()` e `getmarkers()` das classes `Wave_read` e `Wave_write`, a serem removidos no Python 3.15. (Contribuição de Victor Stinner em [gh-105096](#).)

8.1 Remoção pendente no Python 3.14

- O sistema de importação:
 - A definição de `__loader__` em um módulo enquanto falha na definição de `__spec__.loader` está descontinuado. No Python 3.14, `__loader__` deixará de ser definido ou levado em consideração pelo sistema de importação ou pela biblioteca padrão.
- `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 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`

Em vez disso, use `ast.Constant`. (Contribuição de Serhiy Storchaka em [gh-90953](#).)

- `asyncio`:
 - As classes filhas dos observadores `MultiLoopChildWatcher`, `FastChildWatcher`, `AbstractChildWatcher` e `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](#).)
 - `collections.abc`: `ByteString` foi descontinuado. Prefira `Sequence` ou `Buffer` Para uso em tipagem, prefira uma união, como `bytes | bytearray` ou `collections.abc.Buffer`. (Contribuição de Shantanu Jain em [gh-91896](#).)
 - `email`: Descontinua o parâmetro `isdst` em `email.utils.localtime()`. (Contribuição de Alan Williams em [gh-72346](#).)
 - `importlib.abc` descontinuou as classes:
 - `importlib.abc.ResourceReader`
 - `importlib.abc.Traversable`
 - `importlib.abc.TraversableResources`
- Em vez disso, use classes de `importlib.resources.abc`:
- `importlib.resources.abc.Traversable`

- `importlib.resources.abc.TraversableResources`

(Contribuição de Jason R. Coombs e Hugo van Kemenade em [gh-93963](#).)

- `itertools` tinha suporte não documentado, ineficiente, historicamente cheio de bugs e inconsistente para operações de cópia, cópia profunda e serialização com `pickle`. 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`: o método de inicialização padrão 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 de 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 `multiprocessing-start-methods`.
- `pathlib`: `is_relative_to()` e `relative_to()`: passar argumentos adicionais foi descontinuado.
- `pkgutil`: `find_loader()` e `get_loader()` agora levantam `DeprecationWarning`; use `importlib.util.find_spec()`. (Contribuição de Nikita Sobolev em [gh-97850](#).)
- `pty`:
 - `master_open()`: use `pty.openpty()`.
 - `slave_open()`: use `pty.openpty()`.
- `sqlite3`:
 - `version` e `version_info`.
 - `execute()` e `executemany()` se espaços reservados nomeados forem usados e *parameters* for uma sequência em vez de um `dict`.
 - adaptador de data e hora, conversor de registro de data e hora: veja a documentação de `sqlite3` para receitas de substituição sugeridas.
- `types.CodeType`: o acesso a `co_notab` foi descontinuado na **PEP 626** desde 3.10 e foi planejado para ser removido em 3.12, mas só recebeu uma `DeprecationWarning` adequada em 3.12. Pode ser removido em 3.14. (Contribuição de Nikita Sobolev em [gh-101866](#).)
- `typing`: `ByteString`, descontinuado desde Python 3.9, agora faz com que uma `DeprecationWarning` seja emitida quando é usado.
- `urllib`: `urllib.parse.Quoter` está obsoleto: não foi planejado para ser uma API pública. (Contribuição de Gregory P. Smith em [gh-88168](#).)

8.2 Remoção pendente no Python 3.15

- O sistema de importação:
 - A definição de `__cached__` em um módulo enquanto falha na definição de `__spec__.loader` está descontinuado. No Python 3.15, `__cached__` deixará de ser definido ou levado em consideração pelo sistema de importação ou pela biblioteca padrão. ([gh-97879](#))
 - A definição de `__package__` em um módulo enquanto falha na definição de `__spec__.parent` está descontinuado. No Python 3.15, `__package__` deixará de ser definido ou levado em consideração pelo sistema de importação ou pela biblioteca padrão. ([gh-97879](#))
- `ctypes`:
 - A função não documentada `ctypes.SetPointerType()` foi descontinuada desde o Python 3.13.
- `http.server`:
 - A classe obsoleta e raramente usada `CGIHTTPRequestHandler` foi descontinuada desde o Python 3.13. Não existe substituição direta. *Qualquer coisa* é melhor que CGI para fazer a interface de um servidor web com um manipulador de requisição.

- O sinalizador `--cgi` para a interface de linha de comando `python -m http.server` foi descontinuado desde o Python 3.13.
- `locale`:
 - A função `getdefaultlocale()` foi descontinuada desde o Python 3.11. Sua remoção foi planejada originalmente para o Python 3.13 ([gh-90817](#)), mas foi adiada para o Python 3.15. Em vez disso, use `getlocale()`, `setlocale()` e `getencoding()`. (Contribuição de Hugo van Kemenade em [gh-111187](#).)
- `pathlib`:
 - `PurePath.is_reserved()` foi descontinuado desde o Python 3.13. Use `os.path.isreserved()` para detectar caminhos reservados no Windows.
- `platform`:
 - `java_ver()` foi descontinuada desde o Python 3.13. Esta função é útil apenas para suporte Jython, tem uma API confusa e é amplamente não testada.
- `threading`:
 - `RLock()` não aceitará argumentos no Python 3.15. A passagem quaisquer argumentos foi descontinuada desde o Python 3.14, pois a versão Python não permite nenhum argumento, mas a versão C permite qualquer número de argumentos posicionais ou nomeados, ignorando todos os argumentos.
- `typing`:
 - A não-documentada sintaxe de argumento nomeado para criar classes `NamedTuple` (por exemplo, `Point = NamedTuple("Point", x=int, y=int)`) foi descontinuada desde o Python 3.13. Em vez disso, use as sintaxes baseada em classe ou funcional.
 - A função decoradora `typing.no_type_check_decorator()` foi descontinuada desde o Python 3.13. Após oito anos no módulo `typing`, ela ainda não foi suportada por nenhum verificador de tipo importante.
- `wave`:
 - Os métodos `getmark()`, `setmark()` e `getmarkers()` das classes `Wave_read` e `Wave_write` foram descontinuados desde o Python 3.13.

8.3 Remoção pendente no Python 3.16

- `builtins`:
 - A inversão bit a bit em tipos booleanos, `~True` ou `~False` foi descontinuada desde o Python 3.12, pois produz resultados surpreendentes e não intuitivos (`-2` e `-1`). Em vez disso, use `not x` para a negação lógica de um booleano. No caso raro de você precisar da inversão bit a bit do inteiro subjacente, converta para `int` explicitamente (`~int(x)`).
- `array`:
 - O código de formato `'u'` (`wchar_t`) foi descontinuado na documentação desde o Python 3.3 e do ambiente de execução desde o Python 3.13. Em vez disso, use o código de formato `'w'` (`Py_UCS4`) para caracteres Unicode.
- `shutil`:
 - A exceção `ExecError` foi descontinuada desde o Python 3.14. Ela não foi usada por nenhuma função em `shutil` desde o Python 3.4, e agora é um alias de `RuntimeError`.
- `symtable`:
 - O método `Class.get_methods` foi descontinuado desde o Python 3.14.
- `sys`:
 - A função `_enablelegacywindowsfsencoding()` foi descontinuada desde o Python 3.13. Em vez disso, use a variável de ambiente `PYTHONLEGACYWINDOWSFSENCODING`.

- `tarfile`:
 - O atributo não documentado e não utilizado `TarFile.tarfile` foi descontinuado desde o Python 3.13.

8.4 Remoção pendente em versões futuras

As APIs a seguir serão removidas no futuro, embora atualmente não haja uma data agendada para sua remoção.

- `argparse`: o aninhamento de grupos de argumentos e o aninhamento de grupos mutuamente exclusivos estão descontinuados.
- código de formatação `'u'` do `array` ([gh-57281](#))
- `builtins`:
 - `bool(NotImplemented)`.
 - Geradores: a assinatura `throw(type, exc, tb)` e `athrow(type, exc, tb)` está descontinuada: use `throw(exc)` e `athrow(exc)`, a assinatura com um único argumento.
 - Atualmente Python aceita literais numéricos imediatamente seguidos por palavras reservadas como, por exemplo, `0in x, 1or x, 0if 1else 2`. Ele 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 imediatamente seguido por uma das palavras reservadas `and`, `else`, `for`, `if`, `in`, `is` e `or`. Em uma versão futura, será alterado para um erro de sintaxe. ([gh-87999](#))
 - Suporte para métodos `__index__()` e `__int__()` retornando tipo não-`int`: esses métodos serão necessários para retornar uma instância de uma subclasse estrita de `int`.
 - Suporte para o método `__float__()` retornando uma subclasse estrita de `float`: esses métodos serão necessários para retornar uma instância de `float`.
 - Suporte para o método `__complex__()` retornando uma subclasse estrita de `complex`: esses métodos serão necessários para retornar uma instância de `complex`.
 - Delegação do método `int()` para o `__trunc__()`.
 - Passar um número complexo como argumento *real* ou *imag* no construtor `complex()` agora está descontinuado; deve ser passado apenas como um único argumento posicional. (Contribuição de Serhiy Storchaka em [gh-109218](#).)
- `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](#).)
- `codeobject.co_lnotab`: use o método `codeobject.co_lines()`.
- `datetime`:
 - `utcnow()`: use `datetime.datetime.now(tz=datetime.UTC)`.
 - `utcfromtimestamp()`: use `datetime.datetime.fromtimestamp(timestamp, tz=datetime.UTC)`.
- `gettext`: o valor de plural deve ser um número inteiro.
- `importlib`:
 - Método `load_module()`: use `exec_module()`.
 - O parâmetro `debug_override` de `cache_from_source()` foi descontinuado: use o parâmetro `optimization`.
- `importlib.metadata`:
 - Interface de tupla `EntryPoints`.
 - `None` implícito nos valores de retorno.
- `logging`: o método `warn()` foi descontinuado desde o Python 3.3, use `warning()`.

- `mailbox`: o uso da entrada `StringIO` e do modo de texto foi descontinuado, use `BytesIO` e o modo binário.
- `os`: chamar `os.register_at_fork()` em processo multithread.
- `pydoc.ErrorDuringImport`: Um valor de tupla para o parâmetro `exc_info` foi descontinuado, use uma instância de exceção.
- `re`: 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](#).)
- Módulos `sre_compile`, `sre_constants` e `sre_parse`.
- `shutil`: o parâmetro `onerror` de `rmtree()` foi descontinuado no Python 3.12; use o parâmetro `onexc`.
- Protocolos e opções de `ssl`
 - `ssl.SSLContext` sem argumento de protocolo foi descontinuado.
 - `ssl.SSLContext`: `set_npn_protocols()` e `selected_npn_protocol()` foram descontinuados, use ALPN.
 - Opções de `ssl.OP_NO_SSL*`
 - Opções de `ssl.OP_NO_TLS*`
 - `ssl.PROTOCOL_SSLv3`
 - `ssl.PROTOCOL_TLS`
 - `ssl.PROTOCOL_TLSv1`
 - `ssl.PROTOCOL_TLSv1_1`
 - `ssl.PROTOCOL_TLSv1_2`
 - `ssl.TLSVersion.SSLv3`
 - `ssl.TLSVersion.TLSv1`
 - `ssl.TLSVersion.TLSv1_1`
- O parâmetro `check_home` de `sysconfig.is_python_build()` foi descontinuado e é ignorado.
- Métodos de `threading`:
 - `threading.Condition.notifyAll()`: use `notify_all()`.
 - `threading.Event.isSet()`: use `is_set()`.
 - `threading.Thread.isDaemon()`, `threading.Thread.setDaemon()`: use o atributo `threading.Thread.daemon`.
 - `threading.Thread.getName()`, `threading.Thread.setName()`: use o atributo `threading.Thread.name`.
 - `threading.currentThread()`: use `threading.current_thread()`.
 - `threading.activeCount()`: use `threading.active_count()`.
- `typing.Text` ([gh-92332](#)).
- `unittest.IsolatedAsyncioTestCase`: foi descontinuado retornar um valor que não seja `None` de um caso de teste.
- Funções descontinuadas de `urllib.parse`: use `urlparse()`
 - `splitattr()`
 - `splithost()`
 - `splitnport()`
 - `splitpasswd()`

- `splitport()`
 - `splitquery()`
 - `splittag()`
 - `splitttype()`
 - `splituser()`
 - `splitvalue()`
 - `to_bytes()`
- `urllib.request`: o estilo de `URLopener` e `FancyURLopener` de invocar solicitações foi descontinuado. Use as mais novas funções e métodos `urlopen()`.
 - `wsgiref`: `SimpleHandler.stdout.write()` não deve fazer gravações parciais.
 - `xml.etree.ElementTree`: testar o valor verdade de um `Element` está descontinuado. Em um lançamento futuro isso sempre retornará `True`. Em vez disso, prefira os testes explícitos `len(elem)` ou `elem is not None`.
 - `zipimport.zipimporter.load_module()` foi descontinuado: use `exec_module()`.

9 Alterações de bytecode do CPython

- O oparg de `YIELD_VALUE` agora é 1 se o `yield` fizer parte de um `yield-from` ou um `await`, e 0 caso contrário. O oparg de `RESUME` foi alterado para adicionar um bit indicando se a profundidade de exceção é 1, o que é necessário para otimizar o fechamento dos geradores. (Contribuição de Irit Katriel em [gh-111354](#).)

10 Alterações na API C

10.1 Novas funcionalidades

- Adiciona a API C `PyMonitoring C` para gerar eventos de monitoramento da **PEP 669**:

- `PyMonitoringState`
- `PyMonitoring_FirePyStartEvent()`
- `PyMonitoring_FirePyResumeEvent()`
- `PyMonitoring_FirePyReturnEvent()`
- `PyMonitoring_FirePyYieldEvent()`
- `PyMonitoring_FireCallEvent()`
- `PyMonitoring_FireLineEvent()`
- `PyMonitoring_FireJumpEvent()`
- `PyMonitoring_FireBranchEvent()`
- `PyMonitoring_FireCReturnEvent()`
- `PyMonitoring_FirePyThrowEvent()`
- `PyMonitoring_FireRaiseEvent()`
- `PyMonitoring_FireCRaiseEvent()`
- `PyMonitoring_FireReraiseEvent()`
- `PyMonitoring_FireExceptionHandledEvent()`
- `PyMonitoring_FirePyUnwindEvent()`
- `PyMonitoring_FireStopIterationEvent()`

- `PyMonitoring_EnterScope()`
- `PyMonitoring_ExitScope()`

(Contribuição de Irit Katriel em [gh-111997](#)).

- Adiciona `PyMutex`, um mutex leve que ocupa um único byte, e as novas funções `PyMutex_Lock()` e `PyMutex_Unlock()`. `PyMutex_Lock()` irá liberar a GIL (se atualmente mantida) se a operação precisar de bloqueio. (Contribuição de Sam Gross em [gh-108724](#).)

- Adiciona a API C `PyTime`, fornecendo acesso aos relógios do sistema.

- `PyTime_t`.
- `PyTime_MIN` e `PyTime_MAX`.
- `PyTime_AsSecondsDouble()`.
- `PyTime_Monotonic()`.
- `PyTime_MonotonicRaw()`.
- `PyTime_PerfCounter()`.
- `PyTime_PerfCounterRaw()`.
- `PyTime_Time()`.
- `PyTime_TimeRaw()`.

(Contribuição de Victor Stinner e Petr Viktorin em [gh-110850](#).)

- Adiciona a função `PyDict_ContainsString()` com o mesmo comportamento de `PyDict_Contains()`, mas `key` é especificada como uma string de bytes `const char*` codificada em UTF-8, em vez de um `PyObject*`. (Contribuição de Victor Stinner em [gh-108314](#).)
- Adiciona as funções `PyDict_GetItemRef()` e `PyDict_GetItemStringRef()` functions, que são semelhantes a `PyDict_GetItemWithError()`, mas retornam uma referência forte em vez de uma referência emprestada. Além disso, essas funções retornam `-1` em caso de erro, dispensando a necessidade de checar `PyErr_Occurred()`. (Contribuição de Victor Stinner em [gh-106004](#).)
- Adiciona a função `PyDict_SetDefaultRef()`, que é semelhante a `PyDict_SetDefault()`, mas retorna uma referência forte em vez de uma referência emprestada. Esta função retorna `-1` em caso de erro, `0` na inserção e `1` se a chave já estivesse presente no dicionário. (Contribuição de Sam Gross em [gh-112066](#).)
- Adiciona as funções `PyDict_Pop()` e `PyDict_PopString()` para remover uma chave de um dicionário e, opcionalmente, retornam o valor removido. Isto é semelhante a `dict.pop()`, mas sem o valor padrão e sem levantar `KeyError` se a chave estiver ausente. (Contribuição de Stefan Behnel e Victor Stinner em [gh-111262](#).)
- Adiciona as funções `PyMapping_GetOptionalItem()` e `PyMapping_GetOptionalItemString()` como alternativas a `PyObject_GetAttr()` e `PyObject_GetAttrString()`, respectivamente. As novas funções não levantam `KeyError` se o atributo não for encontrado no mapeamento. Essas variantes são mais convenientes e rápidas se o atributo ausente não for tratado como uma falha. (Contribuição de Serhiy Storchaka em [gh-106307](#).)
- Adiciona as funções `PyObject_GetOptionalAttr()` e `PyObject_GetOptionalAttrString()`, variantes de `PyObject_GetAttr()` e `PyObject_GetAttrString()`, respectivamente. Essas novas funções não levantam `AttributeError` se o atributo não for encontrado. Essas variantes são mais convenientes e rápidas se o atributo ausente não for tratado como uma falha. (Contribuição de Serhiy Storchaka em [gh-106521](#).)
- Adiciona a função `PyErr_FormatUnraisable()` como uma extensão para `PyErr_WriteUnraisable()`, que permite personalizar a mensagem de aviso. (Contribuição de Serhiy Storchaka em [gh-108082](#).)
- Adiciona novas funções que retornam uma referência forte em vez de uma referência emprestada para quadros locais, globais, e embutidos, como parte de [PEP 667](#):
 - `PyEval_GetFrameBuiltins()` substitui `PyEval_GetBuiltins()`
 - `PyEval_GetFrameGlobals()` substitui `PyEval_GetGlobals()`

- `PyEval_GetFrameLocals()` substitui `PyEval_GetLocals()`

(Contribuição de Mark Shannon e Tian Gao em [gh-74929](#).)

- Adiciona as funções `Py_GetConstant()` e `Py_GetConstantBorrowed()` para referências fortes ou emprestadas para constantes. Por exemplo, `Py_GetConstant(Py_CONSTANT_ZERO)` retorna uma referência forte para a constante zero. (Contribuição de Victor Stinner em [gh-115754](#).)
- Adiciona a função `PyImport_AddModuleRef()` como uma substituta para `PyImport_AddModule()` mas retorna uma referência forte em vez de uma referência emprestada. (Contribuição de Victor Stinner em [gh-105922](#).)
- Adiciona a função `Py_IsFinalizing()` para verificar se o interpretador Python principal está sendo desligado. (Contribuição de Victor Stinner em [gh-108014](#).)
- Adiciona a função `PyList_GetItemRef()` como uma substituta para `PyList_GetItem()` mas que retorna uma referência forte em vez de uma referência emprestada. (Contribuição de Sam Gross em [gh-114329](#).)
- Adiciona as funções `PyList_Extend()` e `PyList_Clear()`, espelhando os métodos `list.extend()` e `list.clear()` do Python. (Contribuição de Victor Stinner em [gh-111138](#).)
- Adiciona a função `PyLong_AsInt()`, semelhante à `PyLong_AsLong()`, mas armazena o resultado em um `int` em vez de um `long` em C. (Contribuição de Victor Stinner em [gh-108014](#).)
- Adiciona as funções `PyLong_AsNativeBytes()`, `PyLong_FromNativeBytes()` e `PyLong_FromUnsignedNativeBytes()` para simplificar a conversão entre tipos inteiros nativos e objetos `int` do Python. (Contribuição de Steve Dower em [gh-111140](#).)
- Adiciona a função `PyModule_Add()`, que é semelhante a `PyModule_AddObjectRef()` e `PyModule_AddObject()`, mas sempre rouba uma referência ao valor. (Contribuição de Serhiy Storchaka em [gh-86493](#).)
- Adiciona a função `PyObject_GenericHash()` que implementa a função hashing padrão de um objeto Python. (Contribuição de Serhiy Storchaka em [gh-113024](#).)
- Adiciona a função `Py_HashPointer()` para fazer hash de um ponteiro. (Contribuição de Victor Stinner em [gh-111545](#).)
- Adiciona as funções `PyObject_VisitManagedDict()` e `PyObject_ClearManagedDict()` que devem ser chamadas pelas funções `traverse` e `clear` de um tipo usando o sinalizador `Py_TPFLAGS_MANAGED_DICT`. O projeto [pythoncapi-compat](#) pode ser usado para obter essas funções no Python 3.11 e 3.12. (Contribuição de Victor Stinner em [gh-107073](#).)
- Adiciona as funções `PyRefTracer_SetTracer()` e `PyRefTracer_GetTracer()`, que permitem rastrear a criação e destruição de objetos da mesma forma que o módulo `tracemalloc`. (Contribuição de Pablo Galindo em [gh-93502](#).)
- Adiciona a função `PySys_AuditTuple()` como uma alternativa à `PySys_Audit()`, que recebe argumentos de evento como um objeto `tuple` do Python. (Contribuição de Victor Stinner em [gh-85283](#).)
- Adiciona a função `PyThreadState_GetUnchecked()` como uma alternativa a `PyThreadState_Get()` que não mata o processo com um erro fatal se for `NULL`. O chamador é responsável por verificar se o resultado é `NULL`. (Contribuição de Victor Stinner em [gh-108867](#).)
- Adiciona a função `PyType_GetFullyQualifiedName()` para obter o nome totalmente qualificado do tipo. O nome do módulo é prefixado se `type.__module__` for uma string e não for igual a `'builtins'` ou `'__main__'`. (Contribuição de Victor Stinner em [gh-111696](#).)
- Adiciona a função `PyType_GetModuleName()` para obter o nome do módulo do tipo. Isto é equivalente a obter o atributo `type.__module__`. (Contribuição de Eric Snow e Victor Stinner em [gh-111696](#).)
- Adiciona as funções `PyUnicode_EqualToUTF8AndSize()` e `PyUnicode_EqualToUTF8()`: comparam o objeto `Unicode` com uma string `const char*` codificada em UTF-8 e retornam 1 se forem iguais ou 0 caso contrário. Essas funções não levantam exceções. (Contribuição de Serhiy Storchaka em [gh-110289](#).)
- Adiciona a função `PyWeakref_GetRef()` como uma alternativa a `PyWeakref_GetObject()`, mas retorna uma referência forte, ou `NULL` se o referente não mais existir. (Contribuição de Victor Stinner em [gh-105927](#).)

- Adiciona variantes fixas de funções que ignoram erros silenciosamente:
 - `PyObject_HasAttrWithError()` substitui `PyObject_HasAttr()`.
 - `PyObject_HasAttrStringWithError()` substitui `PyObject_HasAttrString()`.
 - `PyMapping_HasKeyWithError()` substitui `PyMapping_HasKey()`.
 - `PyMapping_HasKeyStringWithError()` substitui `PyMapping_HasKeyString()`.

As novas funções retornam `-1` para erros e o padrão `1` para verdadeiro e `0` para falso.

(Contribuição de Serhiy Storchaka em [gh-108511](#).)

10.2 Alterações na API C

- O parâmetro `keywords` das funções `PyArg_ParseTupleAndKeywords()` e `PyArg_VaParseTupleAndKeywords()` agora é do tipo `char *const*` em C e `const char *const*` em C++, em vez de `char**`. Em C++, isso torna essas funções compatíveis com argumentos dos tipos `const char *const*`, `const char**`, ou `char *const*` sem uma conversão de tipo explícita. Em C, as funções aceitam apenas argumentos do tipo `char *const*`. Isso pode ser sobrescrito com a macro `PY_CXX_CONST`. (Contribuição de Serhiy Storchaka em [gh-65210](#).)
- `PyArg_ParseTupleAndKeywords()` agora oferece suporte a nomes de parâmetros nomeados não-ASCII. (Contribuição de Serhiy Storchaka em [gh-110815](#).)
- A função `PyCode_GetFirstFree()` é agora uma API instável e foi renomeada para `PyUnstable_Code_GetFirstFree()`. (Contribuição de Bogdan Romanyuk em [gh-115781](#).)
- As funções `PyDict_GetItem()`, `PyDict_GetItemString()`, `PyMapping_HasKey()`, `PyMapping_HasKeyString()`, `PyObject_HasAttr()`, `PyObject_HasAttrString()`, e `PySys_GetObject()`, que apagam todos os erros que ocorrem ao chamá-las, agora reportam tais erros usando `sys.unraisablehook()`. Você pode substituí-las por outras funções conforme recomendado na documentação. (Contribuição de Serhiy Storchaka em [gh-106672](#).)
- Adiciona suporte para os formatos `%T`, `%#T`, `%N` e `%#N` a `PyUnicode_FromFormat()`:
 - `%T`: Obtém o nome totalmente qualificado de um tipo de objeto
 - `%#T`: Como acima, mas usa dois pontos como separador
 - `%N`: Obtém o nome totalmente qualificado de um tipo
 - `%#N`: Como acima, mas usa dois pontos como separador

Veja [PEP 737](#) para uma descrição completa. (Contribuição de Victor Stinner em [gh-111696](#).)

- Você não precisa mais definir a macro `PY_SSIZE_T_CLEAN` antes de incluir `Python.h` ao usar formatos `#` em códigos de formato. APIs que aceitam os códigos de formato sempre usam `Py_ssize_t` para formatos `#`. (Contribuição de Inada Naoki em [gh-104922](#).)
- Se Python for construído em modo de depuração ou com asserções, `PyTuple_SET_ITEM()` e `PyList_SET_ITEM()` agora verificam o argumento do índice com uma asserção. (Contribuição de Victor Stinner em [gh-106168](#).)

10.3 Mudanças na API C Limitada

- As seguintes funções agora estão incluídas na API C limitada:
 - `PyMem_RawMalloc()`
 - `PyMem_RawCalloc()`
 - `PyMem_RawRealloc()`
 - `PyMem_RawFree()`
 - `PySys_Audit()`
 - `PySys_AuditTuple()`

- `PyType_GetModuleByDef()`

(Contribuição de Victor Stinner em [gh-85283](#), [gh-85283](#) e [gh-116936](#).)

- Python construído com `--with-trace-refs` (referências de rastreamento) agora oferece suporte à API Limitada. (Contribuição de Victor Stinner em [gh-108634](#).)

10.4 APIs C removidas

- Remove diversas funções, macros, variáveis, etc com nomes prefixados por `_Py` ou `_PY` (consideradas como API privada). Se o seu projeto for afetado por uma dessas remoções e você acreditar que a API removida deve permanecer disponível, por favor abra um novo relatório de problema para solicitar uma API C pública e adicione `cc: @vstinner` ao problema para notificar Victor Stinner. (Contribuição de Victor Stinner em [gh-106320](#).)

- Remove protocolos de buffer antigos descontinuados no Python 3.0: use `bufferobjects`.

- `PyObject_CheckReadBuffer()`: use `PyObject_CheckBuffer()` para testar se o objeto oferece suporte ao protocolo de buffer. Observe que `PyObject_CheckBuffer()` não garante que `PyObject_GetBuffer()` terá sucesso. Para testar se o objeto é realmente legível, veja o próximo exemplo de `PyObject_GetBuffer()`.

- `PyObject_AsCharBuffer()`, `PyObject_AsReadBuffer()`: use `PyObject_GetBuffer()` e `PyBuffer_Release()`:

```
Py_buffer view;
if (PyObject_GetBuffer(obj, &view, PyBUF_SIMPLE) < 0) {
    return NULL;
}
// Usa `view.buf` e `view.len` para ler do buffer.
// Você pode precisar converter buf como `(const char*)view.buf`.
PyBuffer_Release(&view);
```

- `PyObject_AsWriteBuffer()`: use `PyObject_GetBuffer()` e `PyBuffer_Release()`:

```
Py_buffer view;
if (PyObject_GetBuffer(obj, &view, PyBUF_WRITABLE) < 0) {
    return NULL;
}
// Usa `view.buf` e `view.len` para gravar no buffer.
PyBuffer_Release(&view);
```

(Contribuição de Inada Naoki in [gh-85275](#).)

- Remove diversas funções descontinuadas no Python 3.9:

- `PyEval_CallObject()`, `PyEval_CallObjectWithKeywords()`: use `PyObject_CallNoArgs()` ou `PyObject_Call()`.

Aviso

Os argumentos posicionais de `PyObject_Call()` devem ser uma tuple e não podem ser NULL, os argumentos nomeados devem ser dict ou NULL, enquanto funções removidas verificavam o tipo de argumentos e aceitavam argumentos posicionais e nomeados NULL. Para substituir `PyEval_CallObjectWithKeywords(func, NULL, kwargs)` por `PyObject_Call()`, passe uma tupla vazia como argumentos posicionais usando `PyTuple_New(0)`.

- `PyEval_CallFunction()`: use `PyObject_CallFunction()`.

- `PyEval_CallMethod()`: use `PyObject_CallMethod()`.

- `PyCFunction_Call()`: use `PyObject_Call()`.

(Contribuição de Victor Stinner em [gh-105107](#).)

- Remove as seguintes funções antigas da configuração de inicialização do Python, descontinuadas no Python 3.11:
 - `PySys_AddWarnOptionUnicode()`: use `PyConfig.warnoptions`.
 - `PySys_AddWarnOption()`: use `PyConfig.warnoptions`.
 - `PySys_AddXOption()`: use `PyConfig.xoptions`.
 - `PySys_HasWarnOptions()`: use `PyConfig.xoptions`.
 - `PySys_SetPath()`: defina `PyConfig.module_search_paths`.
 - `Py_SetPath()`: defina `PyConfig.module_search_paths`.
 - `Py_SetStandardStreamEncoding()`: defina `PyConfig.stdio_encoding`, e também talvez `PyConfig.legacy_windows_stdio` (no Windows).
 - `_Py_SetProgramFullPath()`: defina `PyConfig.executable`.

Em vez disso, use a nova API `PyConfig` da Configuração de Inicialização do Python ([PEP 587](#)), adicionada ao Python 3.8. (Contribuição de Victor Stinner em [gh-105145](#).)

- Remove as funções `PyEval_AcquireLock()` e `PyEval_ReleaseLock()`, descontinuadas no Python 3.2. Eles não atualizavam o estado atual da thread. Elas podem ser substituídas por:
 - `PyEval_SaveThread()` e `PyEval_RestoreThread()`;
 - `PyEval_AcquireThread()` e `PyEval_RestoreThread()` de baixo nível;
 - ou `PyGILState_Ensure()` e `PyGILState_Release()`.

(Contribuição de Victor Stinner em [gh-105182](#).)

- Remove a função `PyEval_ThreadsInitialized()`, descontinuada no Python 3.9. Desde Python 3.7, `Py_Initialize()` sempre cria a GIL: chamar `PyEval_InitThreads()` não faz nada e `PyEval_ThreadsInitialized()` sempre retorna diferente de zero. (Contribuição de Victor Stinner em [gh-105182](#).)
- Remove o apelido `_PyInterpreterState_Get()` de `PyInterpreterState_Get()` que era mantido para compatibilidade com versões anteriores do Python 3.8. O projeto [pythoncapi-compat](#) pode ser usado para obter `PyInterpreterState_Get()` no Python 3.8 e versões anteriores. (Contribuição de Victor Stinner em [gh-106320](#).)
- Remove a função privada `_PyObject_FastCall()`: use `PyObject_Vectorcall()`, a qual está disponível desde Python 3.8 ([PEP 590](#)). (Contribuição de Victor Stinner em [gh-106023](#).)
- Remove o arquivo de cabeçalho `cpython/pytime.h`, que continha apenas funções privadas. (Contribuição de Victor Stinner em [gh-106316](#).)
- Remove a constante não documentada `PY_TIMEOUT_MAX` da API C limitada. (Contribuição de Victor Stinner em [gh-110014](#).)
- Remove as antigas macros de trashcan `Py_TRASHCAN_SAFE_BEGIN` e `Py_TRASHCAN_SAFE_END`. Substitua ambas pelas novas macros `Py_TRASHCAN_BEGIN` e `Py_TRASHCAN_END`. (Contribuição de Irit Katriel em [gh-105111](#).)

10.5 APIs C descontinuadas

- Descontinua antigas funções de inicialização do Python:
 - `PySys_ResetWarnOptions()`: apague `sys.warnoptions` e `warnings.filters`.
 - `Py_GetExecPrefix()`: leia `sys.exec_prefix`.
 - `Py_GetPath()`: leia `sys.path`.
 - `Py_GetPrefix()`: leia `sys.prefix`.

- `Py_GetProgramFullPath()`: leia `sys.executable`.
- `Py_GetProgramName()`: leia `sys.executable`.
- `Py_GetPythonHome()`: leia `PyConfig.home` ou a variável de ambiente `PYTHONHOME`.

(Contribuição de Victor Stinner em [gh-105145](#).)

- Descontinua suavemente as funções `PyEval_GetBuiltins()`, `PyEval_GetGlobals()` e `PyEval_GetLocals()`, que retornam uma referência emprestada. (Suavemente descontinuadas como parte da [PEP 667](#).)
- Descontinua a função `PyImport_ImportModuleNoBlock()` que é apenas um apelido para `PyImport_ImportModule()` desde o Python 3.3. (Contribuição de Victor Stinner em [gh-105396](#).)
- Descontinua suavemente a função `PyModule_AddObject()`. Ela deve ser substituída por `PyModule_Add()` ou `PyModule_AddObjectRef()`. (Contribuição de Serhiy Storchaka em [gh-86493](#).)
- Descontinua os tipos antigos `Py_UNICODE` e `PY_UNICODE_TYPE` e define `Py_UNICODE_WIDE`. Em vez disso, use o tipo `wchar_t` diretamente. Desde Python 3.3, `Py_UNICODE` e `PY_UNICODE_TYPE` são apenas apelidos para `wchar_t`. (Contribuição de Victor Stinner em [gh-105156](#).)
- Descontinua as funções `PyWeakref_GetObject()` e `PyWeakref_GET_OBJECT()`, que retornam uma referência emprestada. Em vez disso, use a nova função `PyWeakref_GetRef()`, que retorna uma referência forte. O projeto [pythoncapi-compat](#) pode ser usado para obter `PyWeakref_GetRef()` no Python 3.12 e versões anteriores. (Contribuição de Victor Stinner em [gh-105927](#).)

Remoção pendente no Python 3.14

- O campo `ma_version_tag` em `PyDictObject` para módulos de extensão ([PEP 699](#); [gh-101193](#)).
- A criação de tipos imutáveis com bases mutáveis ([gh-95388](#)).
- Funções para configurar a inicialização do Python, descontinuadas no Python 3.11:
 - `PySys_SetArgvEx()`: defina `PyConfig.argv`.
 - `PySys_SetArgv()`: defina `PyConfig.argv`.
 - `Py_SetProgramName()`: defina `PyConfig.program_name`.
 - `Py_SetPythonHome()`: defina `PyConfig.home`.

Em vez disso, a API `Py_InitializeFromConfig()` deve ser usada com `PyConfig`.

- 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` e `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`. (veja `Py_PreInitialize()`)

Em vez disso, a API `Py_InitializeFromConfig()` deve ser usada com `PyConfig`.

Remoção pendente no Python 3.15

- A cópia empacotada do `libmpdecimal`.
- `PyImport_ImportModuleNoBlock()`: **use** `PyImport_ImportModule()`.
- `PyWeakref_GetObject()` e `PyWeakref_GET_OBJECT()`: **use** `PyWeakref_GetRef()`.
- O tipo `Py_UNICODE` e a macro `Py_UNICODE_WIDE`: **use** `wchar_t`.
- Funções de inicialização do Python
 - `PySys_ResetWarnOptions()`: **apague** `sys.warnoptions` e `warnings.filters`.
 - `Py_GetExecPrefix()`: **obtenha** `sys.base_exec_prefix` e `sys.exec_prefix`.
 - `Py_GetPath()`: **leia** `sys.path`.
 - `Py_GetPrefix()`: **obtenha** `sys.base_prefix` e `sys.prefix`.
 - `Py_GetProgramFullPath()`: **leia** `sys.executable`.
 - `Py_GetProgramName()`: **leia** `sys.executable`.
 - `Py_GetPythonHome()`: **leia** `PyConfig.home` ou a variável de ambiente `PYTHONHOME`.

Remoção pendente em versões futuras

As APIs a seguir foram descontinuadas 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()`.
- O membro `PyBytesObject.ob_shash`: chame `PyObject_Hash()`.
- O 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.

11 Mudanças na construção

- `arm64-apple-ios` e `arm64-apple-ios-simulator` são agora plataformas de nível 3 da **PEP 11**. (Escrita e implementação da **PEP 730** foi uma contribuição de Russell Keith-Magee em [gh-114099](#).)
- `aarch64-linux-android` e `x86_64-linux-android` são agora plataformas de nível 3 da **PEP 11**. (Escrita e implementação da **PEP 738** foi uma contribuição de Malcolm Smith em [gh-116622](#).)
- `wasm32-wasi` agora é uma plataforma de nível 2 segundo a **PEP 11**. (Contribuição de Brett Cannon em [gh-115192](#).)
- `wasm32-emscrip` não é mais uma plataforma suportada pela **PEP 11**. (Contribuição de Brett Cannon em [gh-115192](#).)
- Construir CPython agora requer um compilador com suporte para a biblioteca atômica C11, funções atômicas embutidas do GCC ou intrínsecos interligados MSVC.
- `Autoconf 2.71` e `aclocal 1.16.5` agora são necessários para regerar o script `configure`. (Contribuição de Christian Heimes em [gh-89886](#) e de Victor Stinner em [gh-112090](#).)
- `SQLite 3.15.2` ou mais recente é necessário para construir o módulo de extensão `sqlite3`. (Contribuição de Erlend Aasland em [gh-105875](#).)
- CPython agora inclui a biblioteca `mimalloc` por padrão. Está licenciada sob a licença do MIT; veja licença do `mimalloc`. O `mimalloc` incluído tem alterações personalizadas, veja [gh-113141](#) para detalhes. (Contribuição de Dino Viehland em [gh-109914](#).)
- A opção `--with-system-libmpdec` do `configure` agora tem como padrão `yes`. A cópia empacotada de `libmpdecimal` será removida no Python 3.15.
- Python construído com `--with-trace-refs` (referências de rastreamento) do `configure` tem compatibilidade de ABI com a construção de lançamento do Python e a construção de depuração. (Contribuição de Victor Stinner em [gh-108634](#).)
- Em sistemas POSIX, os nomes de arquivos `pkg-config(.pc)` agora incluem os sinalizadores de ABI. Por exemplo, a construção com threads livres gera `python-3.13t.pc` e a construção de depuração gera `python-3.13d.pc`.
- As extensões `C` `errno`, `fcntl`, `grp`, `md5`, `pwd`, `resource`, `termios`, `winsound`, `_ctypes_test`, `_multiprocessing.posixshm`, `_scproxy`, `_stat`, `_statistics`, `_testconsole`, `_testimportmultiple` e `_uuid` são agora construídas com a API C limitada. (Contribuição de Victor Stinner em [gh-85283](#).)

12 Portando para o Python 3.13

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

12.1 Alterações na API Python

- **PEP 667** introduz várias mudanças na semântica de `locals()` e `f_locals`:
 - Chamar `locals()` em um escopo otimizado agora produz uma captura independente em cada chamada e, portanto, não atualiza mais implicitamente as referências retornadas anteriormente. Obter o comportamento legado do CPython agora requer chamadas explícitas para atualizar o dicionário inicialmente retornado com os resultados de chamadas subsequentes para `locals()`. Funções de execução de código que visam implicitamente `locals()` (como `exec` e `eval`) devem receber um espaço de nomes explícito para acessar seus resultados em um escopo otimizado. (Alterado como parte da **PEP 667**.)
 - Chamar `locals()` de uma compreensão no módulo ou escopo de classe (inclusive via `exec` ou `eval`) mais uma vez se comporta como se a compreensão estivesse sendo executada como uma função aninhada independente (isto é, as variáveis locais de o escopo que contém não está incluído). No Python 3.12, isso mudou para incluir as variáveis locais do escopo que o contém ao implementar a **PEP 709**. (Alterado como parte da **PEP 667**.)
 - Acessar `FrameType.f_locals` em um escopo otimizado agora retorna um proxy de “write-through” em vez de uma captura que é atualizado em horários mal especificados. Se uma captura for desejada, ela deve ser criada explicitamente com `dict` ou com o método `.copy()` do proxy. (Alterado como parte da **PEP 667**.)
- `functools.partial` agora emite uma `FutureWarning` quando é usado como método. Seu comportamento será alterado em versões futuras do Python. Envolve-a em `staticmethod()` se quiser preservar o comportamento antigo. (Contribuição de Serhiy Storchaka em [gh-121027](#).)
- Uma exceção `OSError` agora é levantada por `getpass.getuser()` para qualquer falha na recuperação de um nome de usuário, em vez de `ImportError` em plataformas não-Unix ou `KeyError` em plataformas Unix onde o banco de dados de senhas está vazio.
- O valor do atributo `mode` de `gzip.GzipFile` agora é uma string (`'rb'` ou `'wb'`) e não mais um inteiro (1 ou 2). O valor do atributo `mode` do objeto arquivo ou similar legível retornado por `zipfile.ZipFile.open()` é agora `'rb'` e não mais `'r'`. (Contribuição de Serhiy Storchaka em [gh-115961](#).)
- `mailbox.Maildir` agora ignora arquivos com um ponto inicial (`.`). (Contribuição de Zackery Spytz em [gh-65559](#).)
- `pathlib.Path.glob()` e `rglob()` agora retornam arquivos e diretórios se um padrão que termina com `“**”` for fornecido, em vez de apenas diretórios. Os usuários podem adicionar uma barra final para manter o comportamento anterior e corresponder apenas aos diretórios.
- O módulo `threading` agora espera que o módulo `_thread` tenha uma função `_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 obrigatoriamente fornecer `_is_main_interpreter()`, da mesma forma que outros atributos “privados” do módulo. (Veja [gh-112826](#).)

12.2 Alterações na API C

- `Python.h` não inclui mais o cabeçalho padrão `<ieee754.h>`. Ele foi incluído para a função `finite()` que agora é fornecida pelo cabeçalho `<math.h>`. Agora deve ser incluído explicitamente, se necessário. Remove também a macro `HAVE_IEEEFP_H`. (Contribuição de Victor Stinner em [gh-108765](#).)
- `Python.h` não inclui mais estes arquivos de cabeçalho padrão: `<time.h>`, `<sys/select.h>` e `<sys/time.h>`. Se necessário, deverão agora ser incluídos explicitamente. Por exemplo, `<time.h>` fornece as funções `clock()` e `gmtime()`, `<sys/select.h>` fornece o `select()`, e `<sys/time.h>` fornece as funções `futimes()`, `gettimeofday()` e `setitimer()`. (Contribuição de Victor Stinner em [gh-108765](#).)

- No Windows, `Python.h` não inclui mais o arquivo de cabeçalho padrão `<stddef.h>`. Se necessário, deverá agora ser incluído explicitamente. Por exemplo, ele fornece a função `offsetof()` e os tipos `size_t` e `ptrdiff_t`. Incluindo `<stddef.h>` explicitamente já era necessário para todas as outras plataformas, a macro `HAVE_STDDEF_H` só é definida no Windows. (Contribuição de Victor Stinner em [gh-108765](#).)
- Se a macro `Py_LIMITED_API` estiver definida, as macros `Py_BUILD_CORE`, `Py_BUILD_CORE_BUILTIN` e `Py_BUILD_CORE_MODULE` agora são indefinidas por `<Python.h>`. (Contribuição de Victor Stinner em [gh-85283](#).)
- As macros antigas da lixeira `Py_TRASHCAN_SAFE_BEGIN` e `Py_TRASHCAN_SAFE_END` foram removidas. Elas devem ser substituídas pelas novas macros `Py_TRASHCAN_BEGIN` e `Py_TRASHCAN_END`.

Uma função `tp_dealloc` que contém as macros antigas, como:

```
static void
deallocador_de_meutipo(meutipo *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_SAFE_BEGIN(p);
    ...
    Py_TRASHCAN_SAFE_END
}
```

deve migrar para as novas macros da seguinte forma:

```
static void
deallocador_de_meutipo(meutipo *p)
{
    PyObject_GC_UnTrack(p);
    Py_TRASHCAN_BEGIN(p, deallocador_de_meutipo);
    ...
    Py_TRASHCAN_END
}
```

Observe que `Py_TRASHCAN_BEGIN` tem um segundo argumento que deve ser a função de desalocação em que está. As novas macros foram adicionadas no Python 3.8 e as macros antigas foram descontinuadas no Python 3.11. (Contribuição de Irit Katriel em [gh-105111](#).)

- [PEP 667](#) introduz várias mudanças em funções relacionadas a quadro:
 - Os efeitos da mutação do dicionário retornado de `PyEval_GetLocals()` em um escopo otimizado foram alterados. Novas entradas de dict adicionadas desta forma agora *apenas* ficarão visíveis para chamadas `PyEval_GetLocals()` subsequentes nesse quadro, como `PyFrame_GetLocals()`, `locals()` e `FrameType.f_locals` não acessa mais o mesmo dicionário armazenado em cache subjacente. As alterações feitas nas entradas para nomes de variáveis reais e nomes adicionados através das interfaces proxy de “write-through” serão substituídas em chamadas subsequentes para `PyEval_GetLocals()` nesse quadro. A atualização de código recomendada depende de como a função estava sendo usada, portanto, consulte o aviso de descontinuação na função para obter detalhes.
 - Chamar `PyFrame_GetLocals()` em um escopo otimizado agora retorna um proxy de “write-through” em vez de uma captura que é atualizada em horários mal especificados. Se uma captura for desejada, ela deve ser criada explicitamente (por exemplo, com `PyDict_Copy()`) ou chamando a nova API `PyEval_GetFrameLocals()`.
 - `PyFrame_FastToLocals()` e `PyFrame_FastToLocalsWithError()` não têm mais nenhum efeito. Chamar essas funções tem sido redundante desde o Python 3.11, quando `PyFrame_GetLocals()` foi introduzido pela primeira vez.
 - `PyFrame_LocalsToFast()` não tem mais efeito. Chamar esta função é redundante agora que `PyFrame_GetLocals()` retorna um proxy de “write-through” para escopos otimizados.

13 Mudanças em teste de regressão

- Python construído com `--with-pydebug` do `configure` agora oferece suporte a uma opção de linha de comando `-X presite=pacote.módulo`. Se usado, especifica um módulo que deve ser importado no início do ciclo de vida do interpretador, antes que `site.py` seja executado. (Contribuição de Łukasz Langa em [gh-110769](#).)

Índice

C

Common Vulnerabilities and Exposures

- CVE 2023-27043, 13
- CVE 2023-52425, 19
- CVE 2024-4030, 15, 17

P

Propostas de Melhorias do Python

- PEP 11, 8, 38
- PEP 11#tier-2, 4
- PEP 11#tier-3, 4
- PEP 587, 35
- PEP 590, 35
- PEP 594, 3, 20
- PEP 602, 5
- PEP 626, 26
- PEP 667, 3, 8, 36, 39
- PEP 669, 4, 30
- PEP 696, 4
- PEP 699, 36
- PEP 702, 4, 19
- PEP 703, 3, 7
- PEP 705, 4, 18
- PEP 709, 39
- PEP 719, 3
- PEP 730, 4, 8
- PEP 737, 33
- PEP 738, 4, 8
- PEP 742, 4, 18
- PEP 744, 3, 4, 8
- PYTHON_BASIC_REPL, 5
- PYTHON_COLORS, 4, 5, 13
- PYTHON_CPU_COUNT, 15
- PYTHON_FROZEN_MODULES, 9
- PYTHON_GIL, 6, 7
- PYTHON_HISTORY, 9
- PYTHON_PERF_JIT_SUPPORT, 9
- PYTHONHOME, 36, 37
- PYTHONLEGACYWINDOWSFSENCODING, 24, 27

R

RFC

- RFC 5280, 17

V

variável de ambiente

- PYTHON_BASIC_REPL, 5
- PYTHON_COLORS, 4, 5, 13
- PYTHON_CPU_COUNT, 15
- PYTHON_FROZEN_MODULES, 9
- PYTHON_GIL, 6, 7
- PYTHON_HISTORY, 9
- PYTHON_PERF_JIT_SUPPORT, 9
- PYTHONHOME, 36, 37
- PYTHONLEGACYWINDOWSFSENCODING, 24, 27