
What's New in Python

Release 3.13.0rc2

A. M. Kuchling

setembro 25, 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
2.7	Coleta de lixo incremental	9
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>	11
5.3	<code>ast</code>	11
5.4	<code>asyncio</code>	11
5.5	<code>base64</code>	12
5.6	<code>compileall</code>	12
5.7	<code>concurrent.futures</code>	12
5.8	<code>configparser</code>	12
5.9	<code>copy</code>	12
5.10	<code>dbm</code>	13
5.11	<code>dis</code>	13
5.12	<code>doctest</code>	13
5.13	<code>email</code>	13
5.14	<code>fractions</code>	13
5.15	<code>gc</code>	14
5.16	<code>glob</code>	14
5.17	<code>importlib</code>	14
5.18	<code>io</code>	14
5.19	<code>ipaddress</code>	15
5.20	<code>itertools</code>	15
5.21	<code>marshal</code>	15
5.22	<code>math</code>	15

5.23	mimetypes	15
5.24	mmap	15
5.25	multiprocessing	16
5.26	os	16
5.27	os.path	16
5.28	pathlib	17
5.29	pdb	17
5.30	queue	17
5.31	random	17
5.32	re	18
5.33	shutil	18
5.34	site	18
5.35	sqlite3	18
5.36	ssl	18
5.37	statistics	18
5.38	subprocess	19
5.39	sys	19
5.40	tempfile	19
5.41	time	19
5.42	tkinter	19
5.43	traceback	20
5.44	types	20
5.45	typing	20
5.46	unicodedata	21
5.47	venv	21
5.48	warnings	21
5.49	xml	21
5.50	zipimport	21
6	Otimizações	21
7	Módulos e APIs removidas	22
7.1	PEP 594: Remover “baterias mortas” da biblioteca padrão	22
7.2	2to3	23
7.3	embutidos	23
7.4	configparser	23
7.5	importlib.metadata	23
7.6	locale	23
7.7	opcode	24
7.8	pathlib	24
7.9	re	24
7.10	tkinter.tix	24
7.11	turtle	24
7.12	typing	24
7.13	unittest	24
7.14	urllib	25
7.15	webbrowser	25
8	Novas descontinuações	25
8.1	Remoção pendente no Python 3.14	27
8.2	Remoção pendente no Python 3.15	29
8.3	Remoção pendente no Python 3.16	29
8.4	Remoção pendente em versões futuras	30
9	Alterações de bytecode do CPython	32
10	Alterações na API C	32
10.1	Novas funcionalidades	32
10.2	Changed C APIs	35

10.3 Limited C API Changes	36
10.4 APIs C removidas	36
10.5 APIs C descontinuadas	38
11 Alterações na construção	40
12 Portando para o Python 3.13	41
12.1 Alterações na API Python	41
12.2 Alterações na API C	42
13 Mudanças em teste de regressão	43
Índice	44

Editors

Adam Turner and Thomas Wouters

Este artigo explica os novos recursos no Python 3.13, em comparação com 3.12. Python 3.13 será lançado em 1 de outubro de 2024. Veja changelog para uma lista completa de mudanças.

Ver também

PEP 719 – Agendamento de lançamento do Python 3.13

1 Resumo – Destaques da versão

O Python 3.13 será 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 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 de 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 de 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__()`.
- The new `dbm.sqlite3` module is now the default `dbm` backend.
- 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.
- There is a new suite of functions for generating **PEP 669** monitoring events in the C API.

Novos recursos de tipagem:

- **PEP 696:** Parâmetros de tipo (`typing.TypeVar`, `typing.ParamSpec` e `typing.TypeVarTuple`) agora oferecem suporte a padrões.
- **PEP 702:** The new `warnings.deprecated()` decorator adds support for marking deprecations in the type system and at runtime.
- **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 a plataforma:

- **PEP 730:** Apple's iOS is now an *officially supported platform*, at **tier 3**.
- **PEP 738:** Android is now an *officially supported platform*, at **tier 3**.
- `wasm32-wasi` agora é suportado como uma plataforma **tier 2**.

- `wasm32-emscrip` não é mais uma plataforma oficialmente suportada.

Remoções importantes:

- [PEP 594](#): The remaining 19 “dead batteries” (legacy stdlib modules) have been removed from the standard library: `aifc`, `audioop`, `cgi`, `cgitb`, `chunk`, `crypt`, `imghdr`, `mailcap`, `msilib`, `nis`, `nnplib`, `ossaudiodev`, `pipes`, `sndhdr`, `spwd`, `sunau`, `telnetlib`, `uu` and `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 the `locale.resetlocale()` function.
- Remove the `typing.io` and `typing.re` namespaces.
- 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

O 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, os seguintes novos recursos agora são suportados:

- 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 maiores de código (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
  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 the import system gives it precedence)
```

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 third-party module 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`.

Free-threaded execution allows for full utilization of the available processing power by running threads in parallel on available CPU cores. While not all software will benefit from this automatically, programs designed with threading in mind will run faster on multi-core hardware. **The free-threaded mode is experimental** and work is ongoing to improve it: expect some bugs and a substantial single-threaded performance hit. Free-threaded builds of CPython support optionally running with the GIL enabled at runtime using the environment variable `PYTHON_GIL` or the command-line option `-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.

This work was made possible thanks to many individuals and organizations, including the large community of contributors to Python and third-party projects to test and enable free-threading support. Notable contributors include: Sam Gross, Ken Jin, Donghee Na, Itamar Oren, Matt Page, Brett Simmers, Dino Viehland, Carl Meyer, Nathan Goldbaum, Ralf Gommers, Lysandros Nikolaou, and many others. Many of these contributors are employed by Meta, which has provided significant engineering resources to support this project.

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-interpret` 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()`

Historically, the expected result of mutating the return value of `locals()` has been left to individual Python implementations to define. Starting from Python 3.13, **PEP 667** standardises the historical behavior of CPython for most code execution scopes, but changes optimized scopes (functions, generators, coroutines, comprehensions, and generator expressions) to explicitly return independent snapshots of the currently assigned local variables, including locally referenced nonlocal variables captured in closures.

This change to the semantics of `locals()` in optimized scopes also affects the default behavior of code execution functions that implicitly target `locals()` if no explicit namespace is provided (such as `exec()` and `eval()`). In previous versions, whether or not changes could be accessed by calling `locals()` after calling the code execution function was implementation-dependent. In CPython specifically, such code would typically appear to work as desired, but could sometimes fail in optimized scopes based on other code (including debuggers and code execution tracing tools) potentially resetting the shared snapshot in that scope. Now, the code will always run against an independent snapshot of the local variables in optimized scopes, and hence the changes will never be visible in subsequent calls to `locals()`. To access the changes made in these cases, an explicit namespace reference must now be passed to the relevant function. Alternatively, it may make sense to update affected code to use a higher level code execution API that returns the resulting code execution namespace (e.g. `runpy.run_path()` when executing Python files from disk).

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: O iOS agora é uma plataforma suportada 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 tier 3, mas terá suporte de melhor esforço. (PEP escrita e implementação como contribuição de Russell Keith-Magee em [gh-114099](#).)

PEP 738: Android is now a **PEP 11** supported platform, with the `aarch64-linux-android` and `x86_64-linux-android` targets at tier 3. The 32-bit targets `arm-linux-androideabi` and `i686-linux-android` are not tier 3 supported platforms, but will have best-effort support. (PEP written and implementation contributed by Malcolm Smith in [gh-116622](#).)

Ver também

PEP 730, PEP 738

2.7 Coleta de lixo incremental

O coletor de lixo do ciclo agora é incremental. Isso significa que os tempos máximos de pausa são reduzidos em uma ordem de magnitude ou mais para heaps maiores.

Agora há apenas duas gerações: jovem e velha. Quando `gc.collect()` não é chamado diretamente, o GC é invocado com um pouco menos de frequência. Quando invocado, ele coleta a geração jovem e um incremento da geração velha, em vez de coletar uma ou mais gerações.

O comportamento de `gc.collect()` mudou ligeiramente:

- `gc.collect(1)`: Performs an increment of garbage collection, rather than collecting generation 1.
- Outras chamadas para `gc.collect()` não foram alteradas.

(Contribuição de Mark Shannon em [gh-108362](#).)

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 código do tipo `'u'` descontinuado. (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

- The constructors of node types in the `ast` module are now stricter in the arguments they accept, with more intuitive behavior when arguments are omitted.

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 iterador 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 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](#).)

5.11 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](#).)

5.12 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](#).)

5.13 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()` and `parseaddr()` now return `(' ', '')` pairs in more situations where invalid email addresses are encountered instead of potentially inaccurate values. The two functions have a new optional `strict` parameter (default `True`). To get the old behavior (accepting malformed input), use `strict=False`. `getattr(email.utils, 'supports_strict_parsing', False)` can be used to check if the `strict` parameter is available. (Contributed by Thomas Dwyer and Victor Stinner for [gh-102988](#) to improve the CVE-2023-27043 fix.)

5.14 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](#).)

5.15 gc

O coletor de lixo cíclico agora é incremental, o que altera o significado dos resultados de `get_threshold()` e `set_threshold()`, bem como de `get_count()` e `get_stats()`.

- Para compatibilidade reversa, `get_threshold()` continua a retornar uma tupla de três itens. O primeiro valor é o limite para coleções jovens, como antes; o segundo valor determina a taxa na qual a coleção antiga é verificada (o padrão é 10 e valores mais altos significam que a coleção antiga é verificada mais lentamente). O terceiro valor não tem sentido e é sempre zero.
- `set_threshold()` ignora quaisquer itens após o segundo.
- `get_count()` e `get_stats()` continuam a retornar o mesmo formato de resultados. A única diferença é que em vez dos resultados fazerem referência às gerações jovens, em envelhecimento e antigas, os resultados fazem referência à geração jovem e em envelhecimento, e coleta de espaços de geração antigas.

Em resumo, o código que tentou manipular o comportamento do ciclo GC pode não funcionar exatamente como pretendido, mas é muito improvável que seja prejudicial. Todos os outros códigos funcionarão perfeitamente.

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-106532](#).)

- 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-106532](#).)

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 suportada. (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.VIFY_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 descontinuações 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

- O novo *coletor de lixo incremental* significa que os tempos máximos de pausa são reduzidos em uma ordem de magnitude ou mais para heaps maiores. (Contribuição de Mark Shannon em [gh-108362](#).)
- 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: Remover “baterias mortas” da biblioteca padrão

PEP 594 propôs remover 19 módulos da biblioteca padrão, coloquialmente chamados de “baterias mortas” 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`: O algoritmo seguro de hashing de senha Argon2.
 - `legacycrypt`: 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 eram suportados por `imghdr`.
- `mailcap`: use o módulo `mimetypes`.
- `msilib`

- `nis`
- `nntplib`: Use a biblioteca `nntplib` 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](#).)
- Raise a `RuntimeError` when calling `frame.clear()` on a suspended frame (as has always been the case for an executing frame). (Contributed by Irit Katriel in [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, "")` em vez disso. (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. Use o parâmetro `context` em vez disso 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` em vez disso. (Contribuição de Nikita Sobolev em [gh-105546](#).)

8 Novas descontinuações

- User-defined functions:
 - Deprecate assignment to a function's `__code__` attribute, where the new code object's type does not match the function's type. The different types are: plain function, generator, async generator, and coroutine. (Contributed by Irit Katriel in [gh-81137](#).)
- array:
 - Deprecate the `'u'` format code (`wchar_t`) at runtime. This format code has been deprecated in documentation since Python 3.3, and will be removed in Python 3.16. Use the `'w'` format code (`Py_UCS4`) for Unicode characters instead. (Contributed by Hugo van Kemenade in [gh-80480](#).)
- ctypes:
 - Deprecate the undocumented `SetPointerType()` function, to be removed in Python 3.15. (Contributed by Victor Stinner in [gh-105733](#).)
 - Soft-deprecate the `ARRAY()` function in favour of `type * length` multiplication. (Contributed by Victor Stinner in [gh-105733](#).)
- decimal:
 - Deprecate the non-standard and undocumented `Decimal` format specifier `'N'`, which is only supported in the `decimal` module's C implementation. (Contributed by Serhiy Storchaka in [gh-89902](#).)
- dis:
 - Deprecate the `HAVE_ARGUMENT` separator. Check membership in `hasarg` instead. (Contributed by Irit Katriel in [gh-109319](#).)
- getopt and optparse:
 - Both modules are now soft deprecated, with `argparse` preferred for new projects. This is a new soft-deprecation for the `getopt` module, whereas the `optparse` module was already *de facto* soft deprecated. (Contributed by Victor Stinner in [gh-106535](#).)
- gettext:

- Deprecate non-integer numbers as arguments to functions and methods that consider plural forms in the `gettext` module, even if no translation was found. (Contributed by Serhiy Storchaka in [gh-88434](#).)
- `glob`:
 - Deprecate the undocumented `glob0()` and `glob1()` functions. Use `glob()` and pass a path-like object specifying the root directory to the `root_dir` parameter instead. (Contributed by Barney Gale in [gh-117337](#).)
- `http.server`:
 - Deprecate `CGIHTTPRequestHandler`, to be removed in Python 3.15. Process-based CGI HTTP servers have been out of favor for a very long time. This code was outdated, unmaintained, and rarely used. It has a high potential for both security and functionality bugs. (Contributed by Gregory P. Smith in [gh-109096](#).)
 - Deprecate the `--cgi` flag to the **`python -m http.server`** command-line interface, to be removed in Python 3.15. (Contributed by Gregory P. Smith in [gh-109096](#).)
- `mimetypes`:
 - Soft-deprecate file path arguments to `guess_type()`, use `guess_file_type()` instead. (Contributed by Serhiy Storchaka in [gh-66543](#).)
- `re`:
 - Deprecate passing the optional *maxsplit*, *count*, or *flags* arguments as positional arguments to the module-level `split()`, `sub()`, and `subn()` functions. These parameters will become keyword-only in a future version of Python. (Contributed by Serhiy Storchaka in [gh-56166](#).)
- `pathlib`:
 - Deprecate `PurePath.is_reserved()`, to be removed in Python 3.15. Use `os.path.isreserved()` to detect reserved paths on Windows. (Contributed by Barney Gale in [gh-88569](#).)
- `platform`:
 - Deprecate `java_ver()`, to be removed in Python 3.15. This function is only useful for Jython support, has a confusing API, and is largely untested. (Contributed by Nikita Sobolev in [gh-116349](#).)
- `pydoc`:
 - Deprecate the undocumented `ispackage()` function. (Contributed by Zackery Spytz in [gh-64020](#).)
- `sqlite3`:
 - Deprecate passing more than one positional argument to the `connect()` function and the `Connection` constructor. The remaining parameters will become keyword-only in Python 3.15. (Contributed by Erlend E. Aasland in [gh-107948](#).)
 - Deprecate passing `name`, number of arguments, and the callable as keyword arguments for `Connection.create_function()` and `Connection.create_aggregate()`. These parameters will become positional-only in Python 3.15. (Contributed by Erlend E. Aasland in [gh-108278](#).)
 - Deprecate passing the callback callable by keyword for the `set_authorizer()`, `set_progress_handler()`, and `set_trace_callback()` `Connection` methods. The callback callables will become positional-only in Python 3.15. (Contributed by Erlend E. Aasland in [gh-108278](#).)
- `sys`:
 - Deprecate the `_enablelegacywindowsfsencoding()` function, to be removed in Python 3.16. Use the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead. (Contributed by Inada Naoki in [gh-73427](#).)
- `tarfile`:
 - Deprecate the undocumented and unused `TarFile.tarfile` attribute, to be removed in Python 3.16. (Contributed in [gh-115256](#).)

- `traceback`:
 - Deprecate the `TracebackException.exc_type` attribute. Use `TracebackException.exc_type_str` instead. (Contributed by Irit Katriel in [gh-112332](#).)
- `typing`:
 - Deprecate the undocumented keyword argument syntax for creating `NamedTuple` classes (e.g. `Point = NamedTuple("Point", x=int, y=int)`), to be removed in Python 3.15. Use the class-based syntax or the functional syntax instead. (Contributed by Alex Waygood in [gh-105566](#).)
 - Deprecate omitting the *fields* parameter when creating a `NamedTuple` or `typing.TypedDict` class, and deprecate passing `None` to the *fields* parameter of both types. Python 3.15 will require a valid sequence for the *fields* parameter. To create a `NamedTuple` class with zero fields, use `class NT(NamedTuple): pass` or `NT = NamedTuple("NT", ())`. To create a `TypedDict` class with zero fields, use `class TD(TypedDict): pass` or `TD = TypedDict("TD", {})`. (Contributed by Alex Waygood in [gh-105566](#) and [gh-105570](#).)
 - Deprecate the `typing.no_type_check_decorator()` decorator function, to be removed in Python 3.15. After eight years in the `typing` module, it has yet to be supported by any major type checker. (Contributed by Alex Waygood in [gh-106309](#).)
 - Deprecate `typing.AnyStr`. In Python 3.16, it will be removed from `typing.__all__`, and a `DeprecationWarning` will be emitted at runtime when it is imported or accessed. It will be removed entirely in Python 3.18. Use the new type parameter syntax instead. (Contributed by Michael The in [gh-107116](#).)
- `wave`:
 - Deprecate the `getmark()`, `setmark()`, and `getmarkers()` methods of the `Wave_read` and `Wave_write` classes, to be removed in Python 3.15. (Contributed by Victor Stinner in [gh-105096](#).)

8.1 Remoção pendente no Python 3.14

- `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`

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

- `asyncio`:
 - As classes filhas do observador `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`: `__package__` e `__cached__` deixarão de ser definidos ou levados em consideração pelo sistema de importação ([gh-97879](#)).
- `importlib.abc` descontinuou as classes:
 - `importlib.abc.ResourceReader`
 - `importlib.abc.Traversable`
 - `importlib.abc.TraversableResources`

Em vez disso, use classes `importlib.resources.abc`:

- `importlib.resources.abc.Traversable`
- `importlib.resources.abc.TraversableResources`

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

- `itertools` 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()` em vez disso. (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

- `ctypes`:
 - The undocumented `ctypes.SetPointerType()` function has been deprecated since Python 3.13.
- `http.server`:
 - The obsolete and rarely used `CGIHTTPRequestHandler` has been deprecated since Python 3.13. No direct replacement exists. *Anything* is better than CGI to interface a web server with a request handler.
 - The `--cgi` flag to the **python -m http.server** command-line interface has been deprecated since Python 3.13.
- `locale`:
 - The `getdefaultlocale()` function has been deprecated since Python 3.11. Its removal was originally planned for Python 3.13 ([gh-90817](#)), but has been postponed to Python 3.15. Use `getlocale()`, `setlocale()`, and `getencoding()` instead. (Contributed by Hugo van Kemenade in [gh-111187](#).)
- `pathlib`:
 - `PurePath.is_reserved()` has been deprecated since Python 3.13. Use `os.path.isreserved()` to detect reserved paths on Windows.
- `platform`:
 - `java_ver()` has been deprecated since Python 3.13. This function is only useful for Jython support, has a confusing API, and is largely untested.
- `threading`:
 - `RLock()` will take no arguments in Python 3.15. Passing any arguments has been deprecated since Python 3.14, as the Python version does not permit any arguments, but the C version allows any number of positional or keyword arguments, ignoring every argument.
- `typing`:
 - The undocumented keyword argument syntax for creating `NamedTuple` classes (e.g. `Point = NamedTuple("Point", x=int, y=int)`) has been deprecated since Python 3.13. Use the class-based syntax or the functional syntax instead.
 - The `typing.no_type_check_decorator()` decorator function has been deprecated since Python 3.13. After eight years in the `typing` module, it has yet to be supported by any major type checker.
- `wave`:
 - The `getmark()`, `setmark()`, and `getmarkers()` methods of the `Wave_read` and `Wave_write` classes have been deprecated since Python 3.13.

8.3 Remoção pendente no Python 3.16

- `builtins`:
 - Bitwise inversion on boolean types, `~True` or `~False` has been deprecated since Python 3.12, as it produces surprising and unintuitive results (`-2` and `-1`). Use `not x` instead for the logical negation of a Boolean. In the rare case that you need the bitwise inversion of the underlying integer, convert to `int` explicitly (`~int(x)`).
- `array`:
 - The `'u'` format code (`wchar_t`) has been deprecated in documentation since Python 3.3 and at runtime since Python 3.13. Use the `'w'` format code (`Py_UCS4`) for Unicode characters instead.
- `shutil`:

- The `ExecError` exception has been deprecated since Python 3.14. It has not been used by any function in `shutil` since Python 3.4, and is now an alias of `RuntimeError`.
- `symtable`:
 - The `Class.get_methods` method has been deprecated since Python 3.14.
- `sys`:
 - The `_enablelegacywindowsfsencoding()` function has been deprecated since Python 3.13. Use the `PYTHONLEGACYWINDOWSFSENCODING` environment variable instead.
- `tarfile`:
 - The undocumented and unused `TarFile.tarfile` attribute has been deprecated since 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 do argumento único.
 - 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()` em vez disso.
 - O parâmetro `debug_override` de `cache_from_source()` foi descontinuado: em vez disso, use o parâmetro `optimization`.
- `importlib.metadata`:
 - Interface de tupla `EntryPoint`s.
 - `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; em vez disso, 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

- The oparg of `YIELD_VALUE` is now 1 if the yield is part of a yield-from or await, and 0 otherwise. The oparg of `RESUME` was changed to add a bit indicating if the except-depth is 1, which is needed to optimize closing of generators. (Contributed by Irit Katriel in [gh-111354](#).)

10 Alterações na API C

10.1 Novas funcionalidades

- Add the PyMonitoring C API for generating **PEP 669** monitoring events:
 - `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()`

(Contributed by Irit Katriel in [gh-111997](#)).

- Add `PyMutex`, a lightweight mutex that occupies a single byte, and the new `PyMutex_Lock()` and `PyMutex_Unlock()` functions. `PyMutex_Lock()` will release the GIL (if currently held) if the operation needs to block. (Contributed by Sam Gross in [gh-108724](#).)
- Add the `PyTime` C API to provide access to system clocks:
 - `PyTime_t`.
 - `PyTime_MIN` and `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](#).)

- Add the `PyDict_ContainsString()` function with the same behavior as `PyDict_Contains()`, but `key` is specified as a `const char*` UTF-8 encoded bytes string, rather than a `PyObject*`. (Contributed by Victor Stinner in [gh-108314](#).)
- Add the `PyDict_GetItemRef()` and `PyDict_GetItemStringRef()` functions, which behave similarly to `PyDict_GetItemWithError()`, but return a strong reference instead of a borrowed reference. Moreover, these functions return `-1` on error, removing the need to check `PyErr_Occurred()`. (Contributed by Victor Stinner in [gh-106004](#).)
- Add the `PyDict_SetDefaultRef()` function, which behaves similarly to `PyDict_SetDefault()`, but returns a strong reference instead of a borrowed reference. This function returns `-1` on error, `0` on insertion, and `1` if the key was already present in the dictionary. (Contributed by Sam Gross in [gh-112066](#).)
- Add the `PyDict_Pop()` and `PyDict_PopString()` functions to remove a key from a dictionary and optionally return the removed value. This is similar to `dict.pop()`, though there is no default value, and `KeyError` is not raised for missing keys. (Contributed by Stefan Behnel and Victor Stinner in [gh-111262](#).)
- Add the `PyMapping_GetOptionalItem()` and `PyMapping_GetOptionalItemString()` functions as alternatives to `PyObject_GetItem()` and `PyMapping_GetItemString()` respectively. The new functions do not raise `KeyError` if the requested key is missing from the mapping. These

variants are more convenient and faster if a missing key should not be treated as a failure. (Contributed by Serhiy Storchaka in [gh-106307](#).)

- Add the `PyObject_GetOptionalAttr()` and `PyObject_GetOptionalAttrString()` functions as alternatives to `PyObject_GetAttr()` and `PyObject_GetAttrString()` respectively. The new functions do not raise `AttributeError` if the requested attribute is not found on the object. These variants are more convenient and faster if the missing attribute should not be treated as a failure. (Contributed by Serhiy Storchaka in [gh-106521](#).)
- Add the `PyErr_FormatUnraisable()` function as an extension to `PyErr_WriteUnraisable()` that allows customizing the warning message. (Contributed by Serhiy Storchaka in [gh-108082](#).)
- Add new functions that return a strong reference instead of a borrowed reference for frame locals, globals, and builtins, as part of [PEP 667](#):

- `PyEval_GetFrameBuiltins()` replaces `PyEval_GetBuiltins()`
- `PyEval_GetFrameGlobals()` replaces `PyEval_GetGlobals()`
- `PyEval_GetFrameLocals()` replaces `PyEval_GetLocals()`

(Contributed by Mark Shannon and Tian Gao in [gh-74929](#).)

- Add the `Py_GetConstant()` and `Py_GetConstantBorrowed()` functions to get strong or borrowed references to constants. For example, `Py_GetConstant(Py_CONSTANT_ZERO)` returns a strong reference to the constant zero. (Contributed by Victor Stinner in [gh-115754](#).)
- Add the `PyImport_AddModuleRef()` function as a replacement for `PyImport_AddModule()` that returns a strong reference instead of a borrowed reference. (Contributed by Victor Stinner in [gh-105922](#).)
- Add the `Py_IsFinalizing()` function to check whether the main Python interpreter is shutting down. (Contributed by Victor Stinner in [gh-108014](#).)
- Add the `PyList_GetItemRef()` function as a replacement for `PyList_GetItem()` that returns a strong reference instead of a borrowed reference. (Contributed by Sam Gross in [gh-114329](#).)
- Add the `PyList_Extend()` and `PyList_Clear()` functions, mirroring the Python `list.extend()` and `list.clear()` methods. (Contributed by Victor Stinner in [gh-111138](#).)
- Add the `PyLong_AsInt()` function. It behaves similarly to `PyLong_AsLong()`, but stores the result in a `C int` instead of a `C long`. (Contributed by Victor Stinner in [gh-108014](#).)
- Add the `PyLong_AsNativeBytes()`, `PyLong_FromNativeBytes()`, and `PyLong_FromUnsignedNativeBytes()` functions to simplify converting between native integer types and Python `int` objects. (Contributed by Steve Dower in [gh-111140](#).)
- Add `PyModule_Add()` function, which is similar to `PyModule_AddObjectRef()` and `PyModule_AddObject()`, but always steals a reference to the value. (Contributed by Serhiy Storchaka in [gh-86493](#).)
- Add the `PyObject_GenericHash()` function that implements the default hashing function of a Python object. (Contributed by Serhiy Storchaka in [gh-113024](#).)
- Add the `Py_HashPointer()` function to hash a raw pointer. (Contributed by Victor Stinner in [gh-111545](#).)
- Add the `PyObject_VisitManagedDict()` and `PyObject_ClearManagedDict()` functions, which must be called by the traverse and clear functions of a type using the `Py_TPFLAGS_MANAGED_DICT` flag. The [pythoncapi-compat](#) project can be used to use these functions with Python 3.11 and 3.12. (Contributed by Victor Stinner in [gh-107073](#).)
- Add the `PyRefTracer_SetTracer()` and `PyRefTracer_GetTracer()` functions, which enable tracking object creation and destruction in the same way that the `tracemalloc` module does. (Contributed by Pablo Galindo in [gh-93502](#).)
- Add the `PySys_AuditTuple()` function as an alternative to `PySys_Audit()` that takes event arguments as a Python tuple object. (Contributed by Victor Stinner in [gh-85283](#).)

- Add the `PyThreadState_GetUnchecked()` function as an alternative to `PyThreadState_Get()` that doesn't kill the process with a fatal error if it is `NULL`. The caller is responsible for checking if the result is `NULL`. (Contributed by Victor Stinner in [gh-108867](#).)
- Add the `PyType_GetFullyQualifiedName()` function to get the type's fully qualified name. The module name is prepended if `type.__module__` is a string and is not equal to either `'builtins'` or `'__main__'`. (Contributed by Victor Stinner in [gh-111696](#).)
- Add the `PyType_GetModuleName()` function to get the type's module name. This is equivalent to getting the `type.__module__` attribute. (Contributed by Eric Snow and Victor Stinner in [gh-111696](#).)
- Add the `PyUnicode_EqualToUTF8AndSize()` and `PyUnicode_EqualToUTF8()` functions to compare a Unicode object with a `const char*` UTF-8 encoded string and 1 if they are equal or 0 otherwise. These functions do not raise exceptions. (Contributed by Serhiy Storchaka in [gh-110289](#).)
- Add the `PyWeakref_GetRef()` function as an alternative to `PyWeakref_GetObject()` that returns a strong reference or `NULL` if the referent is no longer live. (Contributed by Victor Stinner in [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()`.

The new functions return `-1` for errors and the standard `1` for true and `0` for false.

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

10.2 Changed C APIs

- The `keywords` parameter of `PyArg_ParseTupleAndKeywords()` and `PyArg_VaParseTupleAndKeywords()` now has type `char *const*` in C and `const char *const*` in C++, instead of `char**`. In C++, this makes these functions compatible with arguments of type `const char *const*`, `const char**`, or `char *const*` without an explicit type cast. In C, the functions only support arguments of type `char *const*`. This can be overridden with the `PY_CXX_CONST` macro. (Contributed by Serhiy Storchaka in [gh-65210](#).)
- `PyArg_ParseTupleAndKeywords()` agora oferece suporte a nomes de parâmetros nomeados não-ASCII. (Contribuição de Serhiy Storchaka em [gh-110815](#).)
- The `PyCode_GetFirstFree()` function is now unstable API and is now named `PyUnstable_Code_GetFirstFree()`. (Contributed by Bogdan Romanyuk in [gh-115781](#).)
- The `PyDict_GetItem()`, `PyDict_GetItemString()`, `PyMapping_HasKey()`, `PyMapping_HasKeyString()`, `PyObject_HasAttr()`, `PyObject_HasAttrString()`, and `PySys_GetObject()` functions, each of which clears all errors which occurred when calling them now reports these errors using `sys.unraisablehook()`. You may replace them with other functions as recommended in the documentation. (Contributed by Serhiy Storchaka in [gh-106672](#).)
- Add support for the `%T`, `%#T`, `%N` and `%#N` formats to `PyUnicode_FromFormat()`:
 - `%T`: Get the fully qualified name of an object type
 - `%#T`: As above, but use a colon as the separator
 - `%N`: Get the fully qualified name of a type
 - `%#N`: As above, but use a colon as the separator

See [PEP 737](#) for more information. (Contributed by Victor Stinner in [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 Limited C API Changes

- The following functions are now included in the Limited C API:

- `PyMem_RawMalloc()`
- `PyMem_RawCalloc()`
- `PyMem_RawRealloc()`
- `PyMem_RawFree()`
- `PySys_Audit()`
- `PySys_AuditTuple()`
- `PyType_GetModuleByDef()`

(Contributed by Victor Stinner in [gh-85283](#), [gh-85283](#), and [gh-116936](#).)

- Python built with `--with-trace-refs` (tracing references) now supports the Limited API. (Contributed by Victor Stinner in [gh-108634](#).)

10.4 APIs C removidas

- Remove several functions, macros, variables, etc with names prefixed by `_Py` or `_PY` (which are considered private). If your project is affected by one of these removals and you believe that the removed API should remain available, please open a new issue to request a public C API and add `cc: @vstinner` to the issue to notify Victor Stinner. (Contributed by Victor Stinner in [gh-106320](#).)
- Remove protocolos de buffer antigos descontinuados no Python 3.0. Use `bufferobjects` em vez disso.

- `PyObject_CheckReadBuffer()`: Use `PyObject_CheckBuffer()` to test whether the object supports the buffer protocol. Note that `PyObject_CheckBuffer()` doesn't guarantee that `PyObject_GetBuffer()` will succeed. To test if the object is actually readable, see the next example of `PyObject_GetBuffer()`.
- `PyObject_AsCharBuffer()`, `PyObject_AsReadBuffer()`: Use `PyObject_GetBuffer()` and `PyBuffer_Release()` instead:

```
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 various functions deprecated in Python 3.9:

- `PyEval_CallObject()`, `PyEval_CallObjectWithKeywords()`: Use `PyObject_CallNoArgs()` or `PyObject_Call()` instead.

Aviso

In `PyObject_Call()`, positional arguments must be a tuple and must not be NULL, and keyword arguments must be a dict or NULL, whereas the removed functions checked argument types and accepted NULL positional and keyword arguments. To replace `PyEval_CallObjectWithKeywords(func, NULL, kwargs)` with `PyObject_Call()`, pass an empty tuple as positional arguments using `PyTuple_New(0)`.

- `PyEval_CallFunction()`: Use `PyObject_CallFunction()` instead.
- `PyEval_CallMethod()`: Use `PyObject_CallMethod()` instead.
- `PyCFunction_Call()`: Use `PyObject_Call()` instead.

(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` instead.
- `PySys_AddWarnOption()`: Use `PyConfig.warnoptions` instead.
- `PySys_AddXOption()`: Use `PyConfig.xoptions` instead.
- `PySys_HasWarnOptions()`: Use `PyConfig.xoptions` instead.
- `PySys_SetPath()`: Set `PyConfig.module_search_paths` instead.
- `Py_SetPath()`: Set `PyConfig.module_search_paths` instead.
- `Py_SetStandardStreamEncoding()`: Set `PyConfig.stdio_encoding` instead, and set also maybe `PyConfig.legacy_windows_stdio` (on Windows).
- `_Py_SetProgramFullPath()`: Set `PyConfig.executable` instead.

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 `PyEval_AcquireLock()` and `PyEval_ReleaseLock()` functions, deprecated in Python 3.2. They didn't update the current thread state. They can be replaced with:

- `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 the `PyEval_ThreadsInitialized()` function, deprecated in Python 3.9. Since Python 3.7, `Py_Initialize()` always creates the GIL: calling `PyEval_InitThreads()` does nothing and `PyEval_ThreadsInitialized()` always returns non-zero. (Contributed by Victor Stinner in [gh-105182](#).)
- Remove the `_PyInterpreterState_Get()` alias to `PyInterpreterState_Get()` which was kept for backward compatibility with Python 3.8. The [pythoncapi-compat](#) project can be used to get `PyInterpreterState_Get()` on Python 3.8 and older. (Contributed by Victor Stinner in [gh-106320](#).)
- Remove the private `_PyObject_FastCall()` function: use `PyObject_Vectorcall()` which is available since Python 3.8 ([PEP 590](#)). (Contributed by Victor Stinner in [gh-106023](#).)
- Remove the `cpython/pytime.h` header file, which only contained private functions. (Contributed by Victor Stinner in [gh-106316](#).)

- Remove the undocumented `PY_TIMEOUT_MAX` constant from the limited C API. (Contributed by Victor Stinner in [gh-110014](#).)
- Remove the old trashcan macros `Py_TRASHCAN_SAFE_BEGIN` and `Py_TRASHCAN_SAFE_END`. Replace both with the new macros `Py_TRASHCAN_BEGIN` and `Py_TRASHCAN_END`. (Contributed by Irit Katriel in [gh-105111](#).)

10.5 APIs C descontinuadas

- Descontinua antigas funções de inicialização do Python:
 - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and `warnings.filters` instead.
 - `Py_GetExecPrefix()`: Get `sys.exec_prefix` instead.
 - `Py_GetPath()`: Get `sys.path` instead.
 - `Py_GetPrefix()`: Get `sys.prefix` instead.
 - `Py_GetProgramFullPath()`: Get `sys.executable` instead.
 - `Py_GetProgramName()`: Get `sys.executable` instead.
 - `Py_GetPythonHome()`: Get `PyConfig.home` or the `PYTHONHOME` environment variable instead.
 (Contributed by Victor Stinner in [gh-105145](#).)
- Soft deprecate the `PyEval_GetBuiltins()`, `PyEval_GetGlobals()`, and `PyEval_GetLocals()` functions, which return a borrowed reference. (Soft deprecated as part of [PEP 667](#).)
- Deprecate the `PyImport_ImportModuleNoBlock()` function, which is just an alias to `PyImport_ImportModule()` since Python 3.3. (Contributed by Victor Stinner in [gh-105396](#).)
- Soft deprecate the `PyModule_AddObject()` function. It should be replaced with `PyModule_Add()` or `PyModule_AddObjectRef()`. (Contributed by Serhiy Storchaka in [gh-86493](#).)
- Deprecate the old `Py_UNICODE` and `PY_UNICODE_TYPE` types and the `Py_UNICODE_WIDE` define. Use the `wchar_t` type directly instead. Since Python 3.3, `Py_UNICODE` and `PY_UNICODE_TYPE` are just aliases to `wchar_t`. (Contributed by Victor Stinner in [gh-105156](#).)
- Deprecate the `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()` functions, which return a borrowed reference. Replace them with the new `PyWeakref_GetRef()` function, which returns a strong reference. The [pythoncapi-compat](#) project can be used to get `PyWeakref_GetRef()` on Python 3.12 and older. (Contributed by Victor Stinner in [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()`: Set `PyConfig.argv` instead.
 - `PySys_SetArgv()`: Set `PyConfig.argv` instead.
 - `Py_SetProgramName()`: Set `PyConfig.program_name` instead.
 - `Py_SetPythonHome()`: Set `PyConfig.home` instead.
 A API `Py_InitializeFromConfig()` deve ser usada com `PyConfig`.
- Variáveis de configuração globais

- `Py_DebugFlag`: Use `PyConfig.parser_debug` instead.
- `Py_VerboseFlag`: Use `PyConfig.verbose` instead.
- `Py_QuietFlag`: Use `PyConfig.quiet` instead.
- `Py_InteractiveFlag`: Use `PyConfig.interactive` instead.
- `Py_InspectFlag`: Use `PyConfig.inspect` instead.
- `Py_OptimizeFlag`: Use `PyConfig.optimization_level` instead.
- `Py_NoSiteFlag`: Use `PyConfig.site_import` instead.
- `Py_BytesWarningFlag`: Use `PyConfig.bytes_warning` instead.
- `Py_FrozenFlag`: Use `PyConfig.pathconfig_warnings` instead.
- `Py_IgnoreEnvironmentFlag`: Use `PyConfig.use_environment` instead.
- `Py_DontWriteBytecodeFlag`: Use `PyConfig.write_bytecode` instead.
- `Py_NoUserSiteDirectory`: Use `PyConfig.user_site_directory` instead.
- `Py_UnbufferedStdioFlag`: Use `PyConfig.buffered_stdio` instead.
- `Py_HashRandomizationFlag`: Use `PyConfig.use_hash_seed` and `PyConfig.hash_seed` instead.
- `Py_IsolatedFlag`: Use `PyConfig.isolated` instead.
- `Py_LegacyWindowsFSEncodingFlag`: Use `PyPreConfig.legacy_windows_fs_encoding` instead.
- `Py_LegacyWindowsStdioFlag`: Use `PyConfig.legacy_windows_stdio` instead.
- `Py_FileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` instead.
- `Py_HasFileSystemDefaultEncoding`: Use `PyConfig.filesystem_encoding` instead.
- `Py_FileSystemDefaultEncodeErrors`: Use `PyConfig.filesystem_errors` instead.
- `Py_UTF8Mode`: Use `PyPreConfig.utf8_mode` instead. (see `Py_PreInitialize()`)

A API `Py_InitializeFromConfig()` deve ser usada com `PyConfig`.

Remoção pendente no Python 3.15

- A cópia empacotada do `libmpdecimal`.
- The `PyImport_ImportModuleNoBlock()`: Use `PyImport_ImportModule()` instead.
- `PyWeakref_GetObject()` and `PyWeakref_GET_OBJECT()`: Use `PyWeakref_GetRef()` instead.
- `Py_UNICODE` type and the `Py_UNICODE_WIDE` macro: Use `wchar_t` instead.
- Funções de inicialização do Python
 - `PySys_ResetWarnOptions()`: Clear `sys.warnoptions` and `warnings.filters` instead.
 - `Py_GetExecPrefix()`: Get `sys.exec_prefix` instead.
 - `Py_GetPath()`: Get `sys.path` instead.
 - `Py_GetPrefix()`: Get `sys.prefix` instead.
 - `Py_GetProgramFullPath()`: Get `sys.executable` instead.
 - `Py_GetProgramName()`: Get `sys.executable` instead.

- `Py_GetPythonHome()`: Get `PyConfig.home` or the `PYTHONHOME` environment variable instead.

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`: Unneeded since Python 3.8.
- `PyErr_Fetch()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_NormalizeException()`: Use `PyErr_GetRaisedException()` instead.
- `PyErr_Restore()`: Use `PyErr_SetRaisedException()` instead.
- `PyModule_GetFilename()`: Use `PyModule_GetFilenameObject()` instead.
- `PyOS_AfterFork()`: Use `PyOS_AfterFork_Child()` instead.
- `PySlice_GetIndicesEx()`: Use `PySlice_Unpack()` and `PySlice_AdjustIndices()` instead.
- `PyUnicode_AsDecodedObject()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsDecodedUnicode()`: Use `PyCodec_Decode()` instead.
- `PyUnicode_AsEncodedObject()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_AsEncodedUnicode()`: Use `PyCodec_Encode()` instead.
- `PyUnicode_READY()`: Unneeded since Python 3.12
- `PyErr_Display()`: Use `PyErr_DisplayException()` instead.
- `_PyErr_ChainExceptions()`: Use `_PyErr_ChainExceptions1()` instead.
- 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()` instead.
 - `PyThread_delete_key()`: Use `PyThread_tss_free()` instead.
 - `PyThread_set_key_value()`: Use `PyThread_tss_set()` instead.
 - `PyThread_get_key_value()`: Use `PyThread_tss_get()` instead.
 - `PyThread_delete_key_value()`: Use `PyThread_tss_delete()` instead.
 - `PyThread_ReInitTLS()`: Unneeded since Python 3.7.

11 Alterações na construção

- `arm64-apple-ios` and `arm64-apple-ios-simulator` are both now **PEP 11** tier 3 platforms. (*PEP 730* written and implementation contributed by Russell Keith-Magee in [gh-114099](#).)
- `aarch64-linux-android` and `x86_64-linux-android` are both now **PEP 11** tier 3 platforms. (*PEP 738* written and implementation contributed by Malcolm Smith in [gh-116622](#).)
- `wasm32-wasi` agora é uma plataforma de nível 2 segundo a **PEP 11**. (Contribuição de Brett Cannon em [gh-115192](#).)
- `wasm32-emscripten` 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 1.16.4 agora são necessários para regerar o script `configure`. (Contribuição de Christian Heimes em [gh-89886](#).)
- SQLite 3.15.2 ou mais recente é necessário para construir o módulo de extensão `sqlite3`. (Contribuição de Erlend Aasland em [gh-105875](#).)
- CPython now bundles the [mimalloc library](#) by default. It is licensed under the MIT license; see [mimalloc license](#). The bundled mimalloc has custom changes, see [gh-113141](#) for details. (Contributed by Dino Viehland in [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](#) introduces several changes to the semantics of `locals()` and `f_locals`:
 - Calling `locals()` in an optimized scope now produces an independent snapshot on each call, and hence no longer implicitly updates previously returned references. Obtaining the legacy CPython behavior now requires explicit calls to update the initially returned dictionary with the results of subsequent calls to `locals()`. Code execution functions that implicitly target `locals()` (such as `exec` and `eval`) must be passed an explicit namespace to access their results in an optimized scope. (Changed as part of [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` now emits a `FutureWarning` when used as a method. The behavior will change in future Python versions. Wrap it in `staticmethod()` if you want to preserve the old behavior. (Contributed by Serhiy Storchaka in [gh-121027](#).)
- The [garbage collector is now incremental](#), which means that the behavior of `gc.collect()` changes slightly:
 - `gc.collect(1)`: Performs an increment of garbage collection, rather than collecting generation 1.

- Outras chamadas para `gc.collect()` não foram alteradas.
- Uma `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.
- The value of the `mode` attribute of `gzip.GzipFile` is now a string ('rb' or 'wb') instead of an integer (1 or 2). The value of the `mode` attribute of the readable file-like object returned by `zipfile.ZipFile.open()` is now 'rb' instead of 'r'. (Contributed by Serhiy Storchaka in [gh-115961](#).)
- `mailbox.Maildir` now ignores files with a leading dot (.). (Contributed by Zackery Spytz in [gh-65559](#).)
- `pathlib.Path.glob()` and `rglob()` now return both files and directories if a pattern that ends with “**” is given, rather than directories only. Add a trailing slash to keep the previous behavior and only match directories.
- The `threading` module now expects the `_thread` module to have an `_is_main_interpreter()` function. This function takes no arguments and returns `True` if the current interpreter is the main interpreter. Any library or application that provides a custom `_thread` module must provide `_is_main_interpreter()`, just like the module’s other “private” attributes. ([gh-112826](#).)

12.2 Alterações na API C

- `Python.h` no longer includes the `<ieee.h>` standard header. It was included for the `finite()` function which is now provided by the `<math.h>` header. It should now be included explicitly if needed. Remove also the `HAVE_IEEEFP_H` macro. (Contributed by Victor Stinner in [gh-108765](#).)
- `Python.h` no longer includes these standard header files: `<time.h>`, `<sys/select.h>` and `<sys/time.h>`. If needed, they should now be included explicitly. For example, `<time.h>` provides the `clock()` and `gmtime()` functions, `<sys/select.h>` provides the `select()` function, and `<sys/time.h>` provides the `futimes()`, `gettimeofday()` and `setitimer()` functions. (Contributed by Victor Stinner in [gh-108765](#).)
- On Windows, `Python.h` no longer includes the `<stddef.h>` standard header file. If needed, it should now be included explicitly. For example, it provides `offsetof()` function, and `size_t` and `ptrdiff_t` types. Including `<stddef.h>` explicitly was already needed by all other platforms, the `HAVE_STDDEF_H` macro is only defined on Windows. (Contributed by Victor Stinner in [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);
```

(continua na próxima página)

```
Py_TRASHCAN_BEGIN(p, dealocador_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](#) introduces several changes to frame-related functions:
 - The effects of mutating the dictionary returned from `PyEval_GetLocals()` in an optimized scope have changed. New dict entries added this way will now *only* be visible to subsequent `PyEval_GetLocals()` calls in that frame, as `PyFrame_GetLocals()`, `locals()`, and `FrameType.f_locals` no longer access the same underlying cached dictionary. Changes made to entries for actual variable names and names added via the write-through proxy interfaces will be overwritten on subsequent calls to `PyEval_GetLocals()` in that frame. The recommended code update depends on how the function was being used, so refer to the deprecation notice on the function for details.
 - Calling `PyFrame_GetLocals()` in an optimized scope now returns a write-through proxy rather than a snapshot that gets updated at ill-specified times. If a snapshot is desired, it must be created explicitly (e.g. with `PyDict_Copy()`), or by calling the new `PyEval_GetFrameLocals()` API.
 - `PyFrame_FastToLocals()` and `PyFrame_FastToLocalsWithError()` no longer have any effect. Calling these functions has been redundant since Python 3.11, when `PyFrame_GetLocals()` was first introduced.
 - `PyFrame_LocalsToFast()` no longer has any effect. Calling this function is redundant now that `PyFrame_GetLocals()` returns a write-through proxy for optimized scopes.

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

P

Propostas de Melhorias do Python

- PEP 11, [8](#), [40](#)
- PEP 11#tier-2, [4](#)
- PEP 11#tier-3, [4](#)
- PEP 587, [37](#)
- PEP 590, [37](#)
- PEP 594, [3](#), [22](#)
- PEP 602, [5](#)
- PEP 626, [28](#)
- PEP 667, [3](#), [8](#), [38](#), [41](#)
- PEP 669, [4](#), [32](#)
- PEP 696, [4](#)
- PEP 699, [38](#)
- PEP 702, [4](#), [21](#)
- PEP 703, [3](#), [7](#)
- PEP 705, [4](#), [20](#)
- PEP 709, [41](#)
- PEP 719, [3](#)
- PEP 730, [4](#), [8](#)
- PEP 737, [35](#)
- PEP 738, [4](#), [8](#)
- PEP 742, [4](#), [20](#)
- PEP 744, [3](#), [4](#), [8](#)

- PYTHON_BASIC_REPL, [5](#)
- PYTHON_COLORS, [4](#), [5](#), [13](#)
- PYTHON_CPU_COUNT, [16](#)
- PYTHON_FROZEN_MODULES, [9](#)
- PYTHON_GIL, [6](#), [7](#)
- PYTHON_HISTORY, [10](#)
- PYTHON_PERF_JIT_SUPPORT, [9](#)
- PYTHONHOME, [38](#), [40](#)
- PYTHONLEGACYWINDOWSFSENCODING, [26](#), [30](#)

R

RFC

- RFC 5280, [18](#)

V

variável de ambiente

- PYTHON_BASIC_REPL, [5](#)
- PYTHON_COLORS, [4](#), [5](#), [13](#)
- PYTHON_CPU_COUNT, [16](#)
- PYTHON_FROZEN_MODULES, [9](#)
- PYTHON_GIL, [6](#), [7](#)
- PYTHON_HISTORY, [10](#)
- PYTHON_PERF_JIT_SUPPORT, [9](#)
- PYTHONHOME, [38](#), [40](#)
- PYTHONLEGACYWINDOWSFSENCODING, [26](#), [30](#)